

BIG DATA ANALYTICS

Introduction to Hadoop, Spark, and Machine-Learning



Raj Kamal | Preeti Saxena



Suitable for Big Data Analytics Course and software developers

BIG DATA ANALYTICS

BIG DATA

ANALYTICS

Introduction to Hadoop, Spark, and Machine-Learning

About the Authors

Dr Raj Kamal did his MSc at the age of 17, published his first research paper in a UK journal at the age of 18, wrote his first programme in FORTRAN that ran at ICT1904, also at the age of 18, and completed his PhD from Indian Institute of Technology Delhi at 22. He was awarded a scholarship for postgraduate studies at IIT Delhi, research fellowships of CSIR, and post-doctoral fellowships at Uppsala University, Uppsala, Sweden, (15 months) and ICTP, Trieste, Italy (3 months).

He has 51 years of research and 46 years of teaching experience, respectively. He has so far successfully guided 19 PhD scholars and is currently supervising 2 PhD scholars and research students. He has published about 94 research papers in journals and 61 conference papers of both international and national repute. He is lovingly referred by a few colleagues as the 'learning machine' and by a few others as a 'human dynamo!', perhaps because of his constant drive for understanding emerging technologies and passion for acquiring latest knowledge and its dissemination.

He has worked as a professor of seven subjects, viz. computer science, computer science and engineering, information technology, electronics, electronics and communication engineering, electrical education and physics. He has served as a faculty member in Devi Ahilya Vishwavidyalaya, Indore (25 years), Punjabi University, Patiala (17 years), Medi-Caps University, Indore (3 years), Kalasalingam University, Tamil Nadu

(2 years), and Guru Nanak Engineering College, Andhra Pradesh, and King Mongkut's Institute of Technology, Bangkok.

He is widely known for his books published by McGraw-Hill India, namely, *The Internet of Things* (2017), *Embedded Systems* (3rd Ed. 2014), *Computer Architecture* (Schaum series adaptation), and *Internet and Web Technologies* (16th Reprint 2017). His Embedded Systems book has international editions from McGraw-Hill USA, McGraw-Hill Singapore, McGraw-Hill China and McGraw-Hill South Korea.

Currently, he is working as a senior professor at Prestige Institute of Engineering Management and Research in Indore, Madhya Pradesh.

Preeti Saxena is working as an associate professor of computer science at School of Computer Science & Information Technology in Devi Ahilya Vishwavidyalaya, Indore, Madhya Pradesh. She has over 18 years of teaching experience at undergraduate and postgraduate levels. She has three years of experience in a multinational software company as a software engineer and programmer. She has published and presented 35 research papers in various international/national journals and conferences. She completed her MCA degree with distinction in 1996 from National Institute of Technology (earlier called MACT), Bhopal. She was awarded the 'Ramanujan Gold Medal - 2008' in her MTech (Computer Science). She did her PhD in mobile computing. She is currently supervising six research scholars in computer networks, mobile computing, data

analytics, augmented reality and Internet of Things (IoT). She has a passion for acquiring knowledge on emerging technologies.

BIG DATA

ANALYTICS

Introduction to Hadoop, Spark, and Machine-Learning

Raj Kamal

Senior Professor

*Prestige Institute of Engineering Management and Research
Indore, Madhya Pradesh*

Preeti Saxena

Associate Professor

*Devi Ahilya Vishwavidyalaya
Indore, Madhya Pradesh*



McGraw Hill Education (India) Private Limited

CHENNAI

McGraw Hill Education Offices

Chennai New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto



McGraw Hill Education (India) Private Limited

Published by McGraw Hill Education (India) Private Limited

444/1, Sri Ekambara Naicker Industrial Estate, Alapakkam, Porur, Chennai 600
116

Big Data Analytics

Copyright © 2019 by McGraw Hill Education (India) Private Limited.

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,

McGraw Hill Education (India) Private Limited. 1 2 3 4 5 6 7 8 9 D103074 22 21
20 19 18

2 3 4 5 6 7 8 9 D103074 22 21 20 18

Printed and bound in India.

Print-Book Edition

ISBN (13): 978-93-5316-496-6

ISBN (10): 93-5316-496-6

E-Book Edition

ISBN (13): 978-93-5316-497-3

ISBN (10): 93-5316-497-4

Managing Director: Lalit Singh

Senior Portfolio Manager: Hemant K Jha

Associate Portfolio Manager: Tushar Mishra

Senior Manager—Content Development & Production Services: Shalini Jha

Senior Content Developer: Vaishali Thapliyal

Content Developer: Malvika Shah

Production Head: Satinder S Baveja

Assistant Manager—Production: Anuj K Shriwastava

General Manager—Production: Rajender P Ghansela

Manager—Production: Reji Kumar

Information contained in this work has been obtained by McGraw Hill Education (India), from sources believed to be reliable. However, neither McGraw Hill Education (India) nor its authors guarantee the accuracy or completeness of any information published herein, and neither McGraw Hill Education (India) nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw Hill Education (India) and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at APS Compugraphics, 4G, PKT 2, Mayur Vihar Phase-III, Delhi 96, and printed at

Cover Designer: Creative Designer

Cover Printer:

Visit us at: www.mheducation.co.in

Write to us at:info.india@mheducation.com

CIN: U22200TN1970PTC111531

Toll Free Number: 1800 103 5875

Preface

Data analysis involves numerical and statistical analysis techniques, which have been widely used in many fields such as sciences, biology, research, industry, business and even sports, since the 1960s. The first author of this textbook, Raj Kamal, himself used analytical techniques in the 1970s for obtaining solutions using matrix multiplication, inversion, transpose, determinants, linear equations, simultaneous equations using matrices and least square fitting for finding parameters from observed data points, which can be described theoretically by superimposition of the functions. His first programme was in FORTRAN that ran on ICT1904 in 1967. A classic book, *Numerical Methods for Scientists and Engineers*, Richard W. Hamming, McGraw-Hill, New York, 1973 (Available at the ACM digital library), fuelled his interest in the field of analytics since then. Both the authors learned from excellent lessons on Big Data Analytics and Advanced Big Data Analytics given by Ching-Yung Lin, PhD and adjunct professor at Departments of Electrical Engineering and Computer Science, Columbia University USA, in 2017. It is here that an idea of writing a textbook on Big Data Analytics for young minds came to the authors.

The chess match of the legendary Garry Kasparov in 1997 against the IBM supercomputer, Deep Blue, is a landmark moment in the history of computing technology. "It was the dawn of a new era in artificial intelligence: a machine capable of beating the reigning human champion at this most cerebral game", in the words of Garry himself. Nowadays, data analytics, decisions, predictions and discovery of new knowledge, are possible with the use of AI techniques of machine learning and deep learning. The rise in technology has led to production and storage of voluminous amount of data. Earlier, megabytes (10^6 B) were used and now petabytes (10^{15} B) plus are used for processing, analysis, predicting, decisions, discovering facts and generating new knowledge. Big Data storage, processing and analysis, face challenges from large growth in volume of data, variety of data, various forms and formats, increasing complexity, fast generation of data and the need to quickly process, analyse and use data.

Many applications such as industry reports, financial reports, social network and social media, cloud applications, public and commercial websites, scientific experiments, simulators, sensors in Internet of Things, and e-services generate Big Data. Big Data Analytics (BDA) finds applications in many areas, such as healthcare, medicine, advertising, marketing, sales, and tracing anomalies in big data in these disciplines.

This textbook explains the concepts of BDA in a simple to complex manner. For example, it uses the popular Ravensburger Beneath the Sea Jigsaw Puzzle (5000 pieces) in an example to show that scaling out

and division of the computations along with data works well in parallel processing shared-nothing architecture at distributed computing nodes. This student-friendly textbook has a number of illustrations, sample codes, case studies and real-life analytics for datasets such as toys, chocolates, cars, students' GPAs and academic performance.

Classic Apache-based Hadoop ecosystem tools and the latest Apache Spark ecosystem tools deploying the Python libraries for analytics have been described in depth.

Readers

This textbook is an extremely useful asset for national as well as international students of Big Data Analytics. This book caters to the needs of undergraduate and postgraduate students of computer science and engineering, information technology, and related disciplines, along with professionals in the industry for developing innovative Big Data Analytics solutions based on Spark ecosystem tools with Python libraries, which include the use of machine-learning concepts.

The book will also be a useful guide in training programmes for Big Data architects and analytics requiring new skills, and for those who wish to learn the latest topics.

The main features of the book are:

- Easy-to-understand and student-friendly content, which includes illustrative figures, examples and sample codes
- The book explains architecture, storage and programming methods for Big Data analytics, while keeping multidisciplinary undergraduate and postgraduate students as primary readers in mind
- Learning objectives for each section, recall from previous chapters and introduction along with meanings of important key terms have been provided in the beginning of each chapter
- Self-assessment questions, classified into three difficulty levels, have been given at the end of each section in a chapter
- Key concepts covered, learning outcomes, objective questions, review questions and practice exercises have been provided towards the end of the textbook.

Roadmap for BDA Readers

The author presumes the readers possess basic intellectual and academic background in mathematical and statistical methods, cloud platform for storage and applications (such as Amazon S3), object oriented programming, familiarity and programme-writing skills in Java, and the knowledge of Python libraries.

Readers intending to learn and use Hadoop ecosystem tools need to study chapters 1 to 4 and 6 to 9. They need prerequisite programme-writing skills in Java. Readers intending to learn and use the latest Spark ecosystem tool need to study chapters 1, 3 and 5 to 10. They need familiarity and programme-writing skills in Java and Python libraries.

Learning and Assessment Tools

Learning Objectives

Design of this book follows Learning Objective (LO) – oriented approach. This educational process emphasises on developing required skills amongst the students. The process tests the outcomes of the study of a course, as opposed to routine learning. This approach creates an ability to acquire knowledge and apply fundamental principles to analytical problems and applications.

Self-Assessment Exercises

Each learning objective is followed by a set of questions for self-assessment. This offers great retention of concepts.

Pedagogical Classification

The pedagogy is arranged as per levels of difficulty—all checkpoint problems are linked with Learning Objectives (LOs) and marked with Levels of Difficulty (LOD), to help assess students' learning. These levels of difficulty have been derived as per Bloom's taxonomy.

- indicates Level 1 and Level 2, i.e. knowledge and comprehension-based easy-to-solve problems.
- indicates Level 3 and Level 4, i.e. application and analysis-based medium to difficult problems.
- indicates Level 5 and Level, 6 i.e. synthesis and evaluation-based very difficult problems.

Learning Outcomes

Summary points specific to each LO are provided at the end of each chapter. This helps in recapitulating the ideas initiated with the outcomes achieved.

Chapter-end Exercises

More than 300 carefully designed chapter-end questions and exercises are arranged as per levels of difficulty, and are framed to enhance knowledge and test new skills learnt. These include objective type multiple choice questions, review questions and practice exercises.

Salient Features

- **Extensive coverage** of topics in Big Data Analytics, such as Big Data NoSQL Column-family, Object and Graph databases, Data reporting and visualization, Programming with open source Big Data Hadoop ecosystems tools, Spark, Spark ecosystem, Streaming, GraphX, and

Mahout tools, have been explained using examples of datasets of interest to students, such as toys, chocolates, cars and GPAs/academic performance of students in theory and practical subjects.

- **Latest topics** such as Machine Learning, Regression analysis, K-NN, Predictive analytics, Clustering, Decision trees, Clusters, and Similar, frequent item sets, Pattern mining solutions, Classifiers, Recommenders, Real-time streaming data analytics, Graph networks for web and social network analytics, and Text analytics.
- **Systematic approach:** Data architecture is followed by Analytics architectures, and the section on Hadoop ecosystem tools is followed by Spark- and Python-based tools. Each chapter starts with learning objectives and a quick recall from earlier chapters. The introduction is followed by important key terms in the beginning of each chapter for easy understanding of the chapter content. The text has been tagged with descriptions and questions, self-assessment exercises and illustrations within the chapter, and each chapter ends with learning outcomes, MCQs, review questions and practice exercises.
- **Rich pedagogy:** 20+ programming codes, 100+ questions, solved examples and practice exercises. Dedicated chapter on a major case study in the textbook, and another major case study in online content. Rich online content, PPTs, guide to solutions of practice exercises and list of select books and references, which makes a comprehensive bibliography for anyone interested in pursuing further studies in Big Data Analytics.

Chapter Organization

Chapter 1 gives overview of Big Data, characteristics, types and classification methods. It describes scalability, need of scaling up and scaling out of processing, analytics using massively parallel processors, and cloud, grid and distributed computing. This chapter introduces data architecture design, data management, data sources, data quality, data pre-processing and export of pre-processed data stores to cloud. Approaches of traditional systems, such as SQL, Relational Database Management System (RDBMS), enterprise servers and data warehouse for data storage and analysis, as well as the approaches for Big Data storage, processing and analytics have been explained in detail. It also includes Berkley Data Analytics architecture, and introduces cases, case studies and

applications of BDA to its readers.

Chapter 2 starts with an interesting example, explaining the distributed parallel computing architecture with shared-nothing architecture. This chapter describes basics of Hadoop, its ecosystem components, streaming and pipe functions, Hadoop physical architecture, Hadoop distributed file system (HDFS). It explains how to organize nodes for computations using large-scale file systems, and provides a conceptual understanding of MapReduce Daemon, functioning of Hadoop MapReduce framework, YARN for managing resources along with the application tasks. The chapter introduces Hadoop ecosystem interactions, and application support for analytics using AVRO, Zookeeper, Ambari, HBase, Hive, Pig and Mahout.

Chapter 3 highlights NoSQL data stores, solutions, schema-less models and increasing flexibility of NoSQL for data manipulation. It describes NoSQL data architecture patterns, namely the key value pairs, graphs, column family, tabular, document and object in the data stores. This chapter explains the use of the shared-nothing architecture, choosing a distribution model, master-slave versus peer-to-peer, and four ways which NoSQL handles Big Data problems. The chapter covers MongoDB and Cassandra databases.

Chapter 4 describes the MapReduce paradigm, map tasks using key-value pairs, grouping-by-keys and reduce tasks. It provides the conceptual understanding of partitioning and combiners in the application execution framework, and MapReduce algorithms by stating various examples. The chapter also describes Hive, HiveQL and Pig architecture, Grunt shell commands, data model, Pig Latin. It provides an understanding how to develop scripts and User-Defined Functions.

Chapter 5 introduces Spark architecture features, software stack components and their functions. It describes the steps in data analysis with Spark, and usage of Spark with Python advanced features. The highlight of this chapter is the description of methods of downloading Spark, programming with the RDDs, usage of the Spark shell, developing and testing Spark codes, and the applications of MLlib. The chapter gives understanding of how to run ETL processes using the built-in functions, operators and pipelines. It also covers data analytics, data reporting and data visualization aspects.

Chapter 6 lucidly explains the classes of variables, and the ways of estimating the relationships, outliers, variances, probability distributions, errors and correlations between variables, items and entities. The chapter gives detailed descriptions of regression analysis, and the use of K-NN distance measures for making predictions using interpolations and extrapolations. It explains machine-learning methods of finding similar items, similarities, filtering of similars, frequent itemset mining, collaborative filtering, associations and association rules mining. The highlight of this chapter is the description of ML methods of clustering, classifiers and recommenders, and Apache Mahout algorithms for big datasets.

Chapter 7 provides understanding of the concept, model, architecture, management of data streams. It describes stream sources and stream computing aspects – sampling, filtering, counting distinct elements, frequent itemset stream analytics, handling of large datasets, and mining of association rules. The chapter explains the real-time analytics platform, Apache SparkStreaming, and case studies on real-time sentiment analytics and stock price analytics.

Chapter 8 describes the modelling of databases as the graphs and representations of graphs using triples. The highlight of this chapter is the description of use of graphs and graph networks. The chapter gives methods of choosing the graph and graph parameters,

such as centralities for analytics. It explains the graph methods of diagnostics, decisions, StatsModel, and probabilities-based analytics. Another highlight is the description of features of Apache Spark GraphX, and its architecture, components and applications.

Chapter 9 describes text mining and the usage of ML techniques . Naive-Bayes analysis, and support-vector machines (SVMs) for analysing text. The chapter explains the methods of web mining, link analytics, analysing of web graphs, PageRank methods, web structure analytics, finding hubs and communities, social-network analysis, and representation of social networks as graphs. It describes computational methods of finding the clustering in social network graphs, SimRank, counting triangles (cliques) and discovering the communities.

Chapter 10 describes installation methods for Hadoop, Hive, Pig and Spark on the Ubuntu platform. The highlight of this chapter is deploying and exploring open-source Lego datasets, schema, processing and storage. The chapter explains MapReduce implementation for counting items in a dataset, creating Hive data tables from a CSV format dataset, and creating DataFrames from RDDs. It describes Hive and PySpark programmes using functions for *Merge and Join* of DataFrames, the SQL-equivalent Join, and the UDFs for customised query processing. The chapter explains programmes for data visualization using pi, bar and scatter plots. Another highlight of the chapter is the description of machine learning programmes using sklearn for SVMs, Naive Bayes Classifiers, linear and polynomial regression analyses, and predictive analytics.

Following content is available towards the end of this textbook:

1. Solution to objective questions
2. Bibliography
 - Printed and e-books
 - Website resources
 - Research journals
 - Reference papers

Online Learning Center

An accompanying web supplement available at <http://www.mhhe.com/kamal/bda> includes:

- PowerPoint slides for each chapter to supplement lecture presentations
- Solution guide to practice exercises
- Write-up on topics
- An additional case study using an open source large dataset of car company

Although much care has been taken to ensure an error-free text, a few mistakes may

have crept in. The authors shall be grateful if they are pointed out by the readers. Feedback on content of the book as well as the web supplement available on the McGraw-Hill site from readers will be highly appreciated through e-mail to dr_rajkamal@hotmail.com and preeti_ms@rediffmail.com.

RAJ KAMAL

PREETI SAXENA

Acknowledgements

Raj Kamal is grateful to Chairman, Dr N N Jain, Vice Chairman, Dr Davis Jain, and Shri Ketan Jain of Prestige Educational Society for providing him an opportunity to serve at PIEMR, a great institution and platform to learn, and continue research and teaching in new and exciting areas of engineering and technology. The author is also grateful to Dr Manojkumar Deshpande, Director, PIEMR, for his continuous encouragement.

Raj Kamal is grateful to Sushil Mittal (wife) for her love and being a constant source of support, and his family members, Dr Shilpi Kondaskar (daughter), Dr Atul Kondaskar (son-in-law), Shalin Mittal (son) Needhi Mittal (daughter-in-law), and grandchildren Arushi Kondaskar, Atharv Raj Mittal, Shruti Shreya Mittal and Ishita Kondaskar for their love and affection.

Preeti Saxena extends her thanks to family members, Manish Saxena (husband) for unconditional support and everlasting faith, Devansh Saxena (son) and Raghvi Saxena (daughter) for their warmth and affection, Suvidya Saxena (mother) and Prateek Saxena (brother) for their encouragement and well wishes.

Both the authors are also grateful to their colleagues, especially Dr Suresh Jain, Dr (Mrs) Maya Ingle,

Dr Sanjay Tanwani, Dr Shraddha Masih, Dr Archana Chaudhary, Dr Manju Chattopadhaya, Ms Kirti Panwar Bhati, Dr Savita Kolhe and Ms Pritika Bahad for their support and continuous encouragement.

The authors would also like to thank the team at McGraw-Hill Education who initiated the idea of writing, perhaps the first textbook on Big Data Analytics and whose agile approach lead to the timely production of this textbook. The authors also thank the reviewers who took out time to go through the manuscript and shared their valuable suggestions.

Contents

Preface

Acknowledgements

List of Acronyms

1. Introduction to Big Data Analytics

1.1 Introduction

1.1.1 Need of Big Data

1.2 Big Data

1.2.1 Classification of Data—Structured, Semi-structured and Unstructured

1.2.2 Big Data Definitions

1.2.3 Big Data Characteristics

1.2.4 Big Data Types

1.2.5 Big Data Classification

1.2.6 Big Data Handling Techniques

1.3 Scalability and Parallel Processing

1.3.1 Analytics Scalability to Big Data

1.3.2 Massively Parallel Processing Platforms

1.3.3 Cloud Computing

1.3.4 Grid and Cluster Computing

1.3.5 Volunteer Computing

1.4 Designing Data Architecture

1.4.1 Data Architecture Design

1.4.2 Managing Data for Analysis

1.5 Data Sources, Quality, Pre-Processing and Storing

1.5.1 Data Sources

1.5.2 Data Quality

1.5.3 Data Pre-processing

1.5.4 Data Store Export to Cloud

1.6 Data Storage and Analysis

1.6.1 Data Storage and Management: Traditional Systems

1.6.2 Big Data Storage

1.6.3 Big Data Platform

1.6.4 Big Data Analytics

1.7 Big Data Analytics Applications and Case Studies

1.7.1 Big Data in Marketing and Sales

1.7.2 Big Data and Healthcare

1.7.3 Big Data in Medicine

1.7.4 Big Data in Advertising

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

2. Introduction to Hadoop

2.1 Introduction

2.2 Hadoop and its Ecosystem

2.2.1 Hadoop Core Components

- 2.2.2 Features of Hadoop
 - 2.2.3 Hadoop Ecosystem Components
 - 2.2.4 Hadoop Streaming
 - 2.2.5 Hadoop Pipes
- 2.3 Hadoop Distributed File System
- 2.3.1 HDFS Data Storage
 - 2.3.2 HDFS Commands
- 2.4 Mapreduce Framework and Programming Model
- 2.4.1 Hadoop MapReduce Framework
 - 2.4.2 MapReduce Programming Model
- 2.5 Hadoop Yarn
- 2.5.1 Hadoop 2 Execution Model
- 2.6 Hadoop Ecosystem Tools
- 2.6.1 Hadoop Ecosystem
 - 2.6.2 Ambari
 - 2.6.3 HBase
 - 2.6.4 Hive
 - 2.6.5 Pig
 - 2.6.6 Mahout

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

3. NoSQL Big Data Management, MongoDB and Cassandra

- 3.1 Introduction
 - 3.2 NoSQL Data Store
 - 3.2.1 NoSQL
 - 3.2.2 Schema-less Models
 - 3.2.3 Increasing Flexibility for Data Manipulation
 - 3.3 NoSQL Data Architecture Patterns
 - 3.3.1 Key-Value Store
 - 3.3.2 Document Store
 - 3.3.3 Tabular Data
 - 3.3.4 Object Data Store
 - 3.3.5 Graph Database
 - 3.3.6 Variations of NoSQL Architectural Patterns
 - 3.4 NoSQL to Manage Big Data
 - 3.4.1 Using NoSQL to Manage Big Data
 - 3.5 Shared-Nothing Architecture for Big Data Tasks
 - 3.5.1 Choosing the Distribution Models
 - 3.5.2 Ways of Handling Big Data Problems
 - 3.6 MongoDB Database
 - 3.7 Cassandra Databases
- Key Concepts
- Learning Outcomes
- Objective Type Questions
- Review Questions
- Practice Exercises

4. MapReduce, Hive and Pig

- 4.1 Introduction
- 4.2 MapReduce Map Tasks, Reduce Tasks and MapReduce Execution
 - 4.2.1 Map-Tasks
 - 4.2.2 Key-Value Pair
 - 4.2.3 Grouping by Key
 - 4.2.4 Partitioning
 - 4.2.5 Combiners
 - 4.2.6 Reduce Tasks
 - 4.2.7 Details of MapReduce Processing Steps
 - 4.2.8 Coping with Node Failures
- 4.3 Composing MapReduce for Calculations and Algorithms
 - 4.3.1 Composing Map-Reduce for Calculations
 - 4.3.2 Matrix–Vector Multiplication by MapReduce
 - 4.3.3 Relational–Algebra Operations
 - 4.3.4 Matrix Multiplication
- 4.4 Hive
 - 4.4.1 Hive Architecture
 - 4.4.2 Hive Installation
 - 4.4.3 Comparison with RDBMS (Traditional Database)
 - 4.4.4 Hive Data Types and File Formats
 - 4.4.5 Hive Data Model
 - 4.4.6 Hive Integration and Workflow Steps
 - 4.4.7 Hive Built-in Functions
- 4.5 HiveQL
 - 4.5.1 HiveQL Data Definition Language (DDL)
 - 4.5.2 HiveQL Data Manipulation Language (DML)

- 4.5.3 HiveQL For Querying the Data
 - 4.5.4 Aggregation
 - 4.5.5 Join
 - 4.5.6 Group by Clause
- 4.6 Pig
 - 4.6.1 Apache Pig – Grunt Shell
 - 4.6.2 Installing Pig
 - 4.6.3 Pig Latin Data Model
 - 4.6.4 Pig Latin and Developing Pig Latin Scripts

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

5. Spark and Big Data Analytics

- 5.1 Introduction
- 5.2 Spark
 - 5.2.1 Introduction to Big Data Tool—Spark
- 5.3 Introduction to Data Analysis with Spark
 - 5.3.1 Spark SQL
 - 5.3.2 Using Python Advanced Features with Spark SQL
 - 5.3.3 Data Analysis Operations
- 5.4 Downloading Spark, and Programming using RDDs and MLIB
 - 5.4.1 Downloading, Installing Spark and Getting Started
 - 5.4.2 Programming with RDDs

- 5.4.3 Machine Learning with MLlib
- 5.5 Data ETL (Extract, Transform and Load) Process
 - 5.5.1 Composing Spark Program Steps for ETL
- 5.6 Introduction to Analytics, Reporting and Visualizing
 - 5.6.1 Introduction to Analytics
 - 5.6.2 Data/Information Reporting
 - 5.6.3 Data Visualization

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

6. Machine Learning Algorithms for Big Data Analytics

- 6.1 Introduction
- 6.2 Estimating the Relationships, Outliers, Variances, Probability Distributions and Correlations
 - 6.2.1 Relationships—Using Graphs, Scatter Plots and Charts
 - 6.2.2 Estimating the Relationships
 - 6.2.3 Outliers
 - 6.2.4 Variance
 - 6.2.5 Probabilistic Distribution of Variables, Items or Entities
 - 6.2.6 Correlation
- 6.3 Regression Analysis
 - 6.3.1 Simple Linear Regression
 - 6.3.2 Least Square Estimation
 - 6.3.3 Multiple Regressions

- 6.3.4 Modelling Possibilities using Regression
- 6.3.5 Predictions using Regression Analysis
- 6.3.6 K–Nearest–Neighbour Regression Analysis
- 6.4 Finding Similar Items, Similarity of Sets and Collaborative Filtering
 - 6.4.1 Finding Similar Items
 - 6.4.2 Jaccard Similarity of Sets
 - 6.4.3 Collaborative Filtering as a Similar–Sets Finding Problem
 - 6.4.4 Distance Measures for Finding Similar Items or Users
- 6.5 Frequent Itemsets and Association Rule Mining
 - 6.5.1 Frequent Itemset Mining
 - 6.5.2 Association Rule— Overview
 - 6.5.3 Apriori Algorithm
 - 6.5.4 Evaluation of Candidate Rules
 - 6.5.5 Applications of Association Rules
- 6.6 Clustering Analysis
 - 6.6.1 Overview of Clustering
 - 6.6.2 K–Means
 - 6.6.3 Hierarchical Clustering
- 6.7 Classification
 - 6.7.1 Concept of Classification
 - 6.7.2 K–Nearest Neighbour Classifier
 - 6.7.3 Stochastic Gradient Descent Method – Logistic Regression
 - 6.7.4 Decision Tree Algorithm
 - 6.7.5 Naïve–Bayes Theorem – Naïve Bayes Classifier
 - 6.7.6 Support Vector Machine Classifier
 - 6.7.7 Random Forest Classifier

- 6.7.8 AdaBoost and Other Ensemble Classifiers
 - 6.8 Recommendation System
 - 6.8.1 Collaborative Recommendation
 - 6.8.2 Model for Recommendation Systems
 - 6.8.3 Content Based Recommendation
 - 6.8.4 Knowledge-based Recommendation
 - 6.8.5 Hybrid Recommendation Approaches
 - 6.9 Apache Mahout Machine-Learning Applications
 - Key Concepts
 - Learning Outcomes
 - Objective Type Questions
 - Review Questions
 - Practice Exercises
7. Data Stream Mining and Real-Time Analytics Platform—**SparkStreaming**
- 7.1 Introduction
 - 7.2 Data Stream Concepts and Data Stream Management
 - 7.2.1 Data Stream Concepts
 - 7.2.2 Data Stream Model
 - 7.2.3 Architecture
 - 7.2.4 Data Stream Management System (DSMS)
 - 7.2.5 Example of Sources of Streams
 - 7.2.6 Stream Queries
 - 7.2.7 Stream Processing Issues
 - 7.2.8 Real-time Processing, Stream Processing and Batch Processing

7.2.9 Summarizing Streaming Processing Needs

7.3 Stream Computing Aspects

7.3.1 Stream Computing

7.3.2 Sampling Data in a Stream

7.3.3 Filtering of Stream

7.3.4 Counting Distinct Elements in a Stream

7.3.5 Estimating Moments

7.3.6 Counting of 1's in a Window

7.3.7 Decaying Windows

7.4 Frequent Itemsets

7.4.1 Finding Frequent Itemsets

7.4.2 Handling Large Datasets for Finding Frequent Itemsets

7.4.3 Limited Passes Algorithms

7.4.4 Counting Frequent Items in a Stream

7.5 Real-time Analytics Platform (RTAP)—SparkStreaming

7.5.1 Apache® Spark™ Streaming

7.5.2 Real-Time Analytics Platform Applications

7.5.3 Case Studies—Real-Time Sentiment Analysis, Positive Negative Sentiments Prediction and Stock Market Predictions

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

8. Graph Analytics for Big Data and Spark GraphX Platform

- 8.1 Introduction**
 - 8.2 Graph Model**
 - 8.2.1 Representing a Graph as Triples**
 - 8.2.2 Resource Description Framework (RDF) for Graph Databases**
 - 8.2.3 SPARQL Querying Language for RDF Graph–Database**
 - 8.2.4 NativeDB Graph Database**
 - 8.2.5 Property Graph Model**
 - 8.3 Graphs, Network Organization and Graph Analytics**
 - 8.3.1 Network Organization**
 - 8.3.2 Probabilistic Graphical Network Organizations—Bayesian and Markov Networks**
 - 8.3.3 Graph Analytics**
 - 8.3.4 Choosing Graph Analytics**
 - 8.3.5 Use Cases of Graph Analytics**
 - 8.4 Graph Analytics Algorithms and Approaches**
 - 8.4.1 StatsModel and Probability Based Analytics**
 - 8.4.2 Technical Complexity in Analyzing Graphs**
 - 8.5 Spark GraphX Platform**
 - 8.5.1 Features of a Graph Analytics Platform—Apache SparkGraphX**
 - 8.5.2 Dedicated Appliances for Graph**
- Key Concepts**
- Learning Outcomes**
- Objective Type Questions**
- Review Questions**
- Practice Exercises**

9. Text, Web Content, Link, and Social Network Analytics

9.1 Introduction

9.2 Text Mining

9.2.1 Text Mining

9.2.2 Naïve Bayes Analysis

9.2.3 Support Vector Machines

9.3 Web Mining, Web Content and Web Usage Analytics

9.3.1 Web Mining

9.3.2 Web Content Mining

9.3.3 Web Usage Mining

9.4 Page Rank, Structure of Web and Analyzing a Web Graph

9.4.1 Page Rank Definition

9.4.2 Web Structure

9.4.3 Computation of PageRank and PageRank Iteration

9.4.4 Topic Sensitive PageRank and Link Spam

9.4.5 Hubs and Authorities

9.4.6 Web Communities

9.4.7 Limitations of Link, Rank and Web Graph Analysis

9.5 Social Networks as Graphs and Social Network Analytics

9.5.1 Social Network as Graphs

9.5.2 Social Graph Network Topological Analysis using Centralities and PageRank

9.5.3 Social Graph Network Analysis using K-core and Neighbourhood Metrics

9.5.4 Clustering in Social Network Graphs

9.5.5 SimRank

- 9.5.6 Counting Triangles and Graph Matches
- 9.5.7 Using SparkGraph (Map–Reduce) for Network Graphs
- 9.5.8 Direct Discovery of Communities

Key Concepts

Learning Outcomes

Objective Type Questions

Review Questions

Practice Exercises

10. Programming Examples in Analytics and Machine Learning using Hadoop, Spark and Python

10.1 Introduction

10.2 Installation Steps for Hadoop and Spark

- 10.2.1 Installation Steps for Hadoop, Hive and Pig
- 10.2.2 Installation Steps for the Spark on Ubuntu
- 10.2.3 Computing Platform Configuration

10.3 Datasets Used in the Examples, Data Deployment and Exploration

- 10.3.1 Counting and Sorting of Items in Datasets using MapReduce
- 10.3.2 Storing CSV Dataset into Hive Database
- 10.3.3 Storing CSV Dataset into the Spark DataFrame
- 10.3.4 Creating DataFrame from the RDD

10.4 Programming Steps using Hive and Pyspark

- 10.4.1 Merge and Join Functions for DataFrame Objects
- 10.4.2 Analysis and Query-Processing Using UDFs in Hive and Pyspark

10.5 Data Visualization using Python Plotting Library

10.6 Machine-Learning Algorithms Implementation

10.6.1 Clustering Algorithm

10.6.2 Classification Algorithm Example 1: SVM Classifier

10.6.3 Classification Algorithm Example 2: Naïve Bayes Classifier

10.6.4 Regression Analysis Algorithms

Practice Exercises

Bibliography

Answers to Multiple Choice Questions

Index

List of Acronyms

3V	Volume, Velocity and/or Variety
4V	Volume, Velocity, Variety and Veracity
ACID	Atomicity, Consistency, Isolation and Durability
ACPAMS	Automotive Components and Predictive Automotive Maintenance Service
ACVM	Automatic Chocolate Vending Machine
ADT	Abstract Data Type
AI	Artificial Intelligence
AM	Application Master
AMP	Algorithms, Machines and Peoples Laboratory at University of Berkeley
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
API	Application Programming Interface
BASE	Basically Available Soft State Eventual Consistency
BFS	Breadth-First Search
BI	Business Intelligence
BIRT	Business Intelligence and Reporting Tools
BLOB	Binary Large Object
BN	Bayesian Network
BNG	Bayesian Network Graph
BP	Business Process
BSON	Binary encoded JSON objects
CAC	Customer Acquisition Cost
CAP	Consistence Availability and Partitioning
CGPA	Cumulative Grade-Point Average
CF	Collaborative Filtering
CBF	Content Based Filtering
CLI	Command Line interface

CLTV	Customer Lifetime Value
CQL	Cassandra Query Language
CRM	Customer Relationship Management
CSV	Comma-Separated Values
CURD	Create, Update, Read and Delete
CV	Customer Value
CVA	Customer Value Analytics
DALS	Distributed regularized Alternating Least Squares
DAS	Direct Attached Storage
DFG	Data Flow Graph
DAG	Directed Acyclic Graph
DB	Database
DBA	Database Administrator
DBMS	Database Management System
DDL	Data Definition Language
DFS	Distributed File System
DL	Deep Learning
DF	Data Flow
DFG	Data Flow Graph
DML	Data Manipulation Language
DRM	Distributed Row Matrix
DSMS	Data Stream Management System
DSPCA	Distributed Stochastic Principal Component Analysis
EDP	Electronic Data Processing
ELT	Extract, Load and Transform
ERP	Enterprise Resource Planning
ETL	Extract, Transform and Load
FIM	Frequent Item-set Mining
FL	Flavour (for example of chocolate, candy, ice-cream)
FP	Frequent Pattern
GP	Grade Point
GVUDF	Grouped Vectorized User Defined Function
IDF	Inverse Document Frequency
IIS	IBM InfoSphere Information Server

IR	Information Retrieval
IS	Information Server or Service or System
JDBC	Java Database Connectivity
JN	Journal Node
JSON	JavaScript Object Notation
HITS	Hypertext-Induced Topic Selection
HDFS	Hadoop Distributed File System
HPQS	High Performance Query Structure
IDE	Integrated Development Environment
10	Input and Output

Chapter 1

Introduction to Big Data Analytics

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 1.1 Get conceptual understanding of data and web data; classification of data as structured, semi-, multi- and unstructured data; Big Data characteristics, types, classifications and handling techniques
- LO 1.2 Get conceptual understanding of scalability, Massively Parallel Processing (MPP), distributed, cloud and grid computing
- LO 1.3 Know the design layers in data-processing architecture for the data management and analytics
- LO 1.4 Get introduced to data sources, data quality, data pre-processing, and the export of data store (such as tables, objects and files) to the cloud
- LO 1.5 Get conceptual understanding of data storage and analysis; comparison between traditional systems such as Relational Database Management System (RDBMS), enterprise servers, data warehouse and approaches for Big Data storage and analytics
- LO 1.6 Get knowledge of use cases and applications of Big Data in various fields.

1.1 | INTRODUCTION

Two Grand Masters, Magnus Carlsen and Sergey Karjakin, played the final in World Chess Championship held on December 1, 2016. Magnus Carlsen won this final and the

title of Grand Master. Sergey Karjakin, in order to win, would have to design a new strategy to defeat Carlsen and other players next year. A Grand Master typically studies the moves made in earlier matches played by Grand Masters, analyzes them and then designs his strategies. Evolving strategy to defeat an opponent could even make good use of the data of Gary Kasparov's matches from 1984. Study and analysis of a large number of matches helps in evolving a winning strategy. Similarly, analytics of Big Data could enable discovery of new facts, knowledge and strategy in a number of fields, such as manufacturing, business, finance, healthcare, medicine and education.

1.1.1 Need of Big Data

The rise in technology has led to the production and storage of voluminous amounts of data. Earlier megabytes (10^6 B) were used but nowadays petabytes (10^{15} B) are used for processing, analysis, discovering new facts and generating new knowledge. Conventional systems for storage, processing and analysis pose challenges in large growth in volume of data, variety of data, various forms and formats, increasing complexity, faster generation of data and need of quickly processing, analyzing and usage.

Figure 1.1 shows data usage and growth. As size and complexity increase, the proportion of unstructured data types also increase.

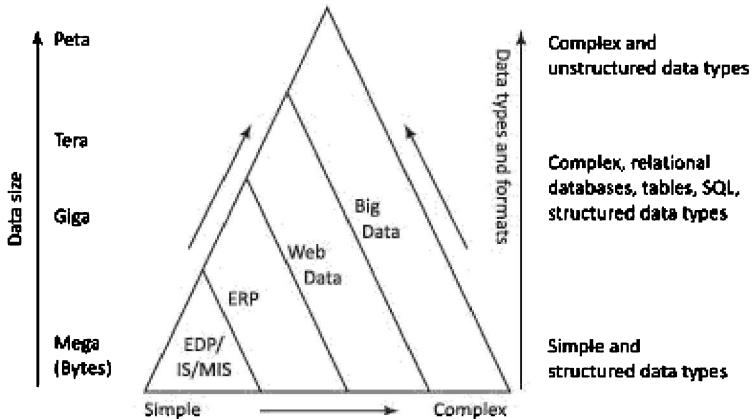


Figure 1.1 Evolution of Big Data and their characteristics

An example of a traditional tool for structured data storage and querying is RDBMS. Volume, velocity and variety (3Vs) of data need the usage of number of programs and tools for analyzing and processing at a very high speed. When integrated with the Internet of Things, sensors and machines data, the veracity of data is an additional V. (Section 1.2.3)

Big Data requires new tools for processing and analysis of a large volume of data. For

example, unstructured, NoSQL (not only SQL) data or Hadoop compatible system data.

Following are selected key terms and their meanings, which are essential to understand the topics discussed in this chapter:

Application means application software or a collection of software components. For example, software for acquiring, storing, visualizing and analyzing data. An *application* performs a group of coordinated activities, functions and tasks.

Application Programming Interface (API) refers to a software component which enables a user to access an application, service or software that runs on a local or remote computing platform. An API initiates running of the application on receiving the message(s) from the user-end. An API sends the user-end messages to the other-end software. The other-end software sends responses or messages to the API and the user.

Data Model refers to a map or schema, which represents the inherent properties of the data. The map shows groupings of the data elements, such as records or tables, and their associations. A model does not depend on software using that data.

Data Repository refers to a collection of data. A data-seeking program relies upon the data repository for reporting. The examples of repositories are database, flat file and spreadsheet. [Repository in English means a group which can be relied upon to look for required things, such as special information or knowledge. For example, a repository of paintings by various artists.]

Data Store refers to a data repository of a set of objects. Data store is a general concept for data repositories, such as database, relational database, flat file, spreadsheet, mail server, web server and directory services. The objects in data store model are instances of the classes which the *database schemas* define. A data store may consist of multiple schemas or may consist of data in only one schema. Example of only one scheme for a data store is a relational database.

Distributed Data Store refers to a data store distributed over multiple nodes. Apache Cassandra is one example of a distributed data store. (Section 3.7)

Database (DB) refers to a grouping of tables for the collection of data. A table ensures a systematic way for accessing, updating and managing data. A database pertains to the applications, which access them. A *database* is a repository for querying the required information for analytics, processes, intelligence and knowledge discovery. The databases can be distributed across a network consisting of servers and data warehouses.

Table refers to a presentation which consists of row fields and column fields. The values at the fields can be number, date, hyperlink, image, object or text of a document.

Flat File means a file in which data cannot be picked from in between and must be read

from the beginning to be interpreted. A file consisting of a single-table file is called a flat file. An example of a flat file is a csv (comma-separated value) file. A flat file is also a data repository.

Flat File Database refers to a database in which each record is in a separate row unrelated to each other.

CSV File refers to a file with comma-separated values. For example, CS101, “Theory of Computations”, 7.8 when a student’s grade is 7.8 in subject code CS101 and subject “Theory of Computations”.

Name-Value Pair refers to constructs used in which a field consists of name and the corresponding value after that. For example, a name value pair is *date*, ““Oct. 20, 2018””, *chocolates_sold*, 178;

Key-Value Pair refers to a construct used in which a field is the key, which pairs with the corresponding value or values after the key. For example, consider a tabular record, ““Oct. 20, 2018””; ““chocolates_sold””, 178. The *date* is the primary key for finding the date of the record and *chocolates_sold* is the secondary key for finding the number of chocolates sold.

Hash Key-Value Pair refers to the construct in which a hash function computes a key for indexing and search, and distributing the entries (key/value pairs) across an array of slots (also called buckets). (Section 3.3.1)

Spreadsheet refers to the recording of data in fields within rows and columns. A field means a specific column of a row used for recording information. The values in fields associates a program, such as Microsoft Excel 2013. An example of a spreadsheet application is *accounting*. The application manages, analyzes and enables new values either directly or using formulae which contain the relationships of a field with cells and rows. Examples of functions are SUMIF and COUNTIF, delete duplicate entries, sort using multiple keys, filter single or multiple columns, create a filter using filtering criteria or rules for multi-fields, and create top-n lists for values or percentages.

Stream Analytics refers to a method of computing continuously, i.e. even while events take place data flows through the system.

Database Maintenance (DBM) refers to a set of tasks which improves a database. DBM uses functions for improving performance (such as by query planning and optimization), freeing-up storage space, updating internal statistics, checking data errors and hardware faults.

Database Administration (DBA) refers to the function of managing and maintaining Database Management System (DBMS) software regularly. A database administering personnel has many responsibilities, such as installation, configuration, database design, implementation upgrading, evaluation of database features, reliable

backup and recovery methods for the database.

Database Management System (DBMS) refers to a software system, which contains a set of programs specially designed for *creation* and *management* of data stored in a database. Transactions can be performed with database relational database.

Relational Database is a collection of data into multiple tables, which relate to each other through special fields, called keys (primary key, foreign key and unique key). Relational databases provide flexibility.

Relational Database Management System (RDBMS) refers to a software system used for creation of relational databases and management of data which are stored in a relational database. RDBMS functions perform the *transactions* on the relational database. Examples of RDBMS are MySQL, PostGreSQL (Oracle database created using PL/SQL) and Microsoft SQL server using T-SQL.

Transaction (trans + action) means two interrelated sets of operations, actions or instructions. A transaction is a set of actions which accesses, changes, updates, appends or deletes various data. A command ‘connect’ enables transfers between DBMS software and a database. The database in return connects the DBMS. An example of this is query transfer from a system to a database. The database in return transfers the answer of the query.

SQL stands for Structured Query Language. It is a language used for schema creation and schema modifications, data-access control, creating an SQL client and creating an SQL server for a database. It is a language for managing relational databases, and viewing, querying and changing (update, insert, append or delete) databases.

Database Connection refers a function DB_connect open () which an application calls to connect to enable the access to the DBMS. The application calls the function DB_connect close () to disable the access.

Database Connectivity (DBC) refers to a standard application programming interface (API), which provides connectivity for accessing the DBMSs. A DBC design is independent of the DB system and OS used. An application written using a DBC can therefore perform operations or actions at both the client and the DB server end. Little changes in code suffice for accessing the data. Two examples of DBCs are Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC).

Database Connectivity Driver refers to a translation layer which resides between an application using the application and the DBMS. The application uses DBC functions through a DBC driver manager with which it is linked. A DBC driver manager manages the drivers associated with the DBMSs. The DBC driver sends the queries to a DBMS. Drivers exist for many data sources and all major DBMSs.

DB2 is IBM RDBMS. DB2 has many features. For example, triggers, stored procedures

and dynamic bitmapped indexing for number of application types, such as traditional host-based applications, client/server-based applications and business intelligence applications.

Data Warehouse refers to sharable data, data stores and databases in an enterprise. It consists of integrated, subject oriented (such as finance, human resources and business) and non-volatile data stores, which update regularly.

Data Mart is a subset of data warehouse. Data mart corresponds to specific business entity on a single subject (or functional area), such as sales or finance data mart is also known as High Performance Query Structures (HPQS).

Process means a composition of group of structured activities, tasks or services that lead to a particular goal. For example, purchase process for airline tickets. A *process* specifies activities with relevance rules based on data in the process.

Process Matrix refers to a multi-element entity, each element of which relates a set of data or inputs to an activity (or subset of activities).

Business Process is an activity, series of activities or a collection of inter-related structured activities, tasks or processes. A business process serves a particular goal, specific result, service or product. The business process is a representation, process matrix or flowchart of a sequence of activities with interleaving decision points.

Business Intelligence is a process which enables a business service to extract new facts and knowledge that enable intelligent decisions. The new facts and knowledge follow from the previous results of business-data processing, aggregation and analysis.

Batch Processing is processing of transactions in batches with no interactions. When one set of transactions finish, the results are stored and the next batch starts processing. Credit card transactions is a good example of the same. The results aggregate at the end of the month for all usages of the card. Batch processing involves the collection of inputs for a specified period and then running them in a scheduled manner.

Batch Transaction Processing refers to the execution of a series of transactions without user interactions. Transaction jobs are set up so they can be run to completion. Scripts, command-line arguments, control files or job-control language *predefine* the input parameters for the transactions.

Streaming Transaction Processing refers to processing for log streams, event streams, twitter streams and queries. The processing of streaming data needs a specialized software framework. Storm from Twitter, S4 from Yahoo, SPARK streaming, HStreaming and Flume are examples of frameworks for real-time streaming computations.

In-memory means operations using CPU memory, such as RAM or caches. Data in-

memory is from a disk or external data source. The operations are fast on in-memory accesses of data, table or data sets, columns or rows compared to disk-accesses.

Interactive Transaction Processing means processing the transactions which involve continual exchange of information between the computer and user; for example, user interactions during e-shopping or e-banking. The processing here is just the opposite of batch processing. Decision on historical data is fast. Interactive query processing has low latency. Low latencies are obtained by the various approaches: massively parallel processing (MPP), in-memory databases and columnar databases.

Real-Time Processing refers to processing for obtaining results for making decisions in real time, processing as and when the data acquires or generates in live data (streaming) with low latency.

Real-Time Transaction Processing means that transactions process at the same time as the data arrives from the data sources. An example of such processing is transaction processing at an ATM machine.

Extract, Transform and Load (ETL) refers to the process, which enables data retrieval, integration, transformation and storage (load). *Extract* means obtaining data from homogeneous or heterogeneous data sources. *Transform* means transforming or optimizing data for the application, and storing the data in an appropriate structure or format. *Load* means the structured data is loaded in the final target database, i.e. data store or data warehouse.

Machine is a computing node or platform for processing, computing and storing. Here, sets of data, programs, applications, DBs or DBMSs reside. When other remote machines access the resources from the machine, it is identified by a name within a network.

Server is a processing, computing and storing node. A server generates responses, sends replies and messages, and renders the data sought. *Server* refers to sets of data, programs, applications, data-stores, DBs or DBMSs which the clients access.

Service means a mechanism which enables the provisioning of access to one or more capabilities. An interface provides the access capabilities. The access to a capability is consistent with various constraints and policies. A *service description* specifies these constraints and policies. Examples of services are web service, cloud service and BigQuery service.

Service-Oriented Architecture (SOA) is a software architecture model which consists of services, messages, operations and processes. SOA components distribute over a network or the Internet in a high-level business entity. New business applications and an application-integration architecture can be developed using an SOA in an enterprise.

Descriptive Analytics refers to deriving additional value from visualizations and reports.

Predictive Analytics refers to advanced analytics which enables extraction of new facts and knowledge to predict or forecast.

Prescriptive Analytics refers to derivation of additional value and undertaking better decisions for new option(s); for example, maximizing profit.

Cognitive Analytics refer to analysis of sentiments, emotions, gestures, facial expressions, and actions similar to ones the humans do. The analytics follow the process of learning, understanding and representing. [Cognitive in English means relating to the process of learning, understanding and representing knowledge. (Collins Dictionary)]

This chapter introduces the readers to the concepts of Big Data, scaling-up and scaling-out of data processing and scalability for storage and analytics. It introduces the concepts of data processing architecture, data sources, data quality and the new technological developments in data management for analysis. These are supported by examples and cases on Big Data analytics. This chapter aims to build a foundation before the in-depth study of Big Data and analytics tools facilitated by the subsequent chapters of the book.

Section 1.2 introduces Big Data and its characteristics, types and classification methods. Section 1.3 describes scalability, scaling up, scaling out of processing and analytics, massively parallel processors, and cloud, grid and distributed computing. Section 1.4 introduces data architecture design and data management. Section 1.5 describes data sources, data quality, data pre-processing and export of data stores to the cloud. Section 1.6 describes traditional systems, such as SQL, Relational Database Management System (RDBMS), enterprise servers and data warehouse for data storage and analysis, as well as the approaches for Big Data storage, processing and analytics. Section 1.7 describes Big Data analytics case studies and applications.

1.2 BIG DATA

Following subsections describe the definitions of data, web data, Big Data, Big Data characteristics, types, classifications and handling techniques:

LO 1.1

Conceptual understanding of data, web data, Big Data, characteristics, types, classifications and handling techniques

Definitions of Data

Data has several definitions. Usages can be singular or plural.

“Data is information, usually in the form of facts or statistics that one can analyze or use for further calculations.” [Collins English Dictionary] “Data is information that can be stored and used by a computer program.”. [Computing] “Data is information presented in numbers, letters, or other form”. [Electrical Engineering, Circuits, Computing and Control] “Data is information from series of observations, measurements or facts”.

[Science] “Data is information from series of behavioural observations, measurements or facts”. [Social Sciences]

Definition of Web Data

Web is large scale integration and presence of data on web servers. Web is a part of the Internet that stores web data in the form of documents and other web resources. URLs enable the access to web data resources.

Web data is the data present on web servers (or enterprise servers) in the form of text, images, videos, audios and multimedia files for web users. A user (client software) interacts with this data. A client can access (pull) data of responses from a server. The data can also publish (push) or post (after registering subscription) from a server. Internet applications including web sites, web services, web portals, online business applications, emails, chats, tweets and social networks provide and consume the web data.

Some examples of web data are Wikipedia, Google Maps, McGraw-Hill Connect, Oxford Bookstore and YouTube.

1. Wikipedia is a web-based, free-content encyclopaedia project supported by the Wikimedia Foundation.
2. Google Maps is a provider of real-time navigation, traffic, public transport and nearby places by Google Inc.
3. McGraw-Hill Connect is a targeted digital teaching and learning environment that saves students' and instructors' time by improving student performance for a variety of critical outcomes.
4. Oxford Bookstore is an online book store where people can find any book that they wish to buy from millions of titles. They can order their books online at www.oxfordbookstore.com
5. YouTube allows billions of people to discover, watch and share originally-created videos by Google Inc.

1.2.1 Classification of Data—Structured, Semi-structured and Unstructured

Data can be classified as structured, semi-structured, multi-structured and unstructured.

Structured data conform and associate with data schemas and data models. Structured data are found in tables (rows and columns). Nearly 15–20% data are in structured or semi-structured form. Unstructured data do not conform and associate with any

data models.

Applications produce continuously increasing volumes of both *unstructured* and *structured* data. Data sources generate data in three forms, viz. structured, semi-structured and unstructured. (Refer online contents associated with the Practice Exercise 1.1 for four forms, viz. structured, semi-structured, multi-structured and unstructured sources.)

Using Structured Data

Structured data enables the following:

- *data insert, delete, update and append*
- *Indexing* to enable faster data retrieval
- *Scalability* which enables increasing or decreasing capacities and data processing operations such as, storing, processing and analytics
- *Transactions processing* which follows ACID rules (Atomicity, Consistency, Isolation and Durability)
- *encryption and decryption* for data security.

Using Semi-Structured Data

Examples of semi-structured data are XML and JSON documents. Semi-structured data contain tags or other markers, which separate semantic elements and enforce hierarchies of records and fields within the data. Semi-structured form of data does not conform and associate with formal data model structures. Data do not associate data models, such as the relational database and table models.

Using Multi-Structured Data

Multi-structured data refers to data consisting of multiple formats of data, viz. structured, semi-structured and/or unstructured data. Multi-structured data sets can have many formats. They are found in non-transactional systems. For example, streaming data on customer interactions, data of multiple sensors, data at web or enterprise server or the data-warehouse data in multiple formats.

Large-scale interconnected systems are thus required to aggregate the data and use the widely distributed resources efficiently.

Multi- or semi-structured data has some semantic meanings and data is in both structured and unstructured formats. But as structured data, semi-structured data nowadays represent a few parts of data (5-10%). Semi-structured data type has a greater presence compared to structured data.

Following is an example of multi-structured data.

EXAMPLE 1.1

Give examples of multi-structured data.

SOLUTION

Structured component of data: Each chess moves is recorded in a table in each match that players refer in future. The records consist of serial numbers (row numbers, which mean move numbers) in the first column and the moves of White and Black in two subsequent vertical columns. Volume of data, i.e. data used for analyzing erroneous or best moves in the matches, keeps growing with more and more tables, and may eventually become ‘voluminous data’.

Unstructured component of data: Social media generates data after each international match. The media publishes the analysis of classical matches played between Grand Masters. The data for analyzing chess moves of these matches are thus in a variety of formats.

Multi-structured data: The voluminous data of these matches can be in a structured format (i.e. tables) as well as in unstructured formats (i.e. text documents, news columns, blogs, Facebook etc.). Tools of multi-structured data analytics assist the players in designing better strategies for winning chess championships.

Using Unstructured Data

Unstructured data does not possess data features such as a table or a database. Unstructured data are found in file types such as .TXT, .CSV. Data may be as key-value pairs, such as hash key-value pairs. Data may have internal structures, such as in emails. The data do not reveal relationships, hierarchy relationships or object-oriented features, such as extendibility. The relationships, schema and features need to be separately established. Growth in data today can be characterised as mostly unstructured data. Following are some examples of unstructured data.

EXAMPLE 1.2

Give examples of unstructured data.

SOLUTION

Examples of unstructured data are:

- **Mobile data:** Text messages, chat messages, tweets, blogs and comments
- **Website content data:** YouTube videos, browsing data, e-payments, web store

- data, user-generated maps
- Social media data: For exchanging data in various forms
- Texts and documents
- Personal documents and e-mails
- Text internal to an organization: Text within documents, logs, survey results
- Satellite images, atmospheric data, surveillance, traffic videos, images from Instagram, Flickr (upload, access, organize, edit and share photos from any device from anywhere in the world).

1.2.2 Big Data Definitions

Big Data is high-volume, high-velocity and/or high-variety information asset that requires new forms of processing for enhanced *decision making, insight discovery and process optimization* (Gartner¹ 2012). Other definitions can be found in existing literature.

Industry analyst Doug Laney described the ‘3Vs’, i.e. volume, variety and/or velocity as the key “data management challenges” for enterprises. Analytics also describe the ‘4Vs’, i.e. volume, velocity, variety and veracity. A number of other definitions are available for Big Data, some of which are given below.

“A collection of data sets so large or complex that traditional data processing applications are inadequate.” – Wikipedia

“Data of a very large size, typically to the extent that its manipulation and management present significant logistical challenges.” [Oxford English Dictionary (traditional database of authoritative definitions)]

“Big Data refers to data sets whose size is beyond the ability of typical database software tool to capture, store, manage and analyze.” [The McKinsey Global Institute, 2011]

1.2.3 Big Data Characteristics

Characteristics of Big Data, called 3Vs (and 4Vs also used) are:

Volume The phrase ‘Big Data’ contains the term *big*, which is related to size of the data and hence the characteristic. Size defines the amount or quantity of data, which is generated from an application(s). The size determines the processing considerations needed for handling that data.

Velocity The term velocity refers to the speed of generation of data. Velocity is a

Meanings and various definitions of the word 'Big Data'

measure of how fast the data generates and processes. To meet the demands and the challenges of processing Big Data, the velocity of generation of data plays a crucial role.

Variety Big Data comprises of a variety of data. Data is generated from multiple sources in a system. This introduces variety in data and therefore introduces ‘complexity’. Data consists of various forms and formats. The variety is due to the availability of a large number of heterogeneous platforms in the industry. This means that the type to which Big Data belongs to is also an important characteristic that needs to be known for proper processing of data. This characteristic helps in effective use of data according to their formats, thus maintaining the importance of Big Data.

Veracity is also considered an important characteristic to take into account the quality of data captured, which can vary greatly, affecting its accurate analysis.

The 4Vs (i.e. volume, velocity, variety and veracity) data need tools for mining, discovering patterns, business intelligence, artificial intelligence (AI), machine learning (ML), text analytics, descriptive and predictive analytics, and the data visualization tools.

1.2.4 Big Data Types

A task team on Big Data classified the types of Big Data (June 2013)². Another team from IBM developed a different classification for Big Data types.³

Following are the suggested types:

1. *Social networks and web data*, such as Facebook, Twitter, e-mails, blogs and YouTube.
2. *Transactions data and Business Processes (BPs) data*, such as credit card transactions, flight bookings, etc. and public agencies data such as medical records, insurance business data etc.
3. *Customer master data*, such as data for facial recognition and for the name, date of birth, marriage anniversary, gender, location and income category,
4. *Machine-generated data*, such as machine-to-machine or Internet of Things data, and the data from sensors, trackers, web logs and computer systems log. Computer generated data is also considered as machine generated data from data store. Usage of programs for processing of data using data repositories, such as database or file, generates data and also machine generated data.
5. *Human-generated data* such as biometrics data, human-machine interaction data, e-mail records with a mail server and MySQL database of student grades. Humans also records their experiences in ways such as writing these in notebooks or

diaries, taking photographs or audio and video clips. Human-sourced information is now almost entirely digitized and stored everywhere from personal computers to social networks. Such data are loosely structured and often ungoverned.

The following examples illustrate machine-generated data.

EXAMPLE 1.3

Give three examples of the machine-generated data.

SOLUTION

Examples of machine-generated data are:

1. Data from computer systems: Logs, web logs, security/surveillance systems, videos/images etc.
2. Data from fixed sensors: Home automation, weather sensors, pollution sensors, traffic sensors etc.
3. Mobile sensors (tracking) and location data.

Section 1.7 describes Big Data Analytics use cases, case studies and applications in detail. The following example illustrates the usages of Big Data generated from multiple types of data sources for optimizing the services offered, products, schedules and predictive tasks.

EXAMPLE 1.4

Think of a manufacturing and retail marketing company, such as LEGO toys.

How does such a toy company optimize the services offered, products and schedules, devise ways and use Big Data processing and storing for predictions using analytics?

SOLUTION

Assume that a retail and marketing company of toys uses several Big Data sources, such as (i) *machine-generated data* from sensors (RFID readers) at the toy packaging, (ii) *transactions data* of the sales stored as web data for automated reordering by the retail stores and (iii) tweets, Facebook posts, e-mails, messages, and web data for messages and reports.

The company uses Big Data for understanding the toys and themes in present days that are popularly demanded by children, predicting the future types and demands. The company using such predictive analytics, optimizes the product mix

and manufacturing processes of toys. The company optimizes the services to retailers by maintaining toy supply schedules. The company sends messages to retailers and children using social media on the arrival of new and popular toys.

The following example illustrates the Big Data features of 3Vs and their applications.

EXAMPLE 1.5

Give an example of features of 3Vs in Big Data and application.

SOLUTION

Consider satellite images of the Earth's atmosphere and its regions. The *Volume* of data from the satellites is large. A number of Indian satellites, such as KALPANA, INSAT-1A and INSAT-3D generate this data. Foreign satellites also generate voluminous data continuously. Satellites record the images of full disk and sectors, such as east and west Asia sectors and regions.

Velocity is also large. A number of satellites collect this data round the clock. Big Data analytics helps in drawing of maps of wind velocities, temperatures and other whether parameters.

Variety of images can be in visible range, such as IR-1 (infrared range -1), IR-2(infrared range -2), shortwave infrared (SWIR), MIR (medium range IR) and colour composite.

Data *Veracity*, uncertain or imprecise data, is as important as Volume, Velocity and Variety. Uncertainty arises due to poor resolutions used for recording or noise in images due to signal impairments.

Data processing needs increased speed of computations due to higher volumes. Need of data management, storage and increased analytics requires new innovative non-traditional methods.

Big Data of satellites helps in predicting weather, and mapping of different crops and from that estimating the expected crop yield.

The following examples explain the uses of Big Data generated from multiple types of data sources.

EXAMPLE 1.6

How are Big Data used in the following companies and services using analytics?

- (i) Chocolate Marketing Company with large number of installed Automatic Chocolate Vending Machines (ACVMs)
- (ii) Automotive Components and Predictive Automotive Maintenance Services (ACPAMS) rendering customer services for maintenance and servicing of (Internet) connected cars and its components
- (iii) Weather data Recording, Monitoring and Prediction (WRMP) Organization.

SOLUTION

- (i) Assume ACVM company. Each ACVM sells five flavours (FL1, FL2, FL3, FL4 and FL5) KitKat, Milk, Fruit and Nuts, Nougat and Oreo. The company uses Big Data types as: *Machine-generated data* on the sale of chocolates, reports of unfilled or filled machine *transaction data*. *Human-generated data* of buyer-machine interactions at the ACVMs. *Social networks and web data* on feedback and personalized messages based on interactions and human-generated data on facial recognition of the buyers. The company uses Big Data for efficient and optimum planning of fill service for chocolates, sentiment analysis of buyers for specific flavours, ACVMs location and periods of higher-sales analysis, assessing needs of predictive maintenances of machines, additions and relocations of machines, and predictions, strategies and planning for festival sales.
- (ii) ACPAMS uses Big Data types as: *machine-generated data* from sensors at automotive components, such as brakes, steering and engine from each car; *transactions data* stored at the service website; social networks and web data in the form of messages, feedback and reports from customers. The service provides messages for scheduled and predictive maintenances. The service generates reports on *social networks and updates the web data* for the manufacturing plant. The service generates reports about components qualities and needed areas for improvement in products of the company.
- (iii) WRMP Organization uses Big Data types as: machine-generated data from sensors at weather stations and satellites, social networks and web data and the reports and alerts issued by many centers around the world. The organization stores and processes the weather records generated by its stations, social networks and web data collected from other centers. The organization issues maps and weather warnings, predicts weather, rainfall in various regions, expected dates of arrival of monsoon in different regions, issues forecasts on social networks and web pages, generates social network and web data for areal maps of cloud and wind.

1.2.5 Big Data Classification

Big Data can be classified on the basis of its characteristics that are used for designing data architecture for processing and analytics. Table 1.1 gives various classification methods for data and Big Data.

Table 1.1 Various classification methods for data and Big Data

Basis of Classification	Examples
Data sources (traditional)	Data storage such as records, RDBMs, distributed databases, row-oriented In-memory data tables, column-oriented In-memory data tables, data warehouse, server, machine-generated data, human-sourced data, Business Process (BP) data, Business Intelligence (BI) data
Data formats (traditional)	Structured and semi-structured
Big Data sources	Data storage, distributed file system, Operational Data Store (ODS), data marts, data warehouse, NoSQL database (MongoDB, Cassandra), sensors data, audit trail of financial transactions, external data such as web, social media, weather data, health records
Big Data formats	Unstructured, semi-structured and multi-structured data
Data Stores structure	Web, enterprise or cloud servers, data warehouse, row-oriented data for OLTP, column-oriented for OLAP, records, graph database, hashed entries for key/value pairs
Processing data rates	Batch, near-time, real-time, streaming
Processing Big Data rates	High volume, velocity, variety and veracity, batch, near real-time and streaming data processing,
Analysis types	Batch, scheduled, near real-time datasets analytics
Big Data processing methods	Batch processing (for example, using MapReduce, Hive or Pig), real-time processing (for example, using SparkStreaming, SparkSQL, Apache Drill)
Data analysis methods	Statistical analysis, predictive analysis, regression analysis, Mahout, machine learning algorithms, clustering algorithms, classifiers, text analysis, social network analysis, location-based analysis, diagnostic analysis, cognitive analysis
	Human, business process, knowledge discovery, enterprise applications, Data

1.2.6 Big Data Handling Techniques

Following are the techniques deployed for Big Data storage, applications, data management and mining and analytics:

- Huge data volumes storage, data distribution, high-speed networks and high-performance computing
- Applications scheduling using open source, reliable, scalable, distributed file system, distributed database, parallel and distributed computing systems, such as Hadoop (Chapter 2) or Spark (Chapters 5–10)
- Open source tools which are scalable, elastic and provide virtualized environment, clusters of data nodes, task and thread management
- Data management using NoSQL, document database, column-oriented database, graph database and other form of databases used as per needs of the applications and in-memory data management using columnar or Parquet formats during program execution
- Data mining and analytics, data retrieval, data reporting, data visualization and machine-learning Big Data tools.

Self-Assessment Exercise linked to LO 1.1

1. How do you define data, web data and Big Data?
2. How do you classify data as structured, semi-structured, multi-structured and unstructured?
3. Give data example of student records at a University and explain structured data, hierarchical relationships between them.
4. Recall three examples in Example 1.6. How would you classify data which you shall be using for analytics in these examples?
5. Consider the usage examples of Big Data for a car company. Assume that company manufactures five models of cars, and each model is available in five colours and five shades. The company collects inputs from customers and sales centres, and inputs of component malfunctions from service centres for different models. The company also uses social media inputs. Explain 3Vs characteristics in this company's data.

1.3 | SCALABILITY AND PARALLEL PROCESSING

LO 1.2

Big Data needs processing of large data volume, and therefore needs intensive computations. Processing complex applications with large datasets (terabyte to petabyte datasets) need hundreds of computing nodes. Processing of this much distributed data within a short time and at minimum cost is problematic.

Scalability, scaling up, scaling out in distributed computing, Massively Parallel Processing (MPP), cloud, grid, volunteering computing systems

Convergence of Data Environments and Analytics

Big Data can co-exist with traditional data store. Traditional data stores use RDBMS tables or data warehouse. Big Data processing and analytics requires scaling up and scaling out, both vertical and horizontal computing resources. Computing and storage systems when run in parallel, enable scaling out and increase system capacity.

Scalability enables increase or decrease in the capacity of data storage, processing and analytics. *Scalability* is the capability of a system to handle the workload as per the magnitude of the work. System capability needs increment with the increased workloads. When the workload and complexity exceed the system capacity, scale it up and scale it out.

The following subsection describes the concept of analytics scalability.

1.3.1 Analytics Scalability to Big Data

Vertical scalability means scaling up the given system's resources and increasing the system's analytics, reporting and visualization capabilities. This is an additional way to solve problems of greater complexities. *Scaling up* means designing the algorithm according to the architecture that uses resources efficiently. For example, x terabyte of data take time t for processing, code size with increasing complexity increase by factor n , then scaling up means that processing takes equal, less or much less than $(n \times t)$.

Horizontal scalability means increasing the number of systems working in coherence and scaling out the workload. Processing different datasets of a large dataset deploys horizontal scalability. *Scaling out* means using more resources and distributing the processing and storage tasks in parallel. If r resources in a system process x terabyte of data in time t , then the $(p \times x)$ terabytes process on p parallel distributed nodes such that the time taken up remains t or is slightly more than t (due to the additional time required for Inter Processing nodes Communication (IPC)).

The easiest way to scale up and scale out execution of analytics software is to implement it on a bigger machine with more CPUs for greater volume, velocity, variety and complexity of data. The software will definitely perform better on a bigger machine. However, buying faster CPUs, bigger and faster RAM modules and hard disks, faster and bigger motherboards will be expensive compared to the extra performance achieved by efficient design of algorithms. Also, if more CPUs add in a computer, but the software does not exploit the advantage of them, then that will not get any increased performance out of the additional CPUs.

Alternative ways for scaling up and out processing of analytics software and Big Data analytics deploy the Massively Parallel Processing Platforms (MPPs), cloud, grid, clusters, and distributed computing software.

The following subsections describe computing methods for high availability and scalable computations and analysis.

1.3.2 Massively Parallel Processing Platforms

Scaling uses parallel processing systems. Many programs are so large and/or complex that it is impractical or impossible to execute them on a single computer system, especially in limited computer memory. Here, it is required to enhance (scale) up the computer system or use massive parallel processing (MPPs) platforms. Parallelization of tasks can be done at several levels: (i) distributing separate tasks onto separate threads on the same CPU, (ii) distributing separate tasks onto separate CPUs on the same computer and (iii) distributing separate tasks onto separate computers.

A solution for Big data processing is to perform parallel and distributed computing in a cloud-computing environment.

When making software, draw the advantage of multiple computers (or even multiple CPUs within the same computer) and software which need to be able to parallelize tasks. Multiple compute resources are used in parallel processing systems. The computational problem is broken into discrete pieces of sub-tasks that can be processed simultaneously. The system executes multiple program instructions or sub-tasks at any moment in time. Total time taken will be much less than with a single compute resource.

1.3.2.1 Distributed Computing Model

A distributed computing model uses cloud, grid or clusters, which process and analyze big and large datasets on distributed computing nodes connected by high-speed networks. Table 1.2 gives the requirements of processing and analyzing big, large and small to medium datasets on distributed computing nodes. Big Data processing uses a parallel, scalable and no-sharing program model, such as MapReduce, for computations on it. (Chapter 2)

Table 1.2 Distributed computing paradigms

Distributed computing on multiple processing nodes/clusters	Big Data > 10 M	Large datasets below 10 M	Small to medium datasets up to 1 M
Distributed computing	Yes	Yes	No
Parallel computing	Yes	Yes	No
Scalable computing	Yes	Yes	No
Shared nothing (No in-between data sharing and inter-processor communication)	Yes	Limited sharing	No
Shared in-between between the distributed nodes/clusters	No	Limited sharing	Yes

1.3.3 Cloud Computing

Wikipedia defines cloud computing as, “Cloud computing is a type of Internet-based computing that provides shared processing resources and data to the computers and other devices on demand.”

One of the best approach for data processing is to perform parallel and distributed computing in a cloud-computing environment. Cloud usages circumvent the single point failure due to failing of one node. Cloud design performs as a whole. Its multiple nodes perform automatically and interchangeably. It offers high data security compared to other distributed technologies.

Cloud resources can be Amazon Web Service (AWS) Elastic Compute Cloud (EC2), Microsoft Azure or Apache CloudStack. Amazon Simple Storage Service (S3) provides simple web services interface to store and retrieve any amount of data, at any time, from anywhere on the web. [Amazon EC2 name possibly drives from the feature that EC2 has a simple web service interface, which provides and configures the storage and computing capacity with minimal friction].

Cloud computing features are: (i) on-demand service (ii) resource pooling, (iii) scalability, (iv) accountability, and (v) broad network access. Cloud services can be accessed from anywhere and at any time through the Internet. A local private cloud can also be set up on a local cluster of computers.

Cloud computing allows availability of computer infrastructure and services “on-demand” basis. The computing infrastructure includes data storage device, development platform, database, computing power or software applications.

Cloud services can be classified into three fundamental types:

1. Infrastructure as a Service (IaaS): Providing access to resources, such as hard disks, network connections, databases storage, data center and virtual server spaces is Infrastructure as a Service (IaaS). Some examples are Tata Communications, Amazon data centers and virtual servers. Apache CloudStack is an open source software for deploying and managing a large network of virtual machines, and offers public cloud services which provide highly scalable Infrastructure as a Service (IaaS).
2. Platform as a Service (PaaS): It implies providing the runtime environment to allow developers to build applications and services, which means cloud Platform as a Service. Software at the clouds support and manage the services, storage, networking, deploying, testing, collaborating, hosting and maintaining applications. Examples are Hadoop Cloud Service (IBM BigInsight, Microsoft Azure HD Insights, Oracle Big Data Cloud Services).
3. Software as a Service (SaaS): Providing software applications as a service to end-users is known as Software as a Service. Software applications are hosted by a service provider and made available to customers over the Internet. Some examples are SQL GoogleSQL, IBM BigSQL, HPE Vertica, Microsoft Polybase and Oracle Big Data SQL.

Cloud services offer IaaS, SaaS and PaaS as service models for processing and analyzing the large datasets on computing nodes.

1.3.4 Grid and Cluster Computing

Grid Computing

Grid Computing refers to distributed computing, in which a group of computers from several locations are connected with each other to achieve a common task. The computer resources are heterogeneously and geographically disperse. A group of computers that might spread over remotely comprise a grid. A grid is used for a variety of purposes. A single grid of course, dedicates at an instance to a particular application only. Grid computing provides large-scale resource sharing which is flexible, coordinated and secure among its users. The users consist of individuals, organizations and resources.

Grid computing suits data-intensive storage better than storage of small objects of few millions of bytes. To achieve the maximum benefit from data grids, they should be used for a large amount of data which can distribute over grid nodes. Besides data grid, the other variation of grid, i.e., computational grid focuses on computationally intensive operations.

Features of Grid Computing Grid computing, similar to cloud computing, is scalable. Cloud computing depends on sharing of resources (for example, networks, servers, storage, applications and services) to attain coordination and coherence among resources similar to grid computing. Similarly, grid also forms a distributed network for resource integration.

Drawbacks of Grid Computing Grid computing is the single point, which leads to failure in case of underperformance or failure of any of the participating nodes. A system's storage capacity varies with the number of users, instances and the amount of data transferred at a given time. Sharing resources among a large number of users helps in reducing infrastructure costs and raising load capacities.

Cluster Computing

A cluster is a group of computers connected by a network. The group works together to accomplish the same task. Clusters are used mainly for load balancing. They shift processes between nodes to keep an even load on the group of connected computers. Hadoop architecture uses the similar methods (Chapter 2).

Table 1.3 gives a comparison of grid computing with the distributed and cluster computing.

Table 1.3 Grid computing and related paradigms

Distributed computing	Cluster computing	Grid computing
<ul style="list-style-type: none">• Loosely coupled• Heterogeneous• Single administration	<ul style="list-style-type: none">• Tightly coupled• Homogeneous• Cooperative working	<ul style="list-style-type: none">• Large scale• Cross organizational• Geographical distribution• Distributed management

1.3.5 Volunteer Computing

Volunteers provide computing resources to projects of importance that use resources to do distributed computing and/or storage. **Volunteer computing** is a distributed computing paradigm which uses computing resources of the volunteers. Volunteers are organizations or members who own personal computers. **Projects** examples are science-related projects executed by universities or academia in general.

Some issues with volunteer computing systems are:

1. Volunteered computers heterogeneity
2. Drop outs from the network over time
3. Their sporadic availability

4. Incorrect results at volunteers are unaccountable as they are essentially from anonymous volunteers.

Self-Assessment Exercise linked to LO 1.2

1. Define analytics scalability, horizontal scalability and vertical scalability.
2. How does platform differ from software? When will a program use SaaS and when PaaS?
3. List the features of grid computing. How does it differ from cluster and cloud computing?
4. Why do we use distributed computing for analytics of large datasets?

1.4 DESIGNING DATA ARCHITECTURE

The following subsections describe how to design Big Data architecture layers and how to manage data for analytics.

LO 1.3

Design of data architecture layers and their functions, and data management functions for the analytics

1.4.1 Data Architecture Design

Techopedia defines *Big Data architecture* as follows: “Big Data architecture is the logical and/or physical layout/structure of how Big Data will be stored, accessed and managed within a Big Data or IT environment. Architecture logically defines how Big Data solution will work, the core components (hardware, database, software, storage) used, flow of information, security and more.”

Characteristics of Big Data make designing Big Data architecture a complex process. Further, faster additions of new technological innovations increase the complexity in design. The requirements for offering competing products at lower costs in the market make the designing task more challenging for a Big Data architect.

Data analytics need the number of sequential steps. Big Data architecture design task simplifies when using the logical layers approach. Figure 1.2 shows the logical layers and the functions which are considered in Big Data architecture.

Five vertically aligned textboxes on the left of Figure 1.2 show the layers. Horizontal textboxes show the functions in each layer.

Data processing architecture consists of five layers: (i) identification of data sources, (ii) acquisition, ingestion, extraction, pre-processing, transformation of data, (iii) data

storage at files, servers, cluster or cloud, (iv) data-processing, and (v) data consumption in the number of programs and tools.

Layer 5 Data consumption	Export of datasets to cloud, web etc.	Datasets usages: BPs, Bls, knowledge discovery	Analytics (real-time, near real-time, scheduled batches), reporting, visualization
Layer 4 Data processing	Processing technology: MapReduce, Hive, Pig, Spark	Processing in real-time, scheduled batches or hybrid	Synchronous or asynchronous processing
Layer 3 Data storage	Considerations of types (historical or incremental), formats, compression, frequency of incoming data, patterns of querying and data consumption	Hadoop distributed file system (scaling, self-managing and self-healing), Spark, Mesos or S3	NoSQL data stores – Hbase, MongoDB, Cassandra, Graph database
Layer 2 Data ingestion and acquisition	Ingestion using Extract Load and Transform (ELT)	Data semantics (such as replace, append, aggregate, compact, fuse)	Pre-processing (validation, transformation or transcoding) requirement
Layer 1 Identification of internal and external sources of data	Sources for ingestion of data	Push or pull of data from the sources for ingestion	Ingestion of data from sources in batches or real time
		Data types for database, files, web or service	Data formats: structured, semi- or unstructured for ingestion

Figure 1.2 Design of logical layers in a data processing architecture, and functions in the layers

Data consumed for applications, such as business intelligence, data mining, discovering patterns/clusters, artificial intelligence (AI), machine learning (ML), text analytics, descriptive and predictive analytics, and data visualization.

Data ingestion, pre-processing, storage and analytics require special tools and technologies.

Logical layer 1 (L1) is for identifying data sources, which are external, internal or both. The layer 2 (L2) is for data-ingestion.

Data ingestion means a process of absorbing information, just like the process of absorbing nutrients and medications into the body by eating or drinking them (Cambridge English Dictionary). Ingestion is the process of obtaining and importing data for immediate use or transfer. Ingestion may be in batches or in real time using pre-processing or semantics.

The L3 layer is for storage of data from the L2 layer. The L4 is for data processing using software, such as MapReduce, Hive, Pig or Spark. The top layer L5 is for data consumption. Data is used in analytics, visualizations, reporting, export to cloud or web servers.

L1 considers the following aspects in a design:

- Amount of data needed at ingestion layer 2 (L2)
- Push from L1 or pull by L2 as per the mechanism for the usages
- Source data-types: Database, files, web or service
- Source formats, i.e., semi-structured, unstructured or structured.

L2 considers the following aspects:

- Ingestion and ETL processes either in real time, which means store and use the data as generated, or in batches. Batch processing is using discrete datasets at scheduled or periodic intervals of time.

L3 considers the followings aspects:

- Data storage type (historical or incremental), format, compression, incoming data frequency, querying patterns and consumption requirements for L4 or L5
- Data storage using Hadoop distributed file system or NoSQL data stores—HBase, Cassandra, MongoDB.

L4 considers the followings aspects:

- Data processing software such as MapReduce, Hive, Pig, Spark, Spark Mahout, Spark Streaming
- Processing in scheduled batches or real time or hybrid
- Processing as per synchronous or asynchronous processing requirements at L5.

L5 considers the consumption of data for the following:

- Data integration
- Datasets usages for reporting and visualization
- Analytics (real time, near real time, scheduled batches), BPs, BIs, knowledge discovery
- Export of datasets to cloud, web or other systems

1.4.2 Managing Data for Analysis

Data managing means enabling, controlling, protecting, delivering and enhancing the value of data and information asset. Reports, analysis and visualizations need well-defined data. Data management also enables data usage in applications. The process for managing needs to be well defined for fulfilling requirements of the applications.

Data management functions include:

1. Data assets creation, maintenance and protection
2. Data governance, which includes establishing the processes for ensuring the availability, usability, integrity, security and high-quality of data. The processes enable trustworthy data availability for analytics, followed by the decision making at the enterprise.
3. Data architecture creation, modelling and analysis
4. Database maintenance, administration and management system. For example, RDBMS (relational database management system), NoSQL
5. Managing data security, data access control, deletion, privacy and security
6. Managing the data quality
7. Data collection using the ETL process
8. Managing documents, records and contents
9. Creation of reference and master data, and data control and supervision
10. Data and application integration
11. Integrated data management, enterprise-ready data creation, fast access and analysis, automation and simplification of operations on the data,
12. Data warehouse management
13. Maintenance of business intelligence
14. Data mining and analytics algorithms.

Self-Assessment Exercise linked to LO 1.3

1. How are data architecture layers used for analytics?
2. Explain the function of each of the five layers in Big Data architecture design (Figure 1.2).
3. List the functions of the ELT at data ingestion layer and at data storage layer.
4. List the functions in data management.

1.5 DATA SOURCES, QUALITY, PRE-PROCESSING AND STORING

LO 1.4

The following subsections describe data sources, data quality, data pre-processing and data store export to the cloud.

Data sources, data quality, data pre-processing, and data store export to the cloud

1.5.1 Data Sources

Applications, programs and tools use data. Sources can be *external*, such as sensors, trackers, web logs, computer systems logs and feeds. Sources can be machines, which source data from data-creating programs.

Data sources can be structured, semi-structured, multi-structured or unstructured. Data sources can be social media (L1 in Figure 1.2). A source can be *internal*. Sources can be data repositories, such as database, relational database, flat file, spreadsheet, mail server, web server, directory services, even text or files such as comma-separated values (CSV) files. Source may be a data store for applications (L4 in Figure 1.2).

1.5.1.1 Structured Data Sources

Data source for ingestion, storage and processing can be a file, database or streaming data. The source may be on the same computer running a program or a networked computer. Examples of structured data sources are SQL Server, MySQL, Microsoft Access database, Oracle DBMS, IBM DB2, Informix, Amazon SimpleDB or a file-collection directory at a server.

A data source name implies a defined name, which a process uses to identify the source. The name needs to be a meaningful name. For example, a name which identifies the stored data in student grades during processing. The data source name could be *StudentName_Data_Grades*.

A *data dictionary* enables references for accesses to data. The dictionary consists of a set of master lookup tables. The dictionary stores at a central location. The central location enables easier access as well as administration of changes in sources. The name of the dictionary can be *UniversityStudents_DataPlusGrades*. A master-directory server can also be called *NameNode*.

Microsoft applications consider two types of sources for processing: (i) machine sources and (ii) file sources.⁴

Machine data sources and file data sources in Microsoft applications

- (i) Machine sources are present on computing nodes, such as servers. A machine identifies a source by the user-defined name, driver-manager name and source-driver name.

- (ii) File sources are stored files. An application executing the data, first *connects* to a driver manager of the source. A user, client or application *does not register* with the source, but *connects* to the manager when required. The process of connection is simple when using a file data source in case the file contains a connection string that would otherwise have to be built using a call to a connect-function driver.

Oracle applications consider two types of data sources: (i) *database*, which identifies the database information that the software needs to connect to database, and (ii) *logic-machine*, which identifies the machine which runs batches of applications and master business functions.⁵ Source definition

Database data sources and logic-machine data sources in Oracle applications

identifies the machine. The source can be on a network. The definition in that case also includes network information, such as the name of the server, which hosts the machine functions.

The applications consider data sources as the ones where the database tables reside and where the software runs logic objects for an enterprise. Data sources can point to:

1. A database in a specific location or in a data library of OS
2. A specific machine in the enterprise that processes logic
3. A data source master table which stores data source definitions. The table may be at a centralized source (enterprise server) or at server-map for the source.

A database can be in an IBM i data library⁶ [IBM i is a computer operating system in which IBM i considers everything as an object, each possessing persistence. The system IBM i offers Unix-like file directories using an integrated file system.].

Specific database instance or file as data sources in IBM i system

IBM applications consider data sources for applications and tools as one which identifies either (i) a specific database instance or (ii) file on a remote system that stores data.⁶ Data sources can be shared. The access to source is restricted according to the roles assigned to both the source and the application that use it.

EXAMPLE 1.7

- (i) How would you name the data sources of the student grade-sheets?
- (ii) How does an analytics application (Analysis_APP) access student grade-sheet data source, using the Data Dictionary or data-source master-table for the grade-sheets of students?

- (iii) How does the application connect and access the data source of students' grade-sheets?

Assume each student can have a grade-sheet for each of the six semesters in UG Computer Science programme.

SOLUTION

- (i) Assume SemID is distinct key for a semester. StudID is a key assigned to a student, whether in CS or another subject, and whether in UG or PG. A StudID is unique. Data source can be file data source named 'UG_CS_Sem_StudID_Grades' for all UG CS student grades. UG_CS_Sem_StudID_Grades database consists of maximum six grade sheets UG_CS_SemID_StudID_Grades, i.e., one for each semester. Assume that Analysis_APP does not connect or directly links to the data source UG_CS_Sem_StudID_Grades database. Then, the Analysis_APP links to a Data Dictionary or data source master table, which is data repository for the pointers of all six semesters of UG Computer Science program and other subject programs.
- (ii) Assume that Analysis_APP associates to Oracle data-source master-table. The table stores the data-source definitions for all UG and PG, and all subjects and semester grades of the students. The data-source master-table stores the pointers of all semester grades. The table thus points to UG_CS_Sem_StudID_Grades DB for the student identified by StudID.
- (iii) Assume that application deploys Microsoft DB. Then, first Analysis_APP links to a Driver Manager. The Driver Manager then calls the ODBC functions in the Driver Manager. The application identifies the target driver for the UG_CS_Sem_StudID_Grades data source with a connection handle. When the Driver Manager loads the driver, the Driver Manager builds a table of pointers to the functions in that driver. It uses the connection handle passed by the application to find the address of the function in the target driver and calls that function by address.

1.5.1.2 Unstructured Data Sources

Unstructured data sources are distributed over high-speed networks. The data need high velocity processing. Sources are from distributed file systems. The sources are of file types, such as .txt (text file), .csv (comma separated values file). Data may be as key-value pairs, such as hash key-values pairs. Data may have internal structures, such as in e-mail, Facebook pages, twitter messages etc. The data do not model, reveal relationships, hierarchy relationships or object-oriented features, such as extensibility.

1.5.1.3 Data Sources - Sensors, Signals and GPS

The data sources can be sensors, sensor networks, signals from machines, devices, controllers and intelligent edge nodes of different types in the industry M2M communication and the GPS systems.

Sensors are electronic devices that sense the physical environment. Sensors are devices which are used for measuring temperature, pressure, humidity, light intensity, traffic in proximity, acceleration, locations, object(s) proximity, orientations and magnetic intensity, and other physical states and parameters. Sensors play an active role in the automotive industry.

RFIDs and their sensors play an active role in RFID based supply chain management, and tracking parcels, goods and delivery.

Sensors embedded in processors, which include machine-learning instructions, and wireless communication capabilities are innovations. They are sources in IoT applications.

1.5.2 Data Quality

Data quality is high when it represents the real-world construct to which references are taken. High quality means data, which enables all the required operations, analysis, decisions, planning and knowledge discovery correctly. A definition for high quality data, especially for artificial intelligence applications, can be data with five R's as follows: Relevancy, recency, range, robustness and reliability. Relevancy is of utmost importance.

Data quality definitions;
Data relevancy, recency,
range, robustness and
reliability

A uniform definition of data quality is difficult. A reference can be made to a set of values of quantitative or qualitative conditions, which must be specified to say that data quality is high or low.

1.5.2.1 Data Integrity

Data integrity refers to the maintenance of consistency and accuracy in data over its usable life. Software, which store, process, or retrieve the data, should maintain the integrity of data. Data should be incorruptible. For example, in Example 1.7 the grades of students should remain unaffected upon processing.

1.5.2.2 Data Noise, Outliers, Missing and Duplicate Values

Noise One of the factors effecting data quality is noise. Noise in data refers to data giving additional meaningless information besides true (actual/required) information. Noise refers to difference in the value measured from true value due to additional influences. Noisy data means data having large additional information. Result of data

analysis is adversely affected due to noisy data.

Noise is random in character, which means frequency with which it occurs is variable over time. The values show nearly equal positive and negative deviations. A statistical analysis of deviation can select the noise in data and true values can be retrieved.

Outliers A factor which effects quality is an outlier. An *outlier* in data refers to data, which appears to not belong to the dataset. For example, data that is outside an expected range. Actual outliers need to be removed from the dataset, else the result will be effected by a small or large amount. Alternatively, if valid data is identified as outlier, then also the results will be affected. The outliers are a result of human data-entry errors, programming bugs, some transition effect or phase lag in stabilizing the data value to the true value.

Missing Values Another factor effecting data quality is missing values. *Missing value* implies data *not appearing in the data set*.

Duplicate Values Another factor effecting data quality is duplicate values. *Duplicate value* implies the same data appearing two or more times in a dataset.

The following example explains noise, outliers, missing values and duplicate data.

EXAMPLE 1.8

Consider use cases of noise, outliers, missing values and duplicate data. Write the effect on the analysis in each case.

SOLUTION

Following are the examples of machine-generated data.

1. **Noise:** Recall WRMP organization for weather recording. Consider noise in wind velocity and direction readings due to external turbulences. The velocity at certain instances will appear too high and sometimes too low. The directions at certain instances will appear inclined more towards the north and sometimes more towards the south.
2. **Outlier:** Consider an outlier in the students' grade-sheets in one subject out of five in the fourth-semester result of a student. A result in a semester shows 9.0 out of 10 points in place of 3.0 out of 10. Data 9.0 is an outlier. The student semester grade point average (SGPA) will be erroneously declared and the student may even be declared to have failed in that semester.
3. **Missing values:** Consider missing values in the sales figures of chocolates. The values not sent for certain dates from an ACVM. This may be due to the failure

of power supply at the machine or network problems on specific days in a month. The chocolate sales not added for a day can be added in the next day's sales data. The effect over a month on the average sales per day is not significant. However, if the failure occurred on last day of a month, then the analysis will be erroneous.

4. **Duplicate values:** Consider duplicate values in the sales figures of chocolates from an ACVM. This may be due to some problem in the system. The number of duplicates for sales when sent and added, then sales result analysis will get affected. It can even result in false alarms to a service, which maintains the supply chain to the ACVMs.

Assume network problems on certain instances. The ACVM may not get an acknowledgement of the sales figures from the server, leading to sending an incorrect sales record once again. If this happens then sales figures of chocolates get recorded twice at that instance. For example, if the chocolate sales data gets added twice in a specific day's sales data, the calculation of monthly sales data is adversely affected.

1.5.3 Data Pre-processing

Data pre-processing is an important step at the ingestion layer (Figure 1.2). For example, consider grade point data in Example 1.8. The outlier needs to be removed. Pre-processing is a must before data mining and analytics. Pre-processing is also a must before running a Machine Learning (ML) algorithm. Analytics needs prior screening of data quality also. Data when being exported to a cloud service or data store needs pre-processing.

Need of data pre-processing for data store portability and usability in applications and services

Pre-processing needs are:

- (i) Dropping out of range, inconsistent and outlier values
- (ii) Filtering unreliable, irrelevant and redundant information
- (iii) Data cleaning, editing, reduction and/or wrangling
- (iv) Data validation, transformation or transcoding
- (v) ELT processing.

Data Cleaning

Data cleaning refers to the process of removing or correcting incomplete, incorrect, inaccurate or irrelevant parts of the data after detecting them. For example, in Example

1.8 correcting the grade outliers or mistakenly entered values means cleaning and correcting the data.

Needs of data pre-processing using cleaning, enrichment, editing, reduction and wrangling methods

Data Cleaning Tools Data cleaning is done before mining of data. Incomplete or irrelevant data may result into misleading decisions. It is not always possible to create well-structured data. Data can generate in a system in many formats when it is obtained from the web. Data cleaning tools help in refining and structuring data into usable data. Examples of such tools are OpenRefine and DataCleaner.

Data Enrichment

Techopedia definition is as follows: “*Data enrichment* refers to operations or processes which refine, enhance or improve the raw data.”

Data Editing

Data editing refers to the process of reviewing and adjusting the acquired datasets. The editing controls the data quality. Editing methods are (i) interactive, (ii) selective, (iii) automatic, (iv) aggregating and (v) distribution.

Data Reduction

Data reduction enables the transformation of acquired information into an ordered, correct and simplified form. The reductions enable ingestion of meaningful data in the datasets. The basic concept is the reduction of multitudinous amount of data, and use of the meaningful parts. The reduction uses editing, scaling, coding, sorting, collating, smoothening, interpolating and preparing tabular summaries.

Data Wrangling

Data wrangling refers to the process of transforming and mapping the data. Results from analytics are then appropriate and valuable. For example, mapping enables data into another format, which makes it valuable for analytics and data visualizations.

Data Format used during Pre-Processing

Examples of formats for data transfer from (a) data storage, (b) analytics application, (b) service or (d) cloud can be:

- (i) Comma-separated values CSV (Example 1.9)
- (ii) Java Script Object Notation (JSON) as batches of object arrays or resource arrays (Example 3.3)
- (iii) Tag Length Value (TLV)
- (iv) Key-value pairs (Section 3.3.1)
- (v) Hash-key-value pairs (Example 3.2).

Need of data format conversion of data CSV, JSON, key-value pairs or other data from Data Store; for example, in the form of tables

CSV Format

An example is a table or Microsoft Excel file which needs conversion to CSV format. A student_record.xlsx converts to student_record.csv file. Comma-separated values (CSV) file refers to a plain text file which stores the table data of numbers and text. When processing for data visualization of Excel format file, the data conversion will be done from csv to xlsx format.

Each CSV file line is a data record. Each record consists of one or more fields, separated from each other by commas. RFC 4180 standard specifies the various specifications. A CSV file may also use space, tab or delimiter tab-separated formats for the values in the fields. This is a loose terminology. The following example explains the conversion process.

EXAMPLE 1.9

Consider the example of a table in a grade sheet. A CSV file is easily understandable when the table's first row specifies the column heads. Three columns of the first row are Subject Code, Subject Name and Grade and three columns of the second row are CS101, "Theory of Computations" and 7.8, as shown below:

Subject Code	Subject Name	Grade
CS101	"Theory of Computations"	7.8
CS102	"Computer Architecture"	7.2
-	-	-

SOLUTION

The first and second lines in the CSV file are:

Subject Code, Subject Name, Grade

CS101, "Theory of Computations", 7.8.

CS102, "Computer Architecture", 7.8.

The two consecutive double-quotes mean that one of the double quotes is retained in the text "Theory of Computations". That one specifies that characters are inside the double quotes and represent a string.

Data Format Conversions

Transferring the data may need pre-processing for data-format conversions. Data sources store need portability and usability. A number of different applications, services and tools need a specific format of data only. Pre-processing before their usages or

storage on cloud services is a must.

1.5.4 Data Store Export to Cloud

Figure 1.3 shows resulting data pre-processing, data mining, analysis, visualization and data store. The data exports to cloud services. The results integrate at the enterprise server or data warehouse.

Export of data from data sources to IBM, Microsoft, Oracle, Amazon, Rackspace or Hadoop cloud services

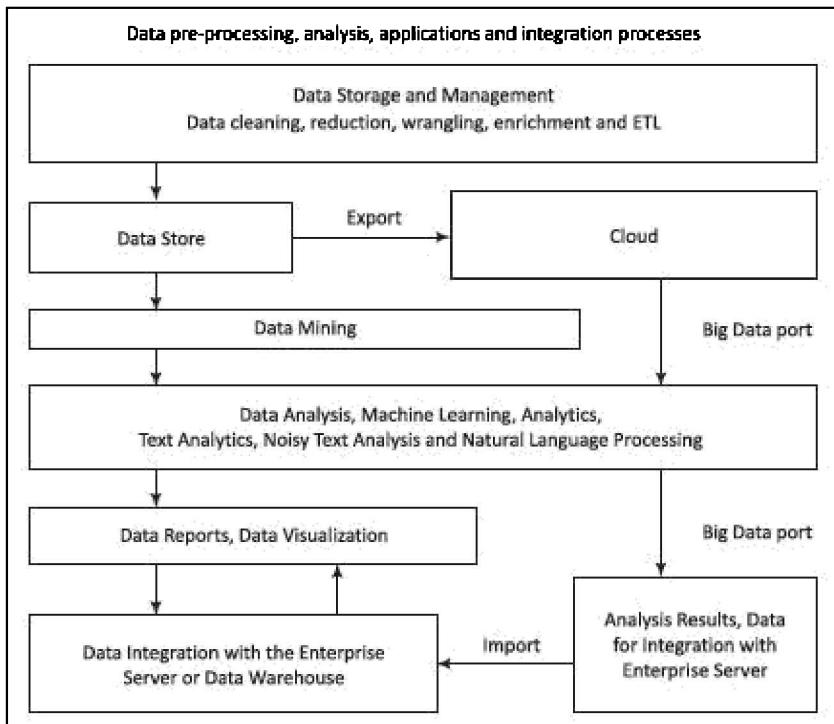


Figure 1.3 Data pre-processing, analysis, visualization, data store export

1.5.4.1 Cloud Services

Cloud offers various services. (Section 1.3.3) These services can be accessed through a cloud client (client application), such as a web browser, SQL or other client. Figure 1.4 shows data-store export from machines, files, computers, web servers and web services. The data exports to clouds, such as IBM, Microsoft, Oracle, Amazon, Rackspace, TCS, Tata Communications or Hadoop cloud services.

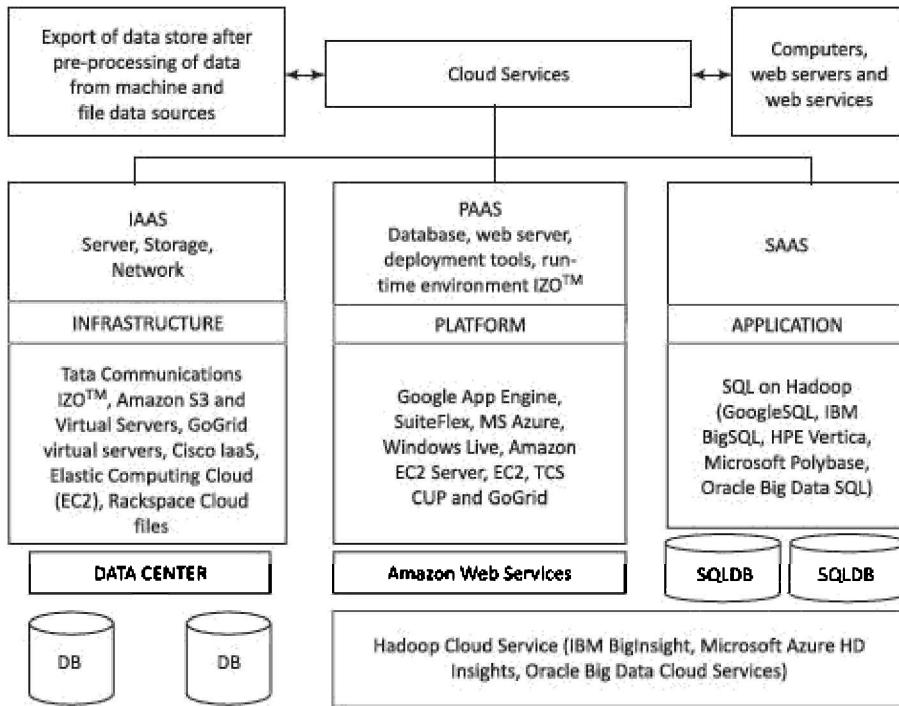


Figure 1.4 Data store export from machines, files, computers, web servers and web services

1.5.4.2 Export of Data to AWS and Rackspace Clouds

The following example explains the export processes to Amazon and Rackspace clouds.

EXAMPLE 1.10

- (a) How do the rows in MySQL database table export to Amazon AWS?
- (b) How do the rows in MySQL database table export to Rackspace?

SOLUTION

- (a) Following are the steps for export to an EC2 instance:
 - (i) A process pre-processes the data from data-rows at table in MySQL database and creates a CSV file.
 - (ii) An EC2 instance provides an AWS data pipeline.
 - (iii) The CSV file exports to Amazon S3 using pipeline. The CSV file then copies into an S3 bucket.⁷ Coping action deploys an EC2 instance.
 - (iv) AWS notification service (SNS) sends notification on completion.⁸

- (b) Following are the steps for export to Rackspace⁹:
- (i) An instance name has maximum 255 characters. One or more databases create a database instance. The process of creation can be configured to create an instance now or later. Each database can have a number of users.
 - (i) Default port number for binding of MySQL is port 3306.
 - (ii) A command `mysqldump - u root - p database_name > database_name.sql` exports to Rackspace cloud.
 - (iii) When a database is at a remote host then a command `mysqldump - h host_name - u user_name - p database_name > database_name.sql` exports to the cloud database.

Google cloud platform provides a cloud service called BigQuery.¹⁰ Figure 1.5 shows BigQuery cloud service at Google cloud platform. The data exports from a table or partition schema, JSON, CSV or AVRO files from data sources after the pre-processing.

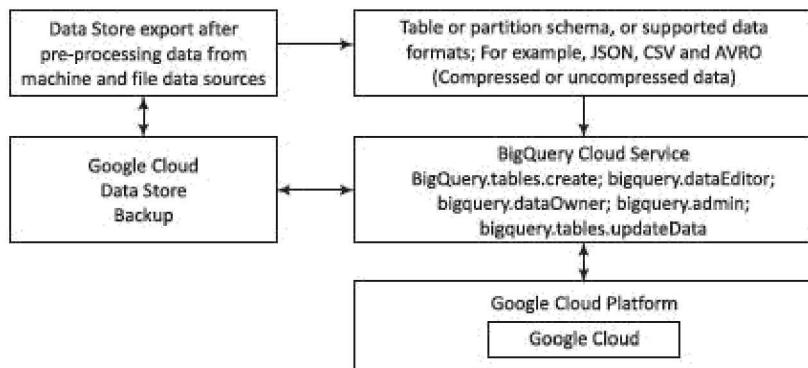


Figure 1.5 BigQuery cloud service at Google cloud platform

Data Store first pre-processes from machine and file data sources. Pre-processing transforms the data in table or partition schema or supported data formats. For example, JSON, CSV and AVRO. Data then exports in compressed or uncompressed data formats. (Avro is a data serialization system in Hadoop related tools for Big Data.)

Cloud service BigQuery consists of `bigquery.tables.create`; `bigquery.dataEditor`; `bigquery.dataOwner`; `bigquery.admin`; `bigquery.tables.updateData` and other service functions. Analytics uses Google Analytics 360. BigQuery cloud exports data to a Google cloud or cloud backup only.

Self-Assessment Exercise linked to LO 1.4

1. Why is data quality important in discovering new knowledge and decision making?
2. List the examples of cloud services for exporting data stores.
3. How is conversion to CSV file before data store beneficial? How is conversion to tables from CSV files from data store beneficial?
4. List the usages of three types of services that clouds offer. List Big Data cloud services, to data sources export from data store, and perform cloud during analytics, visualizations and intelligence discovery.
5. Consider databases storing the daily sales figures of chocolates, such as KitKat, Milk, Fruit and Nuts, Nougat and Oreo, each at every machine in Example 1.6(i). How will you name the data sources in ACVMs analytics? How will the ACVMs sales be analyzed for each type of chocolate using the data-source master-tables?

1.6 | DATA STORAGE AND ANALYSIS

The following subsections describe data storage and analysis, and comparison between Big Data management and analysis with traditional database management systems.

LO 1.5

Data storage, analysis, Comparison between traditional systems, such as Relational Database Management System (RDBMS), enterprise servers and data warehouse, Approaches for Big Data storage, processing and analytics

1.6.1 Data Storage and Management: Traditional Systems

1.6.1.1 Data Store with Structured or Semi-Structured Data

Traditional systems use structured or semi-structured data. The following example explains the sources and data store of structured data.

EXAMPLE 1.11

What are the sources of structured data store?

SOLUTION

The sources of structured data store are:

- Traditional relational database-management system (RDBMS) data, such as MySQL DB2, enterprise server and data warehouse

- Business process data which stores business events, such as registering a customer, taking an order, generating an invoice, and managing products in pre-defined formats. The data falls in the category of highly structured data. The data consists of transaction records, tables, relationships and metadata that build the information about the business data.
- Commercial transactions
- Banking/stock records
- E-commerce transactions data.

The following example explains the sources and data store of semi-structured data.

EXAMPLE 1.12

Give examples of sources of data store of semi-structured data.

SOLUTION

Examples of semi-structured data are:

- XML and JSON semi-structured documents^{7,8}
- A comma-separated values (CSV) file. The CSV stores tabular data in plain text. Each line is a data record. A record can have several fields, each field separated by a comma. Structured data, such as database include multiple relations but CSV does not consider the relations in a single CSV file. CSV cannot represent object-oriented databases or hierarchical data records. A CSV file is as follows:

Preeti, 1995, MCA, Object Oriented Programming, 8.75

Kirti, 2010, M.Tech., Mobile Operating System, 8.5

Data represent the data records for columns and rows of a table. Each row has names, year of passing, degree name, course name and grade point out of 10. Rows are separated by a new line and the columns by a comma.

JSON Object Data Formats: CSV does not represent object-oriented records, databases or hierarchical data records. JSON and XML represent semi-structured data and represent object-oriented and hierarchical data records. Example 3.5 explains CSV and JSON objects and the hierarchical data records in the JSON file format.

1.6.1.2 SQL

An RDBMS uses SQL (Structured Query Language). SQL is a language for viewing or changing (update, insert or append or delete) databases. It is a language for data access control, schema creation and data modifications.

SQL was originally based on the tuple relational calculus and relational algebra. SQL can embed within other languages using SQL modules, libraries and pre-compilers. SQL does the following:

1. *Create schema*, which is a structure which contains description of objects (base tables, views, constraints) created by a user. The user can describe the data and define the data in the database.
2. *Create catalog*, which consists of a set of schemas which describe the database.
3. *Data Definition Language (DDL)* for the commands which depicts a database, that include creating, altering and dropping of tables and establishing the constraints. A user can create and drop databases and tables, establish foreign keys, create view, stored procedure, functions in the database etc.
4. *Data Manipulation Language (DML)* for commands that maintain and query the database. A user can manipulate (INSERT/UPDATE) and access (SELECT) the data.
5. *Data Control Language (DCL)* for commands that control a database, and include administering of privileges and committing. A user can set (grant, add or revoke) permissions on tables, procedures and views.

SQL is a language for managing the RDBMS. A relational DB is a collection of data in multiple tables, which relate to each other through special fields, called keys (primary key, foreign key and unique key). Relational databases provide flexibilities. Relational database examples are MySQL PostGreSQL Oracle database, Informix, IBM DB2 and Microsoft SQL server.

1.6.1.3 Large Data Storage using RDBMS

RDBMS tables store data in a structured form. The tables have rows and columns. Data management of Data Store includes the provisions for privacy and security, data integration, compaction and fusion. The systems use machine-generated data, human-sourced data, and data from business processes (BP) and business intelligence (BI).

A set of keys and relational keys access the fields at tables, and retrieve data using queries (insert, modify, append, join or delete). RDBMSs use software for data administration also.

Online content associated with Practice Exercise 1.12 describes the use of tables in

Traditional systems
Relational database is collection of data into multiple tables which relates to each other through special fields, called keys

relational databases in detail.

1.6.1.4 Distributed Database Management System

A distributed DBMS (DDBMS) is a collection of logically interrelated databases at multiple system over a computer network. The features of a distributed database system are:

Distributed DB is a collection of logically interrelated databases at multiple systems over a computer network.

1. A collection of logically related databases.
2. Cooperation between databases in a transparent manner. Transparent means that each user within the system may access all of the data within all of the databases as if they were a single database.
3. Should be ‘location independent’ which means the user is unaware of where the data is located, and it is possible to move the data from one physical location to another without affecting the user.

1.6.1.5 In-Memory Column Formats Data

A columnar format in-memory allows faster data retrieval when only a few columns in a table need to be selected during query processing or aggregation. Data in a column are kept together in-memory in columnar format. A single memory access, therefore, loads many values at the column. An address increment to a next memory address for the next value is fast when compared to first computing the address of the next value, which is not the immediate next address. The following example explains the in-memory columnar format.

EXAMPLE 1.13

Consider analysis of monthly sales of chocolates on ACVMs (Example 1.6) in company’s annual profit reports.

- (i) How does sales analysis become easy in-memory columnar format?
- (ii) How does during an analysis the access is made to few columns in place of from entire datasets?

SOLUTION

All the column 1 values for several days’ record is physically together in-memory at consecutive addresses. All the column 2 values are then physically together at the next successive addresses. Then, the column 3 and other columns store at the columnar database in-memory.

The data stores for each record order in successive columns, so that the 100th entry at column 1 and the 100th entry for column 2 belong to the same record

and same input accessible from a single row-key. Column vector refers to a vector whose elements are values at column fields.

Analytics, therefore, can be executed faster when data is in the column format, and more rows and few columns need to be selected during analysis. Successive days' sales of each flavour of chocolate stores in successive values in one column from row r to $(r + 30)$ in a month, thirty row-keys for 30 days, and 365 row keys in a year.

Aggregation functions and other analysis functions are easy to run due to successive memory addresses for sales for each day for each flavour. Examples of aggregation functions are sum, count, maximum, minimum, average, minimum and maximum deviation from a specified value.

Online Analytical Processing (OLAP) in real-time transaction processing is fast when using in-memory column format tables. OLAP enables real-time analytics. The CPU accesses all columns in a single instance of access to the memory in columnar format in-memory data-storage.

Online Analytical Processing (OLAP) enables online viewing of analyzed data and visualization up to the desired granularity (fineness or coarseness) enables view by rolling up (finer granulates to coarse granulates data) or drilling down (coarser granulates data to finer granulates). OLAP enables obtaining online summarized information and automated reports for a large database.

Metadata describes the data. Pre-storing of calculated values provide consistently fast response. Result formats from the queries are based on Metadata.

1.6.1.6 In-Memory Row Format Databases

A row format in-memory allows much faster data processing during OLTP (online transaction processing). Refer Example 1.13. Each row record has corresponding values in multiple columns and the on-line values store at the consecutive memory addresses in row format. A specific day's sale of five different chocolate flavours is stored in consecutive columns c to $c+5$ at memory. A single instance of memory accesses loads values of all five flavours at successive columns during online processing. For example, the total number of chocolates sold computes online. Data is in-memory row-formats in stream and event analytics. The stream analytics method does continuous computation that happens as data is flowing through the system. Event analytics does computation on event and use event data for tracking and reporting events.

1.6.1.7 Enterprise Data-Store Server and Data Warehouse

Enterprise data, after *data cleaning* process, integrate with the server data at warehouse. Enterprise data server use data from several distributed sources which store data using

various technologies. All data *merge* using an integration tool. Integration enables collective viewing of the datasets at the data warehouse (Figure 1.3).

Enterprise data integration may also include integration with application(s), such as analytics, visualization, reporting, business intelligence and knowledge discovery. Heterogeneous systems execute complex integration processes when integrating at an enterprise server or data warehouse. Complex application-integration means the integration of heterogeneous application architectures and processes with the databases at the enterprise. Enterprise data warehouse store the databases, and data stores after integration, using tools from number of sources.

Online contents associated with Practice Exercises 1.9 and 1.10 give details of commercial solutions for complex application-integration of processes.

Following are some standardised business processes, as defined in the Oracle application-integration architecture:

1. Integrating and enhancing the existing systems and processes
2. Business intelligence
3. Data security and integrity
4. New business services/products (Web services)
5. Collaboration/knowledge management
6. Enterprise architecture/SOA
7. e-commerce
8. External customer services
9. Supply chain automation/visualization
10. Data centre optimization

Figure 1.6 shows Steps 1 to 5 in enterprise data integration and management with Big Data for high performance computing using local and cloud resources for analytics, applications and services.

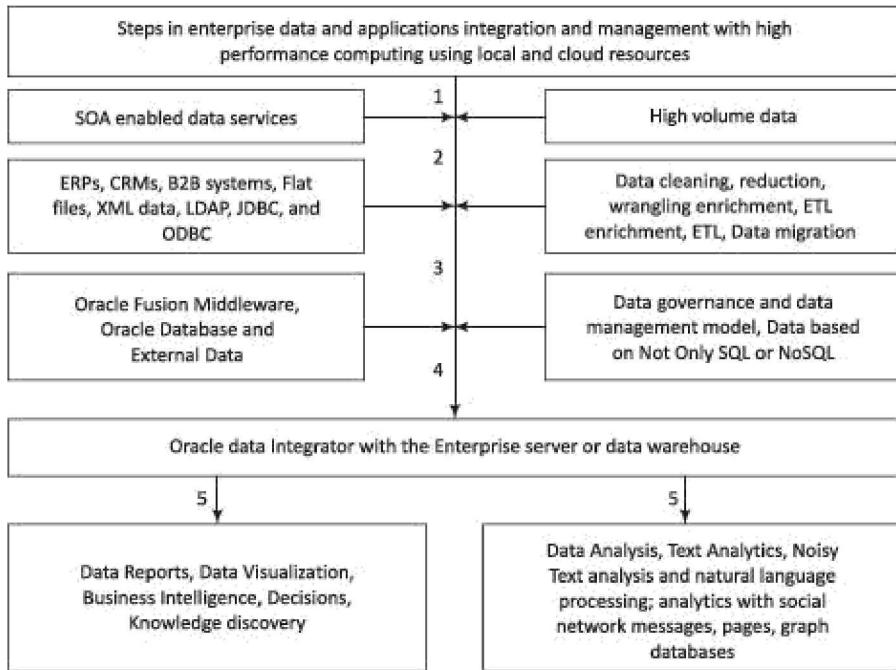


Figure 1.6 Steps 1 to 5 in Enterprise data integration and management with Big-Data for high performance computing using local and cloud resources for the analytics, applications and services

1.6.2 Big Data Storage

Following subsections describe Big Data storage concepts:

1.6.2.1 Big Data NoSQL or Not Only SQL

NoSQL databases are considered as semi-structured data. Big Data Store uses NoSQL. NOSQL stands for No SQL or Not Only SQL. The stores do not integrate with applications using SQL. NoSQL is also used in cloud data store. Features of NoSQL are as follows:

NOSQL or Not Only SQL
class of non-relational data storage systems, flexible data models and multiple schemas

1. It is a class of non-relational data storage systems, and the flexible data models and multiple schema:
- (i) Class consisting of uninterrupted key/value or big hash table [Dynamo (Amazon S3)]
 - (ii) Class consisting of unordered keys and using JSON (PNUTS)
 - (iii) Class consisting of ordered keys and semi-structured data storage systems [BigTable, Cassandra (used in Facebook/Apache) and HBase]

- (iv) Class consisting of JSON (MongoDB)
- (v) Class consisting of name/value in the text (CouchDB)
- (vi) May not use fixed table schema
- (vii) Do not use the JOINS
- (viii) Data written at one node can replicate at multiple nodes, therefore Data storage is fault-tolerant,
- (ix) May relax the ACID rules during the Data Store transactions.
- (x) Data Store can be partitioned and follows CAP theorem (out of three properties, consistency, availability and partitions, at least two must be there during the transactions)

Consistency means all copies have the same value like in traditional DBs. *Availability* means at least one copy is available in case a partition becomes inactive or fails. For example in web applications, the other copy in other partition is available. *Partition* means parts which are active but may not cooperate as in the distributed DBs.

1.6.2.2 Coexistence of Big Data, NoSQL and Traditional Data Stores

Figure 1.7 shows co-existence of data at server, SQL, RDBMS with NoSQL and Big Data at Hadoop, Spark, Mesos, S3 or compatible Clusters.

Table 1.4 gives various data sources for Big Data along with its examples of usages and the tools used.

Table 1.4 Various data sources and examples of usages and tools

Data Source	Examples of Usages	Example of Tools
Relational databases	Managing business applications involving structured data	Microsoft Access, Oracle, IBM DB2, SQL Server, MySQL, PostgreSQL Composite, SQL on Hadoop [HPE (Hewlett Packard Enterprise) Vertica, IBM BigSQL, Microsoft Polybase, Oracle Big Data SQL]
Analysis databases (MPP, columnar, In-memory)	High performance queries and analytics	Sybase IQ, Kognitio, Terradata, Netezza, Vertica, ParAccel, ParStream, Infobright, Vectorwise,
NoSQL databases (Key-value pairs, Columnar format, documents,	Key-value pairs, fast read/write using collections of name-value pairs for storing any type of data; Columnar format, documents,	Key-value pair databases: Riak DS (Data Store), OrientDB, Column format databases (HBase, Cassandra), Document oriented databases: CouchDB, MongoDB; Graph

Objects, graph)	objects, graph DBs and DSs	databases (Neo4j, Tetan)
Hadoop clusters	Ability to process large data sets across a distributed computing environment	Cloudera, Apache HDFS
Web applications	Access to data generated from web applications	Google Analytics, Twitter
Cloud data	Elastic scalable outsourced databases, and data administration services	Amazon Web Services, Rackspace, GoogleSQL
Individual data	Individual productivity	MS Excel, CSV, TLV, JSON, MIME type
Multidimensional	Well-defined bounded exploration especially popular for financial applications	Microsoft SQL Server Analysis Services
Social media data	Text data, images, videos	Twitter, LinkedIn

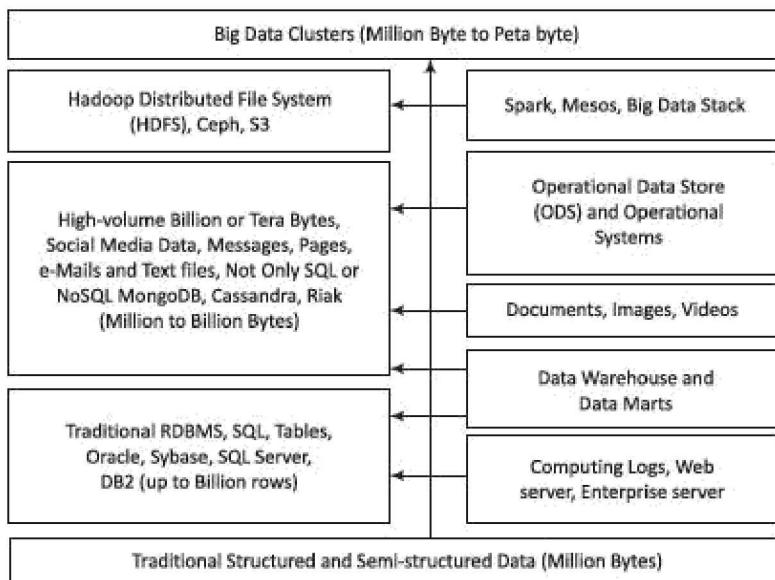


Figure 1.7 Coexistence of RDBMS for traditional server data, NoSQL and Hadoop, Spark and compatible Big Data Clusters

1.6.3 Big Data Platform

A Big Data platform supports large datasets and volume of data. The data generate at a higher velocity, in more varieties or in higher veracity. Managing Big Data requires large resources of MPPs, cloud, parallel processing and specialized tools. Bigdata platform

should provision tools and services for:

1. storage, processing and analytics,
2. developing, deploying, operating and managing Big Data environment,
3. reducing the complexity of multiple data sources and integration of applications into one cohesive solution,
4. custom development, querying and integration with other systems, and
5. the traditional as well as Big Data techniques.

Data management, storage and analytics of Big data captured at the companies and services require the following:

1. New innovative non-traditional methods of storage, processing and analytics
2. Distributed Data Stores
3. Creating scalable as well as elastic virtualized platform (cloud computing)
4. Huge volume of Data Stores
5. Massive parallelism
6. High speed networks
7. High performance processing, optimization and tuning
8. Data management model based on Not Only SQL or NoSQL
9. In-memory data column-formats transactions processing or *dual in-memory data* columns as well as row formats for OLAP and OLTP
10. Data retrieval, mining, reporting, visualization and analytics
11. Graph databases to enable analytics with social network messages, pages and data analytics
12. Machine learning or other approaches
13. Big data sources: Data storages, data warehouse, Oracle Big Data, MongoDB NoSQL, Cassandra NoSQL
14. Data sources: Sensors, Audit trail of Financial transactions data, external data such as Web, Social Media, weather data, health records data.

1.6.3.1 Hadoop

Big Data platform consists of Big Data storage(s), server(s) and data management and business intelligence software. Storage can deploy Hadoop Distributed File System (HDFS), NoSQL data stores, such as HBase, MongoDB, Cassandra. HDFS system is an open source

storage system. HDFS is a scaling, self-managing and self-healing file system.

The Hadoop system packages application-programming model. Hadoop is a scalable and reliable parallel computing platform. Hadoop manages Big Data distributed databases. Figure 1.8 shows Hadoop based Big Data environment. Small height cylinders represent MapReduce and big ones represent the Hadoop.

1.6.3.2 Mesos

Mesos v0.9 is a resources management platform which enables sharing of cluster of nodes by multiple frameworks and which has compatibility with an open analytics stack [data processing (Hive, Hadoop, HBase, Storm), data management (HDFS)].

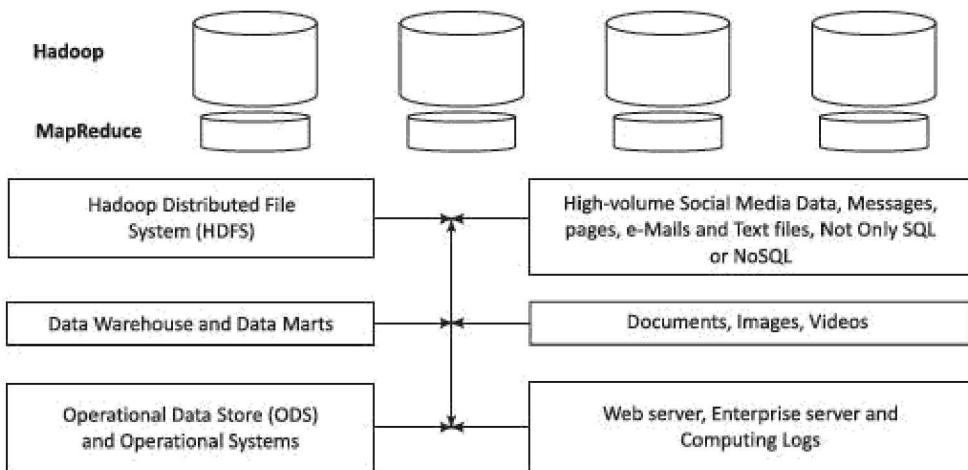


Figure 1.8 Hadoop based Big Data environment

1.6.3.3 Big Data Stack

A stack consists of a set of software components and data store units. Applications, machine-learning algorithms, analytics and visualization tools use Big Data Stack (BDS) at a cloud service, such as Amazon EC2, Azure or private cloud. The stack uses cluster of high performance machines.

Table 1.5 gives Big Data management, storage and processing tools.

Table 1.5 Tools for Big Data environment

Types	Examples
MapReduce	Hadoop, Apache Hive, Apache Pig, Cascading, Cascalog, mrjob (Python MapReduce library), Apache S4, MapR, Apple Acunu, Apache Flume, Apache Kafka
NoSQL Databases	MongoDB, Apache CouchDB, Apache Cassandra, Aerospike, Apache HBase, Hypertable

Processing	Spark, IBM BigSheets, PySpark, R, Yahoo! Pipes, Amazon Mechanical Turk, Datameer, Apache Solr/Lucene, ElasticSearch
Servers	Amazon EC2, S3, GoogleQuery, Google App Engine, AWS Elastic Beanstalk, Salesforce Heroku
Storage	Hadoop Distributed File System, Amazon S3, Mesos

1.6.4 Big Data Analytics

DBMS or RDBMS manages the traditional databases. Data analysis need pre-processing of raw data and gives information useful for decision making. Analysis brings order, structure and meaning to the collection of data. Data is collected and analyzed to answer questions, test the hypotheses or disprove theories.

1.6.4.1 Data Analytics Definition

Data Analytics can be formally defined as the statistical and mathematical data analysis that clusters, segments, ranks and predicts future possibilities. An important feature of data analytics is its predictive, forecasting and prescriptive capability. Analytics uses historical data and forecasts new values or results. Analytics suggests techniques which will provide the most efficient and beneficial results for an enterprise. Data analysis helps in finding business intelligence and helps in decision making.

Data analysis can be defined as,

“Analysis of data is a process of inspecting, cleaning, transforming and modeling data with the goal of discovering useful information, suggesting conclusions and supporting decision making.” (Wikipedia)

1.6.4.2 Phases in Analytics

Analytics has the following phases before deriving the new facts, providing business intelligence and generating new knowledge.

1. *Descriptive analytics* enables deriving the additional value from visualizations and reports
2. *Predictive analytics* is advanced analytics which enables extraction of new facts and knowledge, and then predicts/forecasts
3. *Prescriptive analytics* enable derivation of the additional value and undertake better decisions for new option(s) to maximize the profits
4. *Cognitive analytics* enables derivation of the additional value and undertake better

decisions.

Analytics integrates with the enterprise server or data warehouse.

Figure 1.9 shows an overview of a reference model for analytics architecture. The figure also shows on the right-hand side the Big Data file systems, machine learning algorithms and query languages and usage of the Hadoop ecosystem.

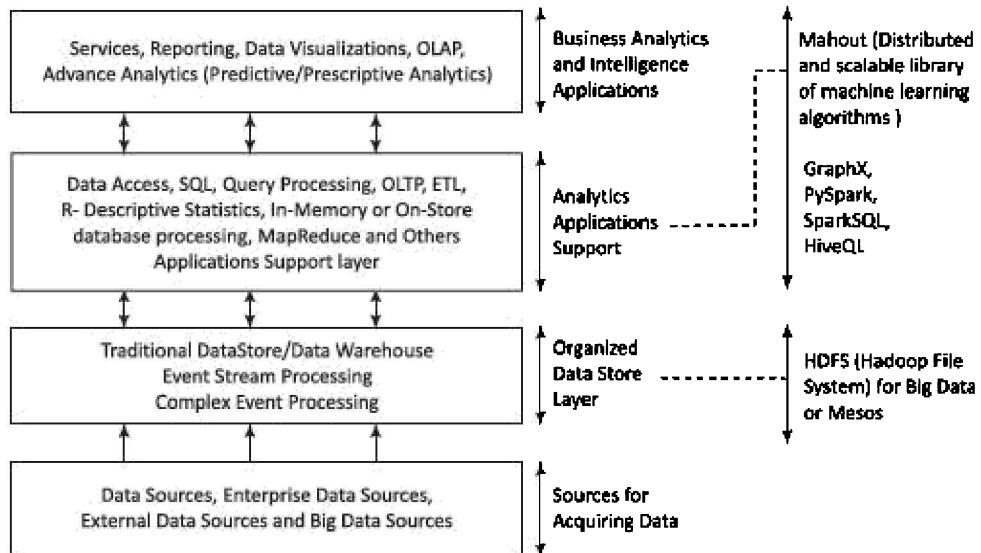


Figure 1.9 Traditional and Big Data analytics architecture reference model

The captured or stored data require a well-proven strategy to calculate, plan or analyze. When Big Data combine with high-powered data analysis, enterprise achieve valued business-related tasks. Examples are:

- Determine root causes of defects, faults and failures in minimum time.
- Deliver advertisements on mobiles or web, based on customer's location and buying habits.
- Detect offender before that affects the organization or society.

1.6.4.3 Berkeley Data Analytics Stack (BDAS)

The importance of Big Data lies in the fact that what one does with it rather than how big or large it is. Identify whether the gathered data is able to help in obtaining the following findings: 1) cost reduction, 2) time reduction, 3) new product planning and development, 4) smart decision making using predictive analytics and 5) knowledge discovery.

BDAS consisting of the data processing, data management and resource management layers

Big Data analytics need innovative as well as cost effective techniques. BDAS is an open-source data analytics stack for complex computations on Big Data.¹¹ It supports efficient, large-scale in-memory data processing, and thus enables user applications achieving three fundamental processing requirements; accuracy, time and cost.

Berkeley Data Analytics Stack (BDAS) consists of data processing, data management and resource management layers. Following list these:

1. Applications, AMP-Genomics and Carat run at the BDAS. Data processing software component provides in-memory processing which processes the data efficiently across the frameworks. AMP stands for Berkeley's Algorithms, Machines and Peoples Laboratory.
2. Data processing combines *batch*, *streaming* and *interactive* computations.
3. Resource management software component provides for sharing the infrastructure across various frameworks.

Figure 1.10 shows a four layers architecture for Big Data Stack that consists of Hadoop, MapReduce, Spark core and SparkSQL, Streaming, R, GraphX, MLlib, Mahout, Arrow and Kafka.

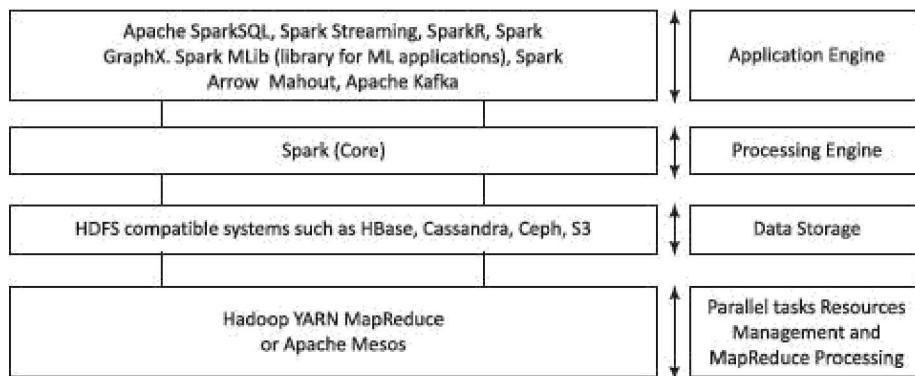


Figure 1.10 Four layers architecture for Big Data Stack consisting of Hadoop, MapReduce, Spark core and SparkSQL, Streaming, R, GraphX, MLlib, Mahout, Arrow and Kafka

Self-Assessment Exercise linked to LO 1.5

1. What are the traditional systems for data storage? How does in-memory columnar format help in OLAP? Give an example.
2. What are hierarchical and object oriented records?

3. What is enterprise server? How does enterprise server data store differ from a web server?
4. What are the functions of data integration software? How does application integration along with data integration help in business processes, intelligence and analytics?
5. What are the functions in SQL? List the differences between SQL data store and NoSQL data store.
6. How does a Big Data stack help in analytics tasks?
7. How does a Berkeley Data analytics stack help in analytics tasks?

1.7 BIG DATA ANALYTICS APPLICATIONS AND CASE STUDIES

Many applications such as social network and social media, cloud applications, public and commercial web sites, scientific experiments, simulators and e-government services generate Big Data. Big Data analytics find applications in many areas. Some of the popular ones are marketing, sales, health care, medicines, advertising etc. Following subsections describe these use cases, applications and case studies.

LO 1.6

Big Data use cases, applications and case studies in various fields, such as marketing, sales and healthcare

1.7.1 Big Data in Marketing and Sales

Data are important for most aspect of marketing, sales and advertising. Customer Value (CV) depends on three factors – quality, service and price. Big data analytics deploy large volume of data to identify and derive intelligence using predictive models about the individuals. The facts enable marketing companies to decide what products to sell.

A definition of marketing is the creation, communication and delivery of value to customers. Customer (desired) value means what a customer desires from a product. Customer (perceived) value means what the customer believes to have received from a product after purchase of the product. Customer value analytics (CVA) means analyzing what a customer really needs. CVA makes it possible for leading marketers, such as Amazon to deliver the consistent customer experiences. Following are the five application areas in order of the popularity of Big Data use cases:

1. CVA using the inputs of evaluated purchase patterns, preferences, quality, price and

- post sales servicing requirements
- 2. Operational analytics for optimizing company operations
- 3. Detection of frauds and compliances
- 4. New products and innovations in service
- 5. Enterprise data warehouse optimization.

An example of fraud is borrowing money on already mortgage assets. Example of timely compliances means returning the loan and interest installments by the borrowers.

A few examples in service-innovation are as follows: A company develops software and then offers services like Uber. Another example is of a company which develops software for hiring services, and then offers costly construction machinery and equipment. That service company might be rendering the services by hiring themselves from the multiple sources and locations of big construction companies.

Big data is providing marketing insights into (i) most effective content at each stage of a sales cycle, (ii) investment in improving the customer relationship management (CRM), (iii) addition to strategies for increasing customer lifetime value (CLTV), (iv) lowering of customer acquisition cost (CAC). Cloud services use Big Data analytics for CAC, CLTV and other metrics, the essentials in any cloud-based business

Big Data revolutionizes a number of areas of marketing and sales. Louis Columbus¹² recently listed the ways of usages. (Refer online content for solution of Practice Exercise 1.14.)

Contextual marketing means using an online marketing model in which a marketer sends to potential customers the targeted advertisements, which are based on the search terms during latest browsing patterns usage by customers.

For example, if a customer is searching an airline for flights on a specific date from Delhi to Bangalore, then a smart travel agency targeting that customer through advertisements will show him/her, at specific intervals, better options for another airline or different but cheap dates for travel or options in which price reduction occurs gradually.

The following example explains the use of search engine optimization.

EXAMPLE 1.14

Why does the search engine at a company product website of a travel agency need optimization?

SOLUTION

Consider a travel agency website offers search results for flights between two destinations A and C, which do not connect directly. The search shows the results in order of increasing travel cost through stopover at an intermediate airport B. Assume that search results show up just mechanically, without embedding intelligence and optimization. The customers find uncomfortable solutions with such searches. The searches show the cheaper options but sometimes show results such as the customer would reach C through stopover at B after 8 hours or even sometimes on the next day.

The searches at that travel agency do not consider stopover options at different Bs, options available in different airlines to cut short travel time from B to C at cheaper costs, or newly introduced flights. The searches therefore need optimization for parameters of travel cost, multiple intermediate stopovers and airlines that will provide maximum customer convenience as well as cost.

Big data algorithms and advanced analytics techniques enable price optimization for a given product or service, and pricing decisions, especially in the commodity driven industries where products are inelastic. Inelastic product means a situation in which the service, required quantity or supply of a product remains unaffected by the price changes.

1.7.1.1 Big Data Analytics in Detection of Marketing Frauds

Fraud detection is vital to prevent financial loses to users. Fraud means someone deceiving deliberately. For example, mortgaging the same assets to multiple financial institutions, compromising customer data and transferring customer information to third party, falsifying company information to financial institutions, marketing product with compromising quality, marketing product with service level different from the promised, stealing intellectual property, and much more.

Big data deployment in fraud detection related to marketing

Big Data analytics enable fraud detection. Big Data usages has the following features-for enabling detection and prevention of frauds:

1. Fusing of existing data at an enterprise data warehouse with data from sources such as social media, websites, blogs, e-mails, and thus enriching existing data
2. Using multiple sources of data and connecting with many applications
3. Providing greater insights using querying of the multiple source data
4. Analyzing data which enable structured reports and visualization
5. Providing high volume data mining, new innovative applications and thus leading to new business intelligence and knowledge discovery

6. Making it less difficult and faster detection of threats, and predict likely frauds by using various data and information publicly available.

1.7.1.2 Big Data Risks

Large volume and velocity of Big Data provide greater insights but also associate risks with the data used. Data included may be erroneous, less accurate or far from reality. Analytics introduces new errors due to such data.

Big Data large volume, velocity and veracity data provide greater insights for data security, privacy, increasing costs, and bad analytics and bad data risks associates

Big Data can cause potential harm to individuals. For example, when someone puts false or distorted data about an individual in a blog, Facebook post, WhatsApp groups or tweets, the individual may suffer loss of educational opportunity, job or credit for his/her urgent needs. A company may suffer financial losses.

Five data risks, described by Bernard Marr are data security, data privacy breach, costs affecting profits, bad analytics and bad data.¹³ (Solutions in online content accompanying the book for Practice Exercise 1.15)

Companies need to take risks of using Big Data and design appropriate risk management procedures. They have to implement robust risk management processes and ensure reliable predictions. Corporate, society and individuals must act with responsibility.

1.7.1.3 Big Data Credit Risk Management

Financial institutions, such as banks, extend loans to industrial and household sectors. These institutions in many countries face credit risks, mainly risks of (i) loan defaults, (ii) timely return of interests and principal amount. Financing institutions are keen to get insights into the following:

1. Identifying high credit rating business groups and individuals,
2. Identifying risk involved before lending money
3. Identifying industrial sectors with greater risks
4. Identifying types of employees (such as daily wage earners in construction sites) and businesses (such as oil exploration) with greater risks
5. Anticipating liquidity issues (availability of money for further issue of credit and rescheduling credit installments) over the years.

The insight using Big Data decreases the default rates in returning of loan, greater accuracy in issuing credit and faster identification of the non-payment or fraud issues of the loan receiving entities. (Example of fraud is using the same assets for drawing credit from two or more institutions or hiding earlier outstanding loans and loan defaults.)

One innovative way to manage credit risks and liquidity risks is use of available data and Big Data. High volume of data analysis gives greater insight into the default patterns, emerging patterns and thus credit risks.

Big Data analytics monitors social media, interactions data, contact addresses, mobile numbers, website, financial status, activities or job changes to find the emerging credit risk that may affect a customer loan returning capacity. Digital footprints across social media provide a valuable alternative data source for credit risk analysis. The data companies assist in rating the customer in application processing and also during the period of repayment of a loan. Friends on Facebook and their credit rating, comments and assets posted also help in determining the risks.

The data insights from the analytics lead to credit and liquidity risk management and faster reactions. Three benefits are (i) minimize the non-payments and frauds, (ii) identifying new credit opportunities, new customers and revenue streams, thereby broadening the company high credit rating customers base and (iii) marketing to low risk businesses and households.

1.7.1.4 Big Data And Algorithmic Trading

Wikipedia gives a definition of algorithm trading as follows: “Algorithmic trading is a method of executing a large order (too large to fill all at once) using automated pre-programmed trading instructions accounting for variables such as time, price and volume.” Complex mathematics computations enable algorithmic trading and business investment decisions to buy and sell. The input data are insights gathered from the risk analysis of market data. Big data bigger volume, velocity and variety in the trading provide an edge over other trading entities

1.7.2 Big Data and Healthcare

Big Data analytics in health care use the following data sources: (i) clinical records, (ii) pharmacy records, (3) electronic medical records (4) diagnosis logs and notes and (v) additional data, such as deviations from person usual activities, medical leaves from job, social interactions. Healthcare analytics using Big Data can facilitate the following:

Big Data large volume, velocity and veracity data provide greater insights in health care systems and medicine

1. Provisioning of value-based and customer-centric healthcare,
2. Utilizing the ‘Internet of Things’ for health care
3. Preventing fraud, waste, abuse in the healthcare industry and reduce healthcare costs (Examples of frauds are excessive or duplicate claims for clinical and hospital treatments. Example of waste is unnecessary tests. Abuse means unnecessary use of medicines, such as tonics and testing facilities.)

4. Improving outcomes
5. Monitoring patients in real time.

Value-based and customer-centric healthcare means cost effective patient care by improving healthcare quality using latest knowledge, usages of electronic health and medical records and improving coordination among the healthcare providing agencies, which reduce avoidable overuse and healthcare costs.

Healthcare Internet of Things create unstructured data. The data enables the monitoring of the devices data for patient parameters, such as glucose, BP, ECGs and necessities of visiting physicians.

Prevention of fraud, waste, and abuse uses Big Data predictive analytics and help resolve excessive or duplicate claims in a systematic manner. The analytics of patient records and billing help in detecting, anomalies such as overutilization of services in short intervals, different hospitals in different locations simultaneously, or identical prescriptions for the same patient filed from multiple locations.

Improving outcomes is possible by accurately diagnosing patient conditions, early diagnosis, predicting problems such as congestive heart failure, anticipating and avoiding complications, matching treatments with outcomes and predicting patients at risk for disease or readmission.

Patient real-time monitoring uses machine learning algorithms which process real-time events. They provide physicians the insights to help them make life-saving decisions and allow for effective interventions. The process automation sends the alerts to care providers and informs them instantly about changes in the condition of a patient.

1.7.3 Big Data in Medicine

Big Data analytics deploys large volume of data to identify and derive intelligence using predictive models about individuals. Big Data driven approaches help in research in medicine which can help patients. Big Data offers potential to transform medicine and the healthcare system—Dr. Eric Schadt and Sastry Chilukuri.¹⁴

Following are some findings: building the health profiles of individual patients and predicting models for diagnosing better and offer better treatment,

1. Aggregating large volume and variety of information around from multiple sources the DNAs, proteins, and metabolites to cells, tissues, organs, organisms, and ecosystems, that can enhance the understanding of biology of diseases. Big data creates patterns and models by data mining and help in better understanding and research,

2. Deploying wearable devices data, the devices data records during active as well as inactive periods, provide better understanding of patient health, and better risk profiling the user for certain diseases,

1.7.4 Big Data in Advertising

The impact of Big Data is tremendous on the digital advertising industry. The digital advertising industry sends advertisements using SMS, e-mails, WhatsApp, LinkedIn, Facebook, Twitter and other mediums.

Big Data technology and analytics provide insights, patterns and models, which relate the media exposure of all consumers to the purchase activity of all consumers using multiple digital channels. Big Data help in identity management and can provide an advertising mix for building better branding exercises.

Big Data captures data of multiple sources in large volume, velocity and variety of data unstructured and enriches the structured data at the enterprise data warehouse. Big data real time analytics provide emerging trends and patterns, and gain actionable insights for facing competitions from similar products. The data helps digital advertisers to discover new relationships, lesser competitive regions and areas.

Success from advertisements depend on collection, analyzing and mining. The new insights enable the personalization and targeting the online, social media and mobile for advertisements called hyper-localized advertising.

Nielson Inc. CEO, Mitch Barns described Big Data's big impact on the future of advertising. Advertising nowadays limits no longer to TV, radio and print. Advertisers use along with these multiple devices and mediums. For example, advertisement of the introduction of new courses by an institution or introduction of new flights by an Airline needs media other than TV and requires targeted and cost effective solutions.

Advertising on digital medium needs optimization. Too much usage can also effect negatively. Phone calls, SMSs, e-mail-based advertisements can be nuisance if sent without appropriate researching on the potential targets. The analytics help in this direction. The usage of Big Data after appropriate filtering and elimination is crucial enabler of BigData Analytics with appropriate data, data forms and data handling in the right manner.

Self-Assessment Exercise linked to LO 1.6

1. How do data inputs help in Big Data based Customer value analytics?

Big data real time analytics for faster insights, emerging trends and patterns, and gain actionable insights for facing competitions from similar products in digital advertising and building relationships

2. How does Big Data help in credit risk management in financial institutions?
3. How does Big Data Analytics enable prevention of fraud, waste and abuse of healthcare system?
4. Why does Big Data offer the potential to transform the medicine and healthcare system?
5. Why are the Cloud services used for Big Data Analytics for customer acquisition, customer lifetime value analytics and other metrics?



KEY CONCEPTS

3Vs

4Vs

application integration

Big Data analytics

Big Data characteristics

Big Data types

Business intelligence

Business process

cloud

cost of acquisition

credit risk

CSV data format

customer value

data

data analytics

data architecture

database

data cleaning

data consumption

data cube

data ingestion

data integration
data management
data mining
data patterns
data pre-processing
data source
data store
data warehouse
descriptive analytics
distributed database
ELT
Enterprise server
ETL
event analytics
Hadoop
Hash-key value pair
in-memory column format
in-memory row format
JSON data format
key-value pair
knowledge discovery
logic machine
machine learning
Management Information Services
Massively Parallel Processing
multi-dimensional data cube
multi structured data
Noise
NoSQL
OLAP
OLTP

online analytic processing

Outlier

predictive analytics

prescriptive analytics

RDBMS

real-time analytics

semi-structured data

SQL

stream analytics

structured data

web data

Learning Outcomes

LO 1.1

- Data is raw information, usually in the form of facts or statistics that one can analyze or that one can use for further calculations and computations.
- Data are structured, semi-structured, multi-structured and unstructured.
- Four types of Big Data: Social networks and web data, transactions and business processes data, machine generated data and human generated data.
- Analytics helps in gathering, organizing, analyzing and reporting meaningful patterns in data. Analytics leads to communicate to user the meaningful patterns in data, helps data visualization, predictions and knowledge discovery. .
- Big Data is a collection of datasets very large and /or complex that traditional data processing applications are inadequate and has 3Vs or 4Vs as characteristics—volume, velocity, variety and veracity.

LO 1.2

- Scalability is the capability of a system to handle increasing workloads. Analytics scalability for Big Data uses distributed computing model. Scalability means multiple independent computational tasks submitted to multiple computing nodes which

function coherently.

- Analytic scalability for Big Data deploys parallel computing, massive parallel processing, cloud computing, cluster or grid computing.

LO 1.3

- Big Data architecture provides the logical and/or physical layout/structure of how big data will be stored, accessed and managed.
- Data architecture design consists of five logical layers: data sources identification, ingestion and acquisition, data storage, data processing and consumption, such as applications, analysis, visualizations, business processes, business intelligence, knowledge discovery.
- Management functions enable controlling, protecting, delivering and enhancing the value of data and information assets.

LO 1.4

- Data sources are data-repositories, such as RDBMS and spreadsheets from which the application seeks the data. Data sources are machines which drive data from data creating programs or form data store.
- High quality data means data with five R's: Relevancy, recency, range, robustness and reliability.
- Data-Store exports after pre-processing the data from data sources, servers, computers and service to the cloud. Cloud services and platforms can be sourced from IBM, Microsoft, Oracle, Google, Amazon, Rackspace, TCS and Tata Communications.
- Big Data applications use cloud services: Hadoop Cloud Service (IBM BigInsight, Microsoft Azure HD Insights, Oracle Big Data Cloud Services and SQL on Hadoop. SQL used are Apache SparkSQL, GoogleSQL, IBM BigSQL, HPE Vertica, Microsoft Polybase, Oracle Big Data SQL or Google BigQuery).

LO 1.5

- Traditional systems use structured or semi-structured data, tables, RDBMS such as DB2, MySQL, JSON and XML represent semi-structured data and are best suited for representing object oriented records or hierarchical data records.
- Big Data systems use new innovative non-traditional methods of storage, processing and analytics. They use distributed Data Stores. Big Data tools use scalable as well as elastic virtualized platform (cloud computing), huge volume of Data Stores, massive

parallelism, high-speed networks and graph databases. Analytics deploy social network messages and pages for Big Data analytics.

- Big Data storage can deploy Hadoop distributed file system, MongoDB, NoSql data stores. For example, HBase, Cassandra. HDFS are open source storage systems.

LO 1.6

- Data are important for most aspects of marketing and sales, credit risks management, fraud detection, healthcare, medicine and advertising.
- Big data analytics deploy a large volume of data to identify and derive intelligence using predictive models about individuals.
- Big Data analytics enables fraud detection and helps manage credit risks and liquidity risks.
- Big Data analytics also uses social media, interactions data, contact addresses, mobile numbers, website, financial status, activities or job changes for credit risk analysis.

Objective Type Questions

Select one correct-answer option for each questions below:

1.1 Data are usually required for:

- (a) Calculation
- (b) Planning
- (c) Input to a software tool
- (d) Calculate, plan, analyze and visualize something, obtaining intelligence or discover new knowledge

1.2 *Web data* is (i) data present at web servers, (ii) data accessible using the Internet, (iii) data which can be used in mobile and web applications, (iv) information in the form of documents and other web resources, (v) data at documents and resources which are accessible from the Internet such that each resource identifies by an URL of the data server store, and (vi) data in the documents interlink by hypertext links, and accessed using the Internet.

- (a) all are true
- (b) all except ii

(c) all except iii and iv

(d) ii to vi

1.3 Big Data analytics (i) deal with a large amount of data, (ii) manage, organize, process, analyze, share using traditional software tools running on require hundreds of computing nodes and large volume of storage devices, (iii) deal with fast generation of needed data, (iv) results in quick processing, analysis and usages, and has increased complexity due to multi-structured, (v) need processing of complex applications with large datasets, and (vi) deal with variety of data, various forms and formats, such as sensors, machine generated data, social media data

(a) iii to vii

(b) all except ii

(c) all except vi

(d) all

1.4 Big Data has (i) structured, semi-structured, and unstructured data formats, (ii) unstructured data format, (iii) stores as column-oriented, record-based, graph-based, hashed or key/value pairs, (iv) stores as column-oriented, row-oriented, and graph-based, (v) batch or real time processing needs, and (vi) real time processing needs.

(a) i, iii and v

(b) ii to v

(c) ii, iv and vi

(d) all except ii

1.5 Data architecture design considers:

(a) Four design layers with the lowest being identification of internal and external data sources

(b) Four design layers with the lowest being ingestion strategy and acquisition, and next identification of internal and external data sources.

(c) Five layers with data consumption for analytics, business processes, business intelligence, data mining, pattern recognition and knowledge discovery

(d) Five layers with data storage, processing and analytics being the highest layer

1.6 Data sources:

- (i) In the Microsoft Applications are of two types: machine data sources and file data sources.
 - (ii) In the Oracle applications are of three types: file data sources, database data sources and logic-machine data sources which use the network functions or a server.
 - (iii) In the IBM applications are of three types: machine data sources, database instances data sources and file data sources.
 - (iv) Can be sensors, sensor networks, devices, controllers, intelligent edge nodes in industrial M2M and signals from the machines.
 - (v) Data is of high quality if they enable all the required operations, analysis, decisions, planning, and knowledge discovery.
 - (vi) A definition for high quality data can be five R's: Relevancy, recency, range, robustness and reliability.
 - (vii) Data noise, outliers, missing-values and duplicate-values affect the data quality. Applications such as analytics, data visualization and data mining need data cleaning, integrity, enrichment, editing, reduction and/or wrangling.
- (a) all
 - (b) all except ii, v and vi
 - (c) all except vi
 - (d) (d) all except ii and iii

1.7 NoSQL features are:

- (i) The systems do not use the concept of joins (in distributed data storage systems)
 - (ii) Data written at one node and replicates to multiple nodes, therefore identical and fault-tolerant, and can be partitioned
 - (iii) Can offer relaxation in one or more of the ACID properties
 - (iv) Out of three properties (consistency, availability and partitions), two are at least present for the application/service/process.
- (a) i, ii and iii
 - (b) i to iv
 - (c) all except iii
 - (d) all except ii

1.8 Big Data platform should enable the following:

- (i) Storage, processing and analytics
 - (ii) Developing, deploying, operating and managing a Big Data environment in an enterprise
 - (iii) Reducing the complexity of multiple data sources and integrate the applications into one cohesive solution
 - (iv) Custom development, querying and integration with other systems involve the complexity
 - (v) Needs traditional as well as new and innovative techniques.
- (a) i to iii
 - (b) all except v
 - (c) all except iv
 - (d) all are true

1.9 *Vertical scalability* means scaling up by:

- (a) Using the giving system resources and increasing the systems analytics, reporting and visualization capabilities requiring additional ways to solve problems of greater complexities
- (b) Adding computers in parallel
- (c) Adding computers serially
- (d) Adding computers in serially as well as parallel

1.10 *Grid Computing* refers to (i) distributed computing, in which a group of computers from several locations are connected with each other to achieve a common task, (ii) remotely connected computers using Internet, (iii) computer resources heterogeneously and geographically disperse, (iv) a group of computers that might spread over remotely forming a grid, (v) computing using cloud, and (vi) computations and no under-performance even on failure of any of the participating nodes.

- (a) all
- (b) all except ii, v and vi
- (c) all except vi
- (d) all except ii and iii

1.11 Cloud computing environment (i) performs parallel and distributed computing for processing and analyzing large datasets on computing nodes, (ii) is on-demand service (iii) enables software, infrastructure and platform resource pooling, (iv) has scalability, (v) has no accountability, and (vi) has restricted network access.

- (a) all are true
- (b) all except v
- (c) i to iv
- (d) all except vi

1.12 Predictive analytics (i) predicts the trends, (ii) enable undertaking of the preventive maintenance in future from the earlier analyzes of equipment and device failure rates, (iii) enable managing the future campaigns and adopting integrated marketing strategy using previous studies of effect of campaigns at different media types, regions, targeted age groups, (iv) predicts by identifying patterns, clusters with similar behaviour, and (v) predicts based on earlier anomalous features detection, anomaly detection and filtering.

- (a) i to iii
- (b) all except iv
- (c) all except v
- (d) all are true

1.13 Automatic Chocolate Vending Machines company can use for selling (i) event analytics followed next by predictive analytics, (ii) manage each flavour supply-chain maintenance with optimization, (iii) manage the regular preventive maintenance of the machines, (iv) predict about changes in user preferences for chocolates in general and for specific flavours and (v) predict future festive season sales, (vi) visualize with finer and coarse granulates and multi-dimensional data cubes, (vii) predicts declining or rising sales, and (viii) plan strategies for boosting sales.

- (a) iv, v and vii
- (b) all except iii and viii
- (c) all
- (d) all except vi

1.14 Use Cases for Marketing and Sales Application areas are: (i) customer value analytics

using the inputs of evaluated purchase patterns preferences, quality, price and post sales servicing requirements, (ii) operational analytics for company operations optimization, (iii) detection of frauds and compliances, (iv) new products and services innovation, and (v) enterprise Data Warehouse Optimization.

- (a) ii to v
- (b) all except ii
- (c) all five and have popularity in order from (i) to (v)
- (d) all except iii

1.15 Big Data usage risks are: (i) data security risk, (ii) data privacy risk, (iii) cost affecting the profits, (iv) bad analytics results, (v) bad data, and (vi) data filtering.

- (a) i and ii
- (b) all except iii
- (c) all except v
- (d) all are true except vi

1.16 Automotive Maintenance Service Center Application in a company can (i) use analytics followed by predictive analytics, (ii) predict failure of components from periodic analysis of data, (iii) provide emergency services, (iv) improve the quality of components in future cars, (v) record driver rash driving habits and issue warnings, (vi) understand customer preferences, (vii) manage regular preventive maintenance of the automobile, (viii) visualization with finer and coarse granulates and multi-dimensional data cubes for maintenance needs, (ix) predict declining or rising sales, and (x) plan strategies for boosting sales.

- (a) i to vii
- (b) all except iv and vii
- (c) all except vi, viii and ix
- (d) all except vii and viii

Review Questions

1.1 Describe the data, web data and Big Data. (LO 1.1)

1.2 Draw a diagram showing evolution of Big Data and their characteristics over the time as size, complexity increased and as unstructured data increased. (LO 1.1)

- 1.3 What do you mean by 3Vs characteristics of Big Data? What are the challenges faced from large growth in volume of data? **(LO 1.1)**
- 1.4 What do you mean by analytical scalability? What are vertical scalability and horizontal scalability? **(LO 1.2)**
- 1.5 Explain uses of massive parallel processing and cluster computing in Big Data scenario. **(LO 1.2)**
- 1.6 Describe grid computing features. Compare these with cluster computing. **(LO 1.2)**
- 1.7 Define Big Data architecture. Draw five layers in architecture design and explain functions in each layer. **(LO 1.3)**
- 1.8 What do you mean by data management? List the data management functions. **(LO 1.3)**
- 1.6 What are the data sources considered in Microsoft Storage applications, IBM databases and Oracle Data Stores? **(LO 1.4)**
- 1.9 What do you mean by the data noise, outliers, duplicate data and data anomaly? Why does the filtering require during pre-processing? Explain the following data processing steps: inspecting, cleaning, transforming, modeling and visualizing data. **(LO 1.4)**
- 1.10 Show using a figure how data store export using machine data sources and file data sources, computers, web servers, web services, Amazon, Rackspace and Hadoop cloud services. **(LO 1.4)**
- 1.11 How do the table rows in MySQL database export to Amazon AWS and Rackspace? **(LO 1.4)**
- 1.12 Define distributed databases. How do they differ from distributed Data Stores? Describe features of distributed database databases. **(LO 1.6)**
- 1.13 How does the data analytics enable predictive, forecasting and prescriptive capabilities? How does the data analytics enable use of historical data to forecast potential values or results? Describe methods of analytics results reporting and visualization. **(LO 1.6)**
- 1.14 What are the requirements for data management, storage and analytics of captured Big data at the Companies and services? **(LO 1.6)**
- 1.15 Explain traditional and Big Data analytics architecture reference models. **(LO 1.6)**
- 1.16 Describe ways of usages of Big Data analytics in marketing, sales and advertising.

(LO 1.6)

1.17 What are the risks in Big Data usages? How does Big Data used in credit risk management? **(LO 1.6)**

1.18 Describe ways of usages of Big Data analytics in healthcare systems and medicine. **(LO 1.6)**

Practice Exercises

- 1.1 Diagrammatically show sources of structured, semi-structured, multi-structured and unstructured data. **(LO 1.1)**
- 1.2 Give examples of data resources at the enterprise server of an education institution. **(LO 1.1)**
- 1.3 List the unstructured data forms used in ‘Automotive Components and Predictive Automotive Maintenance Services’. **(LO 1.1)**
- 1.4 List usages of Big Data analytics in a company for car manufacturing, marketing, sales and maintenance of car service centres. **(LO 1.1)**
- 1.5 Estimate how many massively parallel processing nodes will be required to process data at 4 Tera instructions per second when a single processor processes 1 Gega instructions per second. Assume 10% time is taken up in inter-process communications. **(LO 1.2)**
- 1.6 Show architectural design layers in ‘Automotive Components and Predictive Automotive Maintenance Services’. **(LO 1.3)**
- 1.7 Take examples given in Examples 1.9. Consider example of a table in a student examination grade sheet for a semester. Fill the fields with presume values for two students in the same semester in the same course. Create CSV and JSON files. What are the benefits you can foresee in using JSON file in this case? **(LO 1.4)**
- 1.8 Take a student grade sheet in a semester examination in a course. Fill the presumed values for a student. Now create the hash and key-value pairs associated with a hash in traditional data. **(LO 1.5)**
- 1.9 Give details of the features of IBM IIS and Oracle Data Integrator. **(LO 1.5)**
- 1.10 Give details of features of integration and application integration solutions: Microsoft SQL-Server Integration-Services (SSIS), Informatica and Pentaho data-integration, SAS data-management advanced and SAP® BusinessObjects™

Integration. (LO 1.5)

- 1.11 What are the analytics phases? Take example of automotive maintenance and services. How will the results of analytics be used? (LO 1.5)
 - 1.12 How will RDBMS make tables for the sales data of ACVMs? [Refer Example 1.6(i)] (LO 1.5)
 - 1.13 Write five applications for NoSQL Databases? (LO 1.5)
 - 1.14 Give details of Louis Columbus ten ways using which Big Data analytics is revolutionizing marketing and sales. [Section 1.7.1] (LO 1.6)
 - 1.15 Describe five data risks, described by Bernard Marr. [Section 1.7.1.2] (LO 1.6)
-

- 1 <http://www.gartner.com/it-glossary/big-data>
- 2 <https://statswiki.unece.org/display/bigdata/Classification+of+Types+of+Big+Data>
- 3 <https://www.ibm.com/developerworks/library/bd-archpatterns1/>
- 4 <https://docs.microsoft.com/en-us/sql/odbc/reference/data-sources>
- 5 https://docs.oracle.com/cd/E17984_01/doc.898/e14695/undrstnd_datasources.htm
- 6 https://www.ibm.com/support/knowledgecenter/en/SSMPHH_9.5.0/com.ibm.guardium95.doc/common_tools/topics/datasources.html
- 7 <http://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-object-copyactivity.html>
- 8 <http://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-object-snsalarm.html>
- 9 <https://support.rackspace.com/how-to/cloud-database-instance-parameters/>
- 10 <https://cloud.google.com/bigquery/docs/loading-data>
- 11 <https://amplab.cs.berkeley.edu/software/>
- 12 <https://www.forbes.com/sites/louis columbus/2016/05/09/ten-ways-big-data-is-revolutionizing-marketing-and-sales/#5e90bc21cff3>
- 13 <https://www.linkedin.com/pulse/5-biggest-risks-big-data-bernard-marr>

Note:

- Level 1 & Level 2 category
- Level 3 & Level 4 category
- Level 5 & Level 6 category

Chapter 2

Introduction to Hadoop

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 2.1 Get conceptual understanding of Hadoop core, components of Hadoop ecosystem, and streaming and pipe interfaces for inputs to MapReduce
- LO 2.2 Get understanding of Hadoop Distributed File System (HDFS), and physical-organization of nodes for computing at clusters of large-scale files
- LO 2.3 Get knowledge of MapReduce Daemon framework, and MapReduce programming model
- LO 2.4 Get knowledge of functions of Hadoop YARN, management and scheduling of resources, and parallel processing of the application-tasks
- LO 2.5 Get introduced to functions of Hadoop ecosystem-tools

RECALL FROM CHAPTER 1

Requirements for Big Data processing and analytics are:

1. Huge volume of data stores

2. Flexible, scalable and distributed storage and computations
3. Distributed data blocks and tasks, and processing at the clusters
4. Mapping of the data at the physical nodes
5. Reducing the complexity of multiple data sources, and sending the computational results to the applications
6. Developing, deploying, operating and managing in Big Data environment of an enterprise
7. Integration of solutions into a cohesive solution
8. Uses of large resources of MPPs, cloud, specialized tools and parallel processing and use of high speed networks [Section 1.5.3].

Big Data store should also manage the variety of data formats. Hadoop is scalable and parallel computing platform to handle Big Data (Section 1.6.3.1, Figure 1.8). Hadoop distributed file system design is for storing and analytics of Big Data. HDFS packages Big Data in a distributed data store along with processing using a programming model.

This chapter focuses on Hadoop, its ecosystem, HDFS based programming model, MapReduce, Yarn, and introduces to ecosystem components, such as AVRO, Zookeeper, Ambari, HBase, Hive, Pig and Mahout.

2.1 | INTRODUCTION

A programming model is *centralized* computing of data in which the data is transferred from multiple distributed data sources to a central server. Analyzing, reporting, visualizing, business-intelligence tasks compute centrally. Data are inputs to the central server.

An enterprise collects and analyzes data at the enterprise level. The computations are at an enterprise server or data warehouse integrated with the applications (Figure 1.6). An example is computations using Oracle application integration architecture (Section 1.6.1.7). The computing nodes need to connect to a central system through high-speed networks.

Assume that a centralized server does the function of collection, storing and

analyzing. For example, at an ACVM Company enterprise server. The data at the server gets collected from a large number of ACVMs which the company locates in multiple cities, areas and locations. The server also receives data from social media (Example 1.6 (i)). Applications running at the server does the following analysis:

1. Suggests a strategy for filling the machines at minimum cost of logistics
2. Finds locations of high sales such as gardens, playgrounds etc.
3. Finds days or periods of high sales such as Xmas etc.
4. Finds children's preferences for specific chocolate flavors
5. Finds the potential region of future growth
6. Identifies unprofitable machines
7. Identifies need of replicating the number of machines at specific locations.

Another programming model is distributed computing that uses the databases at multiple computing nodes with data sharing between the nodes during computation. Distributed computing in this model requires the cooperation (sharing) between the DBs in a transparent manner. Transparent means that each user within the system may access all the data within all databases as if they were a single database. A second requirement is location independence. Analysis results should be independent of geographical locations. The access of one computing node to other nodes may fail due to a single link failure.

The following example shows why the simply scaling out and division of the computations on a large number of processors may not work well due to data sharing between distributed computing nodes.

EXAMPLE 2.1

Consider a jigsaw Puzzle Ravensburger Beneath the Sea (5000 pieces). Children above 14 years of age will assemble the pieces in order to solve the puzzle. What will be the effect on time intervals for solution in three situations, when 4, 100 and 200 children simultaneously attempt the solution.

SOLUTION

Let the time taken by a single child to solve the puzzle be T . Assume 4 children sit together and solve the puzzle by dividing the tasks. Each child assembles one-fourth part of the picture for which they pick the pieces from a common basket (Distributed computing and centralized data model).

Alternatively, each child assembles one-fourth part of the picture for which the pieces are distributed in four baskets. The child in case does not find a piece in his/her basket, then searches for it in another basket (Distributed databases and distributed computing tasks with data sharing model).

Partitioning of assembling jobs into four has an issue. A child may complete his/her part much later than the remaining children. Beneath-the-sea portion is too complex, while upper-depth-sea portion is just plain. The children combine all four parts and finally complete the puzzle. Each one has to look into the other three parts to find a match and complete the task. Time taken to solve the puzzle is $[T/4 + T_I(4) + T_C(4)]$, where $T_I(4)$ is the time taken in seeking from others the pieces not available to a child during intermediate phases, and $T_C(4)$ in combining the results of the four children. Scaling factor is slightly less than 4. The proposed distributed model works well.

Assume a second situation in which 100 children assemble their parts of 50 pieces each, and finally combine all 100 parts and complete the puzzle. Each child must seek a piece, not available with her/him during the intermediate phase. Combining also becomes difficult and a time-consuming exercise compared to the four children case because each child now matches the results with the remaining 99 counterparts to arrive at the final solution. The time taken to solve the puzzle is $[T/100 + T_I(100) + T_C(100)]$,

where $T_I(100)$ and $T_C(100)$ are the time taken in seeking pieces not available with the child and combining results of 100 children, respectively. Scaling is by factor less than 100. The distributed model has issues like sharing pieces, seeking pieces not available and combining issues. Issues are at the intermediate as well as at the end stages.

If 200 children attempt to solve the puzzle simultaneously at the same time then finally combining all 200 portions of the Beneath the Sea, the integration of 200 portions will be tedious and will be a far more time-consuming exercise than with 4 or 100. The time taken to solve the puzzle is $[T/200 + T_I(200)]$

+ $T_C(200)$], where $T_I(200)$ and $T_C(200)$ is the time taken in seeking the pieces not available and combining, respectively. Scaling up is by factor much less than 200 and may even be less than even 100. The distributed model with pieces sharing between the children is unsatisfactory because $T_I(200) + T_C(200) < T/200$.

Problem of inter-children interactions exponentially grows with the number of children in the proposed distributed model with seeking pieces in intermediate phases. Time T_I becomes significantly high.

Alternatively, the picture parts and corresponding pieces of each part distribute to each participating child distinctly (Distributed computing model with no data sharing). Time T_I taken in seeking a piece not available with him/her is zero. The time taken in joining the assembled picture portions is only at the end. Problem of inter-children interactions during solving the puzzle does not exist.

Traditionally, a program when executes calls the data inputs. Centralized computing model requires few communication overheads. Distributed computing model requires communication overheads for seeking data from a remote source when not available locally, and arrive at the final result. The completion of computations will take more and more time when the number of distributed computing nodes increase.

Distributed pieces of codes as well as the data at the computing nodes
Transparency between data nodes at computing nodes do not fulfil for Big Data when distributed computing takes place using data sharing between local and remote. Following are the reasons for this:

- Distributed data storage systems do not use the concept of joins.
- Data need to be fault-tolerant and data stores should take into account the possibilities of network failure. When data need to be partitioned

into data blocks and written at one set of nodes, then those blocks need replication at multiple nodes. This takes care of possibilities of network faults. When a network fault occurs, then replicated node makes the data available.

- Big Data follows a theorem known as the CAP theorem. The CAP states that out of three properties (consistency, availability and partitions), two must at least be present for applications, services and processes.

Recall Table 1.2. Consider distributed computing model which requires no sharing between data nodes. The model is equivalent to distribution of the picture's parts and corresponding pieces of each part to each participating child distinctly. Multiple tasks of an application also distribute, run using machines associated with multiple data nodes and execute at the same time in parallel.

The application tasks and datasets needed for computations distribute at a number of geographic locations and remote servers. The enterprise uses MPPs or computing clusters when datasets are too large. Application is divided in number of tasks and sub-tasks. The sub-tasks get inputs from data nodes at the same cluster. The results of sub-tasks aggregate and communicate to the application. The aggregate results from each cluster collect using APIs at the application.

(i) Big Data Store Model

A model for Big Data store is as follows:

Data store in file system consisting of data blocks (physical division of data). The data blocks are distributed across multiple nodes. Data nodes are at the racks of a cluster. Racks are scalable. A Rack has multiple data nodes (data servers), and each cluster is arranged in a number of racks.

Data Store model of files in data nodes in racks in the clusters Hadoop system uses the data store model in which storage is at clusters, racks, data nodes and data blocks. Data blocks replicate at the DataNodes such that a failure of link leads to access of the data block from the other nodes replicated at the same or other racks.

(ii) Big Data Programming Model

Big Data programming model is that application in which application jobs and tasks (or sub-tasks) is scheduled on the same servers which store the data for processing.

Big Data programming model: application jobs run on the same servers which store the data for processing, and parallel running of the jobs

Job means running an assignment of a set of instructions for processing. For example, processing the queries in an application and sending the result back to the application *is a job*. Other example is instructions for sorting the examination performance data is a job.

Job scheduling means assigning a job for processing following a schedule. For example, scheduling after a processing unit finishes the previously assigned job, scheduling as per specific sequence or after a specific period.

Hadoop system uses the programming model, where jobs or tasks are assigned and scheduled on the same servers which hold the data. Hadoop is one of the widely used technologies. Google and Yahoo use Hadoop. Hadoop creators created a cost-effective method to build search indexes. Facebook, Twitter and LinkedIn use Hadoop. IBM implemented BigInsights and uses licensed Apache Hadoop. Oracle implements Hadoop system with Big Data Appliance, IBM with Infosphere and Microsoft with Big Data solutions.

Following are important key terms and their meaning.

Cluster Computing refers to computing, storing and analyzing huge amounts of unstructured or structured data in a distributed computing environment. Each cluster forms by a set of loosely or tightly connected computing nodes that work together and many of the operations can be timed (scheduled) and can be realized as if from a single computing system. Clusters improve the performance, provide cost-effective and improved node accessibility compared to a single computing node. Each node of the computational cluster is set to perform the same task and sub-tasks, such as MapReduce, which software control and schedule.

Data Flow (DF) refers to flow of data from one node to another. For example, transfer of output data after processing to input of application.

Data Consistency means all copies of data blocks have the same values.

Data Availability means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, a copy in the other

partition is available. Partition means parts, which are active but may not cooperate as in distributed databases (DBs).

Resources means computing system resources, i.e., the physical or virtual components or devices, made available for specified or scheduled periods within the system. Resources refer to sources such as files, network connections and memory blocks.

Resource management refers to managing resources such as their creation, deletion and controlled usages. The manager functions also includes managing the (i) availability for specified or scheduled periods, (ii) prevention of resource unavailability after a task finishes and (iii) resources allocation when multiple tasks attempt to use the same set of resources.

Horizontal scalability means increasing the number of systems working in coherence. For example, using MPPs or number of servers as per the size of the dataset. Processing different datasets of a large data store *running similar application* deploys the horizontal scalability.

Vertical scalability means scaling up using the giving system resources and increasing the number of tasks in the system. For example, extending analytics processing by including the reporting, business processing (BP), business intelligence (BI), data visualization, knowledge discovery and machine learning (ML) capabilities which require additional ways to solve problems of greater complexities and greater processing, storage and inter-process communication among the resources. Processing different datasets of a large data store *running multiple application* tasks deploys vertical scalability.

Ecosystem refers to a system made up of multiple computing components, which work together. That is similar to a biological ecosystem, a complex system of living organisms, their physical environment and all their inter-relationships in a particular unit of space.

Distributed File System means a system of storing files. Files can be for the set of data records, key-value pairs, hash key-value pairs, relational database or NoSQL database at the distributed computing nodes, accessible after referring to their resource-pointer using a master directory service, look-up tables or name-node server.

Hadoop Distributed File System means a system of storing files (set of data

records, key-value pairs, hash key-value pairs or applications data) at distributed computing nodes according to Hadoop architecture and accessibility of data blocks after finding reference to their racks and cluster. NameNode servers enable referencing to data blocks.

Scalability of storage and processing means the execution using varying number of servers according to the requirements, i.e., bigger data store on greater number of servers when required and on smaller data when smaller data used on limited number of servers. Big Data Analytics require deploying the *clusters using the servers or cloud* for computing as per the requirements.

Utility Cloud-based Services mean infrastructure, software and computing platform services similar to utility services, such as electricity, gas, water etc. Infrastructure refers to units for data-store, processing and network. The IaaS, SaaS and PaaS are the services at the cloud (Section 1.3.3).

This chapter describes on Hadoop's core components, such as MapReduce, HDFS and YARN, and Hadoop ecosystem components, such as HBase, Hive, Pig, and Mahout. Section 2.2 describes Hadoop, its ecosystem components, streaming and pipe functions. Section 2.3 describes Hadoop physical architecture, Hadoop distributed file system (HDFS) basics. The section describes how to organize the nodes for computations using large-scale file system. Section 2.4 gives a conceptual understanding of MapReduce Daemon and functioning of Hadoop MapReduce framework.

Section 2.5 describes Hadoop YARN for managing of resources along with application tasks. Section 2.6 describes the Hadoop ecosystem interactions, analytics application support with AVRO, Zookeeper, Ambari, HBase, Hive, Pig and Mahout.

2.2 | HADOOP AND ITS ECOSYSTEM

Apache initiated the project for developing storage and processing framework for Big Data storage and processing. Doug Cutting and Machael J. Cafarelle the creators named that framework as Hadoop. Cutting's son was fascinated by a stuffed toy elephant, named Hadoop, and this is how the name Hadoop was

LO 2.1

Hadoop core, components of Hadoop ecosystem, streaming and pipe interfaces for inputs to MapReduce

derived.

The project consisted of two components, one of them is for data store in blocks in the clusters and the other is computations at each individual cluster in parallel with another.

Hadoop components are written in Java with part of native code in C. The command line utilities are written in shell scripts.

Hadoop is a computing environment in which input data stores, processes and stores the results. The environment consists of clusters which distribute at the cloud or set of servers. Each cluster consists of a string of data files constituting data blocks. The toy named Hadoop consisted of a stuffed elephant. The Hadoop system cluster stuffs files in data blocks. The complete system consists of a scalable distributed set of clusters.

Infrastructure consists of cloud for clusters. A cluster consists of sets of computers or PCs. The Hadoop platform provides a low cost Big Data platform, which is open source and uses cloud services. Tera Bytes of data processing takes just few minutes. Hadoop enables distributed processing of large datasets (above 10 million bytes) across clusters of computers using a programming model called MapReduce. The system characteristics are scalable, self-manageable, self-healing and distributed file system.

Scalable means can be scaled up (enhanced) by adding storage and processing units as per the requirements. Self-manageable means creation of storage and processing resources which are used, scheduled and reduced or increased with the help of the system itself. Self-healing means that in case of faults, they are taken care of by the system itself. Self-healing enables functioning and resources availability. Software detect and handle failures at the task level. Software enable the service or task execution even in case of communication or node failure.

The hardware scales up from a single server to thousands of machines that store the clusters. Each cluster stores a large number of data blocks in racks. Default data block size is 64 MB. IBM BigInsights, built on Hadoop deploys default 128 MB block size. Hadoop framework provides the computing features of a system of distributed, flexible, scalable, fault tolerant computing with high computing power. Hadoop system is an efficient platform for the distributed storage and processing of a large amount of data.

Hadoop enables Big Data storage and cluster computing. The Hadoop system manages both, large-sized structured and unstructured data in different formats, such as XML, JSON and text with efficiency and effectiveness. The Hadoop system performs better with clusters of many servers when the focus is on horizontal scalability. The system provides faster results from Big Data and from unstructured data as well.

Yahoo has more than 100000 CPUs in over 40000 servers running Hadoop, with its biggest Hadoop cluster running 4500 nodes as of March 2017, according to the Apache Hadoop website. Facebook has 2 major clusters: a cluster has 1100-machines with 8800 cores and about 12 PB raw storage. A 300-machine cluster with 2400 cores and about 3 PB ($1 \text{ PB} = 10^{15} \text{ B}$, nearly 2^{50} B) raw-storage. Each (commodity) node has 8 cores and 12 TB ($1 \text{ TB} = 10^{12}$, nearly $2^{40} \text{ B} = 1024 \text{ GB}$) of storage.

2.2.1 Hadoop Core Components

Figure 2.1 shows the core components of the Apache Software Foundation's Hadoop framework.

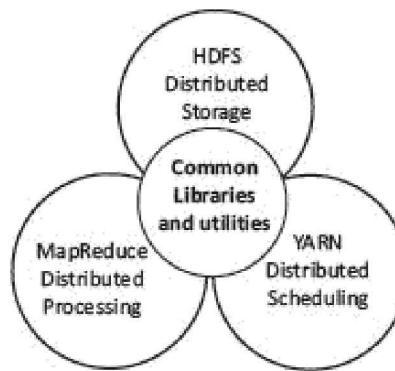


Figure 2.1 Core components of Hadoop

The Hadoop core components of the framework are:

1. Hadoop Common – The common module contains the libraries and utilities that are required by the other modules of Hadoop. For example, Hadoop common provides various components and interfaces for distributed file system and general input/output. This includes serialization, Java RPC (Remote Procedure Call) and file-based data

structures.

2. Hadoop Distributed File System (HDFS) – A Java-based distributed file system which can store all kinds of data on the disks at the clusters.
3. MapReduce v1 – Software programming model in Hadoop 1 using Mapper and Reducer. The v1 processes large sets of data in parallel and in batches.
4. YARN – Software for managing resources for computing. The user application tasks or sub-tasks run in parallel at the Hadoop, uses scheduling and handles the requests for the resources in distributed running of the tasks.
5. MapReduce v2 – Hadoop 2 YARN-based system for parallel processing of large datasets and distributed processing of the application tasks.

2.2.1.1 Spark

Spark is an open-source cluster-computing framework of Apache Software Foundation. Hadoop deploys data at the disks. Spark provisions for in-memory analytics. Therefore, it also enables OLAP and real-time processing. Spark does faster processing of Big Data.

Spark has been adopted by large organizations, such as Amazon, eBay and Yahoo. Several organizations run Spark on clusters with thousands of nodes.

Spark is now increasingly becoming popular. Chapters 5 to 9 will describe Spark and its components in detail.

2.2.2 Features of Hadoop

Hadoop features are as follows:

1. *Fault-efficient scalable, flexible and modular design* which uses simple and modular programming model. The system provides servers at high scalability. The system is scalable by adding new nodes to handle larger data. Hadoop proves very helpful in storing, managing, processing and analyzing Big Data. Modular functions make the system flexible. One can add or replace components at ease. Modularity allows replacing its components for a different software tool.

2. *Robust design of HDFS*: Execution of Big Data applications continue even when an individual server or cluster fails. This is because of Hadoop provisions for backup (due to replications at least three times for each data block) and a data recovery mechanism. HDFS thus has high reliability.
3. *Store and process Big Data*: Processes Big Data of 3V characteristics.
4. *Distributed clusters computing model with data locality*: Processes Big Data at high speed as the application tasks and sub-tasks submit to the DataNodes. One can achieve more computing power by increasing the number of computing nodes. The processing splits across multiple DataNodes (servers), and thus fast processing and aggregated results.
5. *Hardware fault-tolerant*: A fault does not affect data and application processing. If a node goes down, the other nodes take care of the residue. This is due to multiple copies of all data blocks which replicate automatically. Default is three copies of data blocks.
6. *Open-source framework*: Open source access and cloud services enable large data store. Hadoop uses a cluster of multiple inexpensive servers or the cloud.
7. *Java and Linux based*: Hadoop uses Java interfaces. Hadoop base is Linux but has its own set of shell commands support.

Hadoop provides various components and interfaces for distributed file system and general input/output. This includes serialization, Java RPC (Remote Procedure Call) and file-based data structures in Java.

HDFS is basically designed more for batch processing. Streaming uses standard input and output to communicate with the Mapper and Reduce codes. Stream analytics and real-time processing poses difficulties when streams have high throughput of data. The data access is required faster than the latency at DataNode at HDFS.

YARN provides a platform for many different modes of data processing, from traditional batch processing to processing of the applications such as interactive queries, text analytics and streaming analysis.

These different types of data can be moved in HDFS for analysis using interactive query processing tools of Hadoop ecosystem components, such as Hive or can be provided to online business processes with the help of Apache HBase.

2.2.3 Hadoop Ecosystem Components

Hadoop ecosystem refers to a combination of technologies. Hadoop ecosystem consists of own family of applications which tie up together with the Hadoop. The system components support the storage, processing, access, analysis, governance, security and operations for Big Data.

Hadoop ecosystem: AVRO, ZooKeeper, Pig, Hive, Sqoop, Ambari, Mahout, Spark, Flink and Flume

The system enables the applications which run Big Data and deploy HDFS. The data store system consists of clusters, racks, DataNodes and blocks. Hadoop deploys application programming model, such as MapReduce and HBase. YARN manages resources and schedules sub-tasks of the application.

HBase uses columnar databases and does OLAP. Figure 2.2 shows Hadoop core components HDFS, MapReduce and YARN along with the ecosystem. Figure 2.2 also shows Hadoop ecosystem. The system includes the application support layer and application layer components- AVRO, ZooKeeper, Pig, Hive, Sqoop, Ambari, Chukwa, Mahout, Spark, Flink and Flume. The figure also shows the components and their usages.

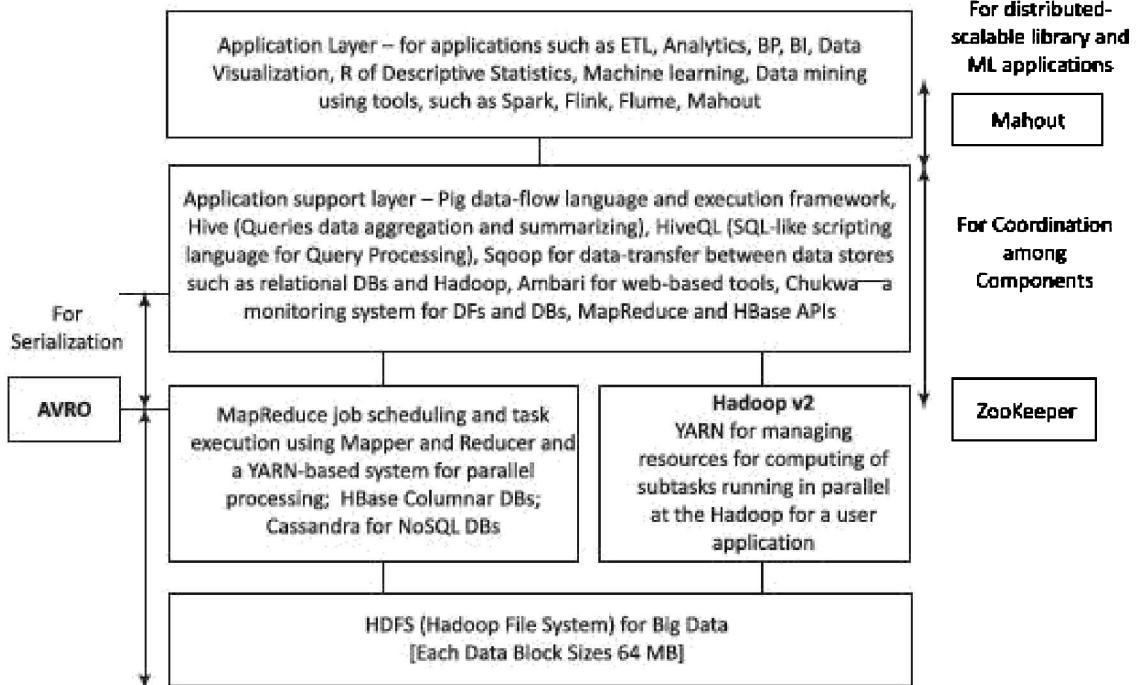


Figure 2.2 Hadoop main components and ecosystem components

The four layers in Figure 2.2 are as follows:

- Distributed storage layer
- Resource-manager layer for job or application sub-tasks scheduling and execution
- Processing-framework layer, consisting of Mapper and Reducer for the MapReduce process-flow
- APIs at application support layer (applications such as Hive and Pig). The codes communicate and run using MapReduce or YARN at processing framework layer. Reducer output communicate to APIs (Figure 2.2).

AVRO enables data serialization between the layers. Zookeeper enables coordination among layer components.

The holistic view of Hadoop architecture provides an idea of implementation of Hadoop components of the ecosystem. Client hosts run applications using Hadoop ecosystem projects, such as Pig, Hive and Mahout.

Most commonly, Hadoop uses Java programming. Such Hadoop programs run on any platform with the Java virtual-machine deployment model. HDFS is a Java-based distributed file system that can store various kinds of data on the computers.

2.2.4 Hadoop Streaming

HDFS with MapReduce and YARN-based system enables parallel processing of large datasets. Spark provides in-memory processing of data, thus improving the processing speed. Spark and Flink technologies enable in-stream processing. The two lead stream processing systems and are more useful for processing a large volume of data. Spark includes security features. Flink is emerging as a powerful tool. Flink improves the overall performance as it provides single run-time for streaming as well as batch processing. Simple and flexible architecture of Flink is suitable for streaming data.

2.2.5 Hadoop Pipes

Hadoop Pipes are the C++ Pipes which interface with MapReduce. Java native interfaces are not used in pipes. Apache Hadoop provides an adapter layer, which processes in pipes. A pipe means data streaming into the system at Mapper input and aggregated results flowing out at outputs. The adapter layer enables running of application tasks in C++ coded MapReduce programs. Applications which require faster numerical computations can achieve higher throughput using C++ when used through the pipes, as compared to Java.

Pipes do not use the standard I/O when communicating with Mapper and Reducer codes. Cloudera distribution including Hadoop (CDH) version CDH 5.0.2 runs the pipes. Distribution means software downloadable from the website distributing the codes. IBM PowerLinux systems enable working with Hadoop pipes and system libraries.

Self-Assessment Exercise linked to LO 2.1

1. How are core Hadoop components used for Big Data analytics?
2. Explain the meaning of distributed computing model with data locality?

3. Why are the Hadoop system and ecosystem components shown in the four layers? Explain by an example.
5. Differentiate between MapReduce v1 and MapReduce v2.

2.3 | HADOOP DISTRIBUTED FILE SYSTEM

LO 2.2

Big Data analytics applications are software applications that leverage large-scale data. The applications analyze Big Data using massive parallel processing frameworks. HDFS is a core component of Hadoop. HDFS is designed to run on a cluster of computers and servers at cloud-based utility services.

Hadoop Distributed File System (HDFS), and physical-organization of nodes for computing at clusters of large-scale files

HDFS stores Big Data which may range from GBs ($1 \text{ GB} = 2^{30} \text{ B}$) to PBs ($1 \text{ PB} = 10^{15} \text{ B}$,

nearly the 2^{50} B). HDFS stores the data in a distributed manner in order to compute fast. The distributed data store in HDFS stores data in any format regardless of schema. HDFS provides high throughput access to data-centric applications that require large-scale data processing workloads.

2.3.1 HDFS Data Storage

Hadoop data store concept implies storing the data at a number of *clusters*. Each cluster has a number of data stores, called *racks*. Each rack stores a number of *DataNodes*. Each *DataNode* has a large number of *data blocks*. The racks distribute across a cluster. The nodes have processing and storage capabilities. The nodes have the data in data blocks to run the application tasks. The data blocks *replicate by default at least on three DataNodes* in same or remote nodes. Data at the stores enable running the distributed applications including analytics, data mining, OLAP using the clusters. A *file, containing the data divides into data blocks*. A *data block default size is 64 MBs* (HDFS division of files concept is similar to Linux or virtual memory page in Intel x86 and Pentium processors where the block size is fixed and is of 4 KB).

Hadoop HDFS features are as follows:

- (i) *Create, append, delete, rename* and *attribute modification* functions
- (ii) Content of individual file cannot be modified or replaced but appended with new data at the end of the file
- (iii) Write once but read many times during usages and processing
- (iv) Average file size can be more than 500 MB.

The following is an example how the files store at a Hadoop cluster.

EXAMPLE 2.2

Consider a data storage for University students. Each student data, *stuData* which is in a file of size less than 64 MB ($1 \text{ MB} = 2^{20}\text{B}$). A data block stores the full file data for a student of *stuData_idN*, where $N = 1 \text{ to } 500$.

- (i) How the files of each student will be distributed at a Hadoop cluster? How many student data can be stored at one cluster? Assume that each rack has two DataNodes for processing each of 64 GB ($1 \text{ GB} = 2^{30}\text{B}$) memory. Assume that cluster consists of 120 racks, and thus 240 DataNodes.
- (ii) What is the total memory capacity of the cluster in TB ($(1 \text{ TB} = 2^{40}\text{B})$ and DataNodes in each rack?
- (iii) Show the distributed blocks for students with ID= 96 and 1025. Assume default replication in the DataNodes = 3.
- (iv) What shall be the changes when a *stuData* file size $\leq 128 \text{ MB}$?

SOLUTION

- (i) Data block default size is 64 MB. Each students file size is less than 64MB. Therefore, for each student file one data block suffices. A data block is in a DataNode. Assume, for simplicity, each rack has two nodes each of memory capacity = 64 GB. Each node can thus store $64 \text{ GB}/64\text{MB} = 1024$ data blocks = 1024 student files. Each rack can thus store $2 \times 64 \text{ GB}/64\text{MB} = 2048$ data blocks = 2048 student files. Each

data block default replicates three times in the DataNodes. Therefore, the number of students whose data can be stored in the cluster = number of racks multiplied by number of files divided by 3 = $120 \times 2048/3 = 81920$. Therefore, the maximum number of 81920 stuData_IDN files can be distributed per cluster, with N = 1 to 81920.

- (ii) Total memory capacity of the cluster = $120 \times 128 \text{ MB} = 15360 \text{ GB} = 15 \text{ TB}$. Total memory capacity of each DataNode in each rack = $1024 \times 64 \text{ MB} = 64 \text{ GB}$.
- (iii) Figure 2.3 shows a Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025. Each stuData file stores at two data blocks, of capacity 64 MB each.
- (iv) Changes will be that each node will have half the number of data blocks.

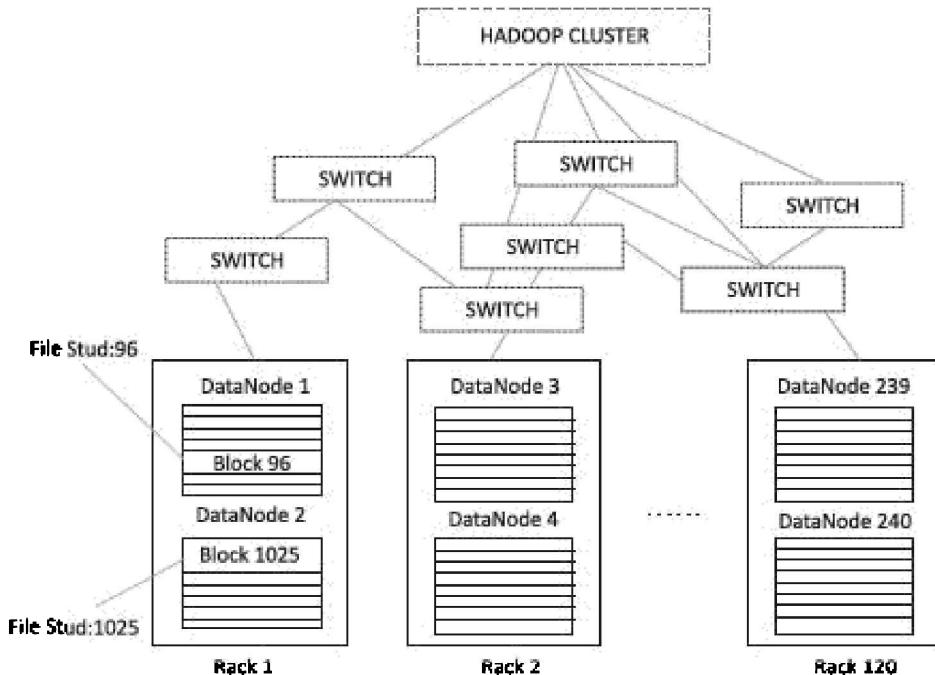


Figure 2.3 A Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025

2.3.1.1 Hadoop Physical Organization

The conventional file system uses directories. A directory consists of folders. A folder consists of files. When data processes, the data sources identify by pointers for the resources. A data-dictionary stores the resource pointers. Master tables at the dictionary store at a central location. (Section 1.5.1 for the details). The centrally stored tables enable administration easier when the data sources change during processing.

Identification of data-blocks, DataNodes and Racks using MasterNodes and NameNodes for processing the data at slave nodes

Similarly, the files, DataNodes and blocks need the identification during processing at HDFS. HDFS use the NameNodes and DataNodes. A NameNode stores the file's meta data. Meta data gives information about the file of user application, but does not participate in the computations. The DataNode stores the actual data files in the data blocks.

Few nodes in a Hadoop cluster act as NameNodes. These nodes are termed as MasterNodes or simply masters. The masters have a different configuration supporting high DRAM and processing power. The masters have much less local storage. Majority of the nodes in Hadoop cluster act as DataNodes and TaskTrackers. These nodes are referred to as slave nodes or slaves. The slaves have lots of disk storage and moderate amounts of processing capabilities and DRAM. Slaves are responsible to store the data and process the computation tasks submitted by the clients.

Figure 2.4 shows the client, master NameNode, primary and secondary MasterNodes and slave nodes in the Hadoop physical architecture.

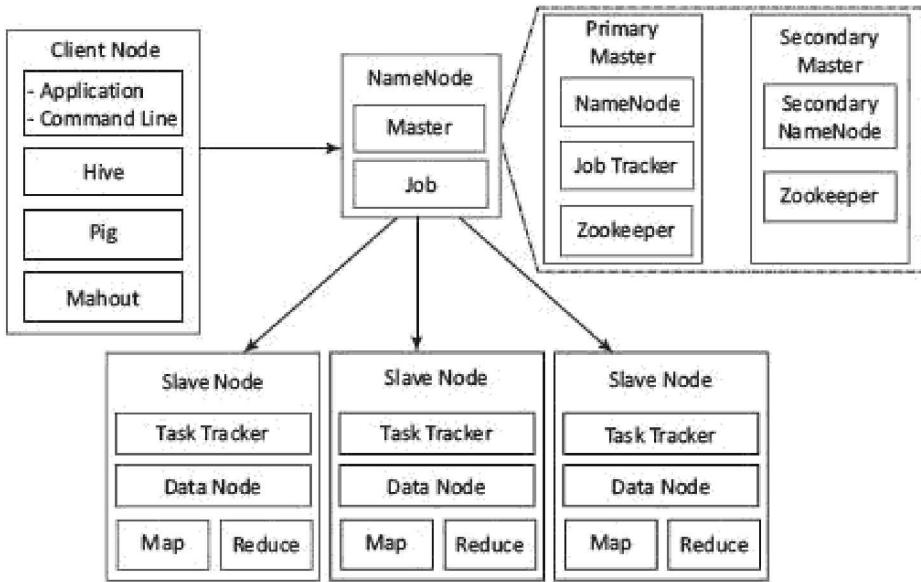


Figure 2.4 The client, master NameNode, MasterNodes and slave nodes

Clients as the users run the application with the help of Hadoop ecosystem projects. For example, Hive, Mahout and Pig are the ecosystem's projects. They are not required to be present at the Hadoop cluster. A single MasterNode provides HDFS, MapReduce and Hbase using threads in small to medium sized clusters. When the cluster size is large, multiple servers are used, such as to balance the load. The secondary NameNode provides NameNode management services and Zookeeper is used by HBase for metadata storage.

The MasterNode fundamentally plays the role of a coordinator. The MasterNode receives client connections, maintains the description of the global file system namespace, and the allocation of file blocks. It also monitors the state of the system in order to detect any failure. The Masters consists of three components NameNode, Secondary NameNode and JobTracker. The NameNode stores all the file system related information such as:

- The file section is stored in which part of the cluster
- Last access time for the files
- User permissions like which user has access to the file.

Secondary NameNode is an alternate for NameNode. Secondary node keeps a

copy of NameNode meta data. Thus, stored meta data can be rebuilt easily, in case of NameNode failure. The JobTracker coordinates the parallel processing of data.

Masters and slaves, and Hadoop client (node) load the data into cluster, submit the processing job and then retrieve the data to see the response after the job completion.

2.3.1.2 Hadoop 2

Single NameNode failure in Hadoop 1 is an operational limitation. Scaling up was also restricted to scale beyond a few thousands of DataNodes and few number of clusters. Hadoop 2 provides the multiple NameNodes. This enables higher resource availability. Each MN has the following components:

- An associated NameNode
- Zookeeper coordination client (an associated NameNode), functions as a centralized repository for distributed applications. Zookeeper uses synchronization, serialization and coordination activities. It enables functioning of a distributed system as a single function.
- Associated JournalNode (JN). The JN keeps the records of the state, resources assigned, and intermediate results or execution of application tasks. Distributed applications can write and read data from a JN.

The system takes care of failure issues as follows:

One set of resources is in active state. The other one remains in standby state. Two masters, one MN1 is in active state and other MN2 is in secondary state. That ensures the availability in case of network fault of an active NameNode NM1. The Hadoop system then activates the secondary NameNode NM2 and creates a secondary in another MasterNode MN3 unused earlier. The entries copy from JN1 in MN1 into the JN2, which is at newly active MasterNode MN2. Therefore, the application runs uninterrupted and resources are available uninterrupted.

2.3.2 HDFS Commands

Figure 2.1 showed Hadoop common module, which contains the libraries and utilities. They are common to other modules of Hadoop. The HDFS shell is not

compliant with the POSIX. Thus, the shell cannot interact similar to Unix or Linux. Commands for interacting with the files in HDFS require /bin/hdfs dfs <args>, where args stands for the command arguments. Full set of the Hadoop shell commands can be found at Apache Software Foundation website. -copyToLocal is the command for copying a file at HDFS to the local. -cat is command for copying to standard output (stdout). All Hadoop commands are invoked by the bin/Hadoop script. % Hadoop fsck / -files -blocks Table 2.1 gives the examples of command usages.

Table 2.1 Examples of usages of commands

HDFS shell command	Example of usage
-mkdir	Assume stu_filesdir is a directory of student files in Example 2.2. Then command for creating the directory is \$Hadoop hdfs-mkdir/user/stu_filesdir creates the directory named stu_files_dir
-put	Assume file stuData_id96 to be copied at stu_filesdir directory in Example 2.2. Then \$Hadoop hdfs-put stuData_id96 /user/ stu_filesdir copies file for student of id96 into stu_filesdir directory
-ls	Assume all files to be listed. Then \$hdfs hdfs dfs-ls command does provide the listing.
-cp	Assume stuData_id96 to be copied from stu_filesdir to new students' directory newstu_filesDir. Then \$Hadoop hdfs-cp stuData_id96 /user/stu_filesdir newstu_filesDir copies file for student of ID 96 into stu_filesdir directory

Self-Assessment Exercise linked to LO 2.2

1. (i) What does the create, append, delete, rename and attribute modification methods mean in the HDFS? (ii) Why is the content of an individual file not modified or replaced but appended at the end of the file? (iii) Why is the *write once but read many times* concept used in HDFS?
2. What are the functions of NameNode, DataNode, slave node and

MasterNode?

3. What are the benefits of multiple MasterNodes?
4. What are the usages of meta data?
5. Make a data-store model using HDFS for SGPs, SGPAAs and CGPAs of each student. Assume 50 UG and 10 PG courses offered at the university. Total intake capacity is 5000 each year. Each student information can extend up to 64 MB. How will the files of 5000 students be stored using HDFS? What shall be the minimum memory requirements in 20 years? (SGP means subject grade point awarded to a student, SGPA semester grade point average, and CGPA cumulative grade-point average.)

2.4 | MAPREDUCE FRAMEWORK AND PROGRAMMING MODEL

Figure 2.4 showed MapReduce functions as integral part of the Hadoop physical organization. MapReduce is a programming model for distributed computing.

Mapper means software for doing the assigned task after organizing the data blocks imported using the keys. A key specifies in a command line of Mapper. The command maps the key to the data, which an application uses.

Reducer means software for reducing the mapped data by using the aggregation, query or user-specified function. The reducer provides a concise cohesive response for the application.

Aggregation function means the function that groups the values of multiple rows together to result a single value of more significant meaning or measurement. For example, function such as count, sum, maximum, minimum, deviation and standard deviation.

Querying function means a function that finds the desired values. For example, function for finding a best student of a class who has shown the best performance in examination.

MapReduce allows writing applications to process reliably the huge amounts

LO 2.3

MapReduce Daemon
Framework and
MapReduce Programming
model for parallel
processing very large data

of data, in parallel, on large clusters of servers. The cluster size does not limit as such to process in parallel. The parallel programs of MapReduce are useful for performing large scale data analysis using multiple machines in the cluster.

Features of MapReduce framework are as follows:

1. Provides automatic parallelization and distribution of computation based on several processors
2. Processes data stored on distributed clusters of DataNodes and racks
3. Allows processing large amount of data in parallel
4. Provides scalability for usages of large number of servers
5. Provides MapReduce batch-oriented programming model in Hadoop version 1
6. Provides additional processing modes in Hadoop 2 YARN-based system and enables required parallel processing. For example, for queries, graph databases, streaming data, messages, real-time OLAP and ad hoc analytics with Big Data 3V characteristics.

The following subsection describes Hadoop execution model using MapReduce Framework.

2.4.1 Hadoop MapReduce Framework

MapReduce provides two important functions. The distribution of job based on client application task or users query to various nodes within a cluster is one function. The second function is organizing and reducing the results from each node into a cohesive response to the application or answer to the query.

The processing tasks are submitted to the Hadoop. The Hadoop framework in turns manages the task of issuing jobs, job completion, and copying data around the cluster between the DataNodes with the help of JobTracker (Figure 2.4).

Daemon refers to a highly dedicated program that runs in the background in a system. The user does not control or interact with that. An example is MapReduce in Hadoop system [Collins English language dictionary gives one of Daemon meaning as ‘a person who concentrates very hard or is very skilled at an activity and puts in lot of energy into it’].

MapReduce runs as per assigned Job by JobTracker, which keeps track of the job submitted for execution and runs TaskTracker for tracking the tasks. MapReduce programming enables job scheduling and task execution as follows:

A client node submits a request of an application to the JobTracker. A JobTracker is a Hadoop daemon (background program). The following are the steps on the request to MapReduce: (i) estimate the need of resources for processing that request, (ii) analyze the states of the slave nodes, (iii) place the mapping tasks in queue, (iv) monitor the progress of task, and on the failure, restart the task on slots of time available. The job execution is controlled by two types of processes in MapReduce:

1. The Mapper deploys map tasks on the slots. Map tasks assign to those nodes where the data for the application is stored. The Reducer output transfers to the client node after the data serialization using AVRO.
2. The Hadoop system sends the Map and Reduce jobs to the appropriate servers in the cluster. The Hadoop framework in turns manages the task of issuing jobs, job completion and copying data around the cluster between the slave nodes. Finally, the cluster collects and reduces the data to obtain the result and sends it back to the Hadoop server after completion of the given tasks.

The job execution is controlled by two types of processes in MapReduce. A single master process called JobTracker is one. This process coordinates all jobs running on the cluster and assigns map and reduce tasks to run on the TaskTrackers. The second is a number of subordinate processes called TaskTrackers. These processes run assigned tasks and periodically report the progress to the JobTracker.

Figure 2.4 showed the job execution model of MapReduce. Here the JobTracker schedules jobs submitted by clients, keeps track of TaskTrackers and maintains the available Map and Reduce slots. The JobTracker also monitors the execution of jobs and tasks on the cluster. The TaskTracker executes the Map and Reduce tasks, and reports to the JobTracker.

2.4.2 MapReduce Programming Model

MapReduce program can be written in any language including JAVA, C++ PIPEs

or Python. Map function of MapReduce program do mapping to compute the data and convert the data into other data sets (distributed in HDFS). After the Mapper computations finish, the Reducer function collects the result of map and generates the final output result. MapReduce program can be applied to any type of data, i.e., structured or unstructured stored in HDFS.

The input data is in the form of file or directory and is stored in the HDFS. The MapReduce program performs two jobs on this input data, the Map job and the Reduce job. They are also termed as two phases— Map phase and Reduce phase. The map job takes a set of data and converts it into another set of data. The individual elements are broken down into tuples (key/value pairs) in the resultant set of data. The reduce job takes the output from a map as input and combines the data tuples into a smaller set of tuples. Map and reduce jobs run in isolation from one another. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

The MapReduce v2 uses YARN based resource scheduling which simplifies the software development. Here, the jobs can be split across almost any number of servers. For example, the ACVM Company can find the number of chocolates KitKat, Milk, Fruit and Nuts, Nougat and Oreo sold every hour at the number of ACVMs installed all over in the multiple cities on separate servers [Refer Example 1.6(i)]. A server maps the keys for KitKat and another for Oreo. It requires time to scan the hourly sales log sequentially. By contrast, MapReduce programmer can split the application task among multiple sub-tasks, say one hundred sub-tasks, where each sub-task processes the data of the selected set of ACVMs. The results of all the sub-tasks then aggregate to get the final result, hourly sales figures of each chocolate flavor from all ACVMs of the company. Finally, the aggregated hourly results appear from the hourly log of transactions filed at Hadoop DataNodes. The company enterprise server runs analytics and applications consider the results as if from a single server application. The following example shows the usage of HDFS and the map and reduce functions.

MapReduce and YARN based resource scheduling splits the jobs into subtasks which run across number of servers in parallel.

EXAMPLE 2.3

Consider Example 1.6(i) of ACVMs selling KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates. Assume 24 files are created every hour for each

day. The files are at file_1, file_2, ..., file_24. Each file stores as key-value pairs as hourly sales log at the large number of machines.

- (i) How will the large number of machines, say 5000 ACVMs hourly data for each flavor sales log store using HDFS? What will be the strategy to restrict the data size in HDFS?
- (ii) How will the sample of data collected in a file for 0-1,1-2, ... 12-13,13-14, 15-16, up to 23-24 specific hour-sales log for sales at a large number of machines, say 5000?
- (iii) What will be the output streams of map tasks for feeding the input streams to the Reducer?
- (iv) What will be the Reducer outputs?

SOLUTION

5000 machines send sales data every hour for KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates, i.e., a total of 5 flavors. Assume each sales data size = 64 B, then data bytes $64 \times 5 \times 5000 \text{ B} = 1600000 \text{ B}$ will accumulate (append) each hour in a file.

Sales data are date-time stamped key-value pairs. Each of 24 hour hourly log files will use initially 24 data blocks at a DataNode and replicated at three DataNodes. A data file in one year will accumulate $1600000 \times 24 \times 365 \text{ B} = 14016000000 \text{ B} = \text{nearly 16 GB}$. Each data block can store 64 MB. Therefore, $16 \text{ GB}/64 \text{ MB} = 250$ data blocks in each file each year.

However, hourly and daily sales analytics is required only for managing supply chain for chocolate fill service and finding insight into sales during holidays and festival days compared to other days. Therefore, a strategy can be designed to replace the hourly sales data each month and create new files for monthly sales data.

A file sample-data of key-value pairs for hour-sales log in file_16 for sales during 15:00-16:00 will be as follows:

ACVM_id10KitKat, 23

ACVM_id2206Milk, 31

ACVM_id20Oreo, 36

```
ACVM_id10FruitNuts, 18
```

```
ACVM_id16Nougat, 8
```

```
.
```

```
.
```

```
.
```

```
ACVM_id1224KitKat, 48
```

```
ACVM_id4837Nougat, 28
```

```
.
```

Map tasks will map the input streams of key values at files, file_1, file_2, file_23, file_24 every hour. The resulting 5000 key value pairs maps each hour with keys for ACVM_idNKitKats (N = 1 to 5000). The output stream from Mapper will be as follows:

```
(ACVM_id10KitKat, 0), (ACVM_id1224KitKat, 3), ..., ...  
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...
```

Hourly 5 output streams of mapped tasks for all chocolates of all 5000 machines will be input to the reduce task.

The Reducer processes each hour using 5 input streams, sums all machines sales and generates one output (ACVMs_KitKat, 109624), (ACVMs_Milk, 128324), (ACVMs_FruitNuts, 9835), (ACVMs_Nougat, 2074903), and (ACVMs_Oreo, 305163). The reduced output serializes and is input to the analytics applications each hour.

Chapter 4 describes MapReduce programming in detail.

Self-Assessment Exercise linked to LO 2.3

1. Why is mapping required when processing the stored data at HDFS?
2. How do Jobtracker and TaskTracker function?
3. How does MapReducer along with the YARN resources manager enable faster processing of an application?
4. Consider HDFS DataNodes in a cluster. Draw a diagram depicting 10

data nodes storing the data of 4 groups of students. Using the diagram, show the execution of MapReduce sub-tasks for each group in parallel on the DataNodes in a cluster.

2.5 | HADOOP YARN

YARN is a resource management platform. It manages computer resources. The platform is responsible for providing the computational resources, such as CPUs, memory, network I/O which are needed when an application executes. An application task has a number of sub-tasks. YARN manages the schedules for running of the sub-tasks. Each sub-task uses the resources in allotted time intervals.

YARN separates the resource management and processing components. YARN stands for Yet Another Resource Negotiator. An application consists of a number of tasks. Each task can consist of a number of sub-tasks (threads), which run in parallel at the nodes in the cluster. YARN enables running of multi-threaded applications. YARN manages and allocates the resources for the application sub-tasks and submits the resources for them at the Hadoop system.

2.5.1 Hadoop 2 Execution Model

Figure 2.5 shows the YARN-based execution model. The figure shows the YARN components—Client, Resource Manager (RM), Node Manager (NM), Application Master (AM) and Containers.

Figure 2.5 also illustrates YARN components namely, Client, Resource Manager (RM), Node Manager (RM), Application Master (AM) and Containers.

List of actions of YARN resource allocation and scheduling functions is as follows:

- A MasterNode has two components: (i) Job History Server and (ii) Resource Manager(RM).
- A Client Node submits the request of an application to the RM. The RM is the master. One RM exists per cluster. The RM keeps information of all the

LO 2.4

Hadoop YARN for management and scheduling of resources for parallel running of application tasks

slave NMs. Information is about the location (Rack Awareness) and the number of resources (data blocks and servers) they have. The RM also renders the Resource Scheduler service that decides how to assign the resources. It, therefore, performs resource management as well as scheduling.

- Multiple NMs are at a cluster. An NM creates an AM instance (AMI) and starts up. The AMI initializes itself and registers with the RM. Multiple AMIs can be created in an AM.
- The AMI performs role of an Application Manager (ApplM), that estimates the resources requirement for running an application program or sub-task. The ApplMs send their requests for the necessary resources to the RM. Each NM includes several containers for uses by the subtasks of the application.
- NM is a slave of the infrastructure. It signals whenever it initializes. All active NMs send the controlling signal periodically to the RM signaling their presence.

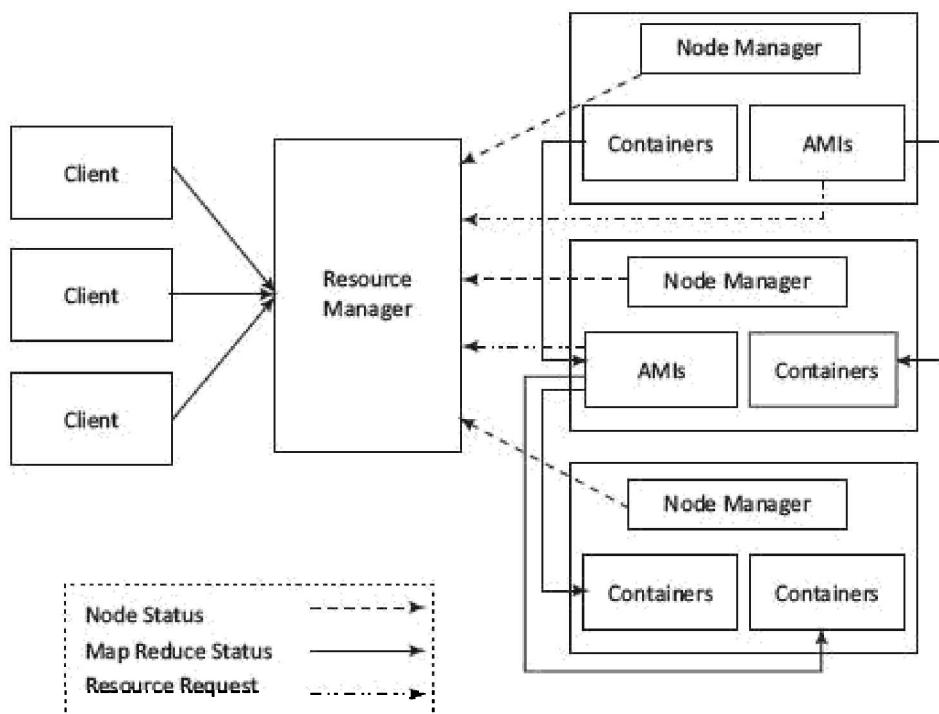


Figure 2.5 YARN-based execution model

- Each NM assigns a container(s) for each AMI. The container(s) assigned at an instance may be at same NM or another NM. ApplM uses just a fraction of the resources available. The ApplM at an instance uses the assigned container(s) for running the application sub-task.
- RM allots the resources to AM, and thus to ApplMs for using assigned containers on the same or other NM for running the application subtasks in parallel.

Self-Assessment Exercise linked to LO 2.4

1. What are the resources required for running an application? How they are allocated?
2. List the functions of YARN.
3. Explain using Example 2.3, how the Application Master coordinates the execution of all tasks submitted for an application and requests for appropriate resource containers to execute the task.
4. List the functions of Client, Resource Manager, Node Manager, Application Master and Containers

2.6 | HADOOP ECOSYSTEM TOOLS

A simple framework of Hadoop enabled development of a number of open-source projects has quickly emerged (Figure 2.2). They solve very specific problems related to distributed storage and processing model. Table 2.2 gives the functionalities of the ecosystem tools and components.

LO 2.4

Hadoop YARN for management and scheduling of resources for parallel running of application tasks

Table 2.2 Functionalities of the ecosystem tools and components

Ecosystem Tool	Functionalities
----------------	-----------------

ZooKeeper – Coordination service	Provisions high-performance coordination service for distributed running of applications and tasks (Sections 2.3.1.2 and 2.6.1.1)
Avro— Data serialization and transfer utility	Provisions data serialization during data transfer between application and processing layers (Figure 2.2 and Section 2.4.1)
Oozie	Provides a way to package and bundles multiple coordinator and workflow jobs and manage the lifecycle of those jobs (Section 2.6.1.2)
Sqoop (SQL-to-Hadoop) – A data-transfer software	Provisions for data-transfer between data stores such as relational DBs and Hadoop (Section 2.6.1.3)
Flume – Large data transfer utility	Provisions for reliable data transfer and provides for recovery in case of failure. Transfers large amount of data in applications, such as related to social-media messages (Section 2.6.1.4)
Ambari – A web-based tool	Provisions, monitors, manages, and viewing of functioning of the cluster, MapReduce, Hive and Pig APIs (Section 2.6.2)
Chukwa – A data collection system	Provisions and manages data collection system for large and distributed systems
HBase – A structured data store using database	Provisions a scalable and structured database for large tables (Section 2.6.3)
Cassandra – A database	Provisions scalable and fault-tolerant database for multiple masters (Section 3.7)
Hive – A data warehouse system	Provisions data aggregation, data-summarization, data warehouse infrastructure, ad hoc (unstructured) querying and SQL-like scripting language for query processing using HiveQL (Sections 2.6.4, 4.4 and 4.5)
Pig – A high-level dataflow	Provisions dataflow (DF) functionality and the execution framework for

language	parallel computations (Sections 2.6.5 and 4.6)
Mahout –A machine learning software	Provisions scalable machine learning and library functions for data mining and analytics (Sections 2.6.6 and 6.9)

The following subsections describe the Hadoop Ecosystem tools.

2.6.1 Hadoop Ecosystem

Consider ZooKeeper, Oozie, Sqoop and Flume.

2.6.1.1 Zookeeper

Designing of a distributed system requires designing and developing the coordination services. Apache Zookeeper is a coordination service that enables synchronization across a cluster in distributed applications (Figure 2.2). The coordination service manages the jobs in the cluster. Since multiple machines are involved, the race condition and deadlock are common problems when running a distributed application.

Zookeeper in Hadoop behaves as a centralized repository where distributed applications can write data at a node called JournalNode and read the data out of it. Zookeeper uses synchronization, serialization and coordination activities. It enables functioning of a distributed system as a single function.

ZooKeeper's main coordination services are:

- 1 Name service – A name service maps a name to the information associated with that name. For example, DNS service is a name service that maps a domain name to an IP address. Similarly, name keeps a track of servers or services those are up and running, and looks up their status by name in name service.
- 2 Concurrency control – Concurrent access to a shared resource may cause inconsistency of the resource. A concurrency control algorithm accesses shared resource in the distributed system and controls concurrency.
- 3 Configuration management – A requirement of a distributed system is a central configuration manager. A new joining node can pick up the up-to-

date centralized configuration from the ZooKeeper coordination service as soon as the node joins the system.

- 4 Failure – Distributed systems are susceptible to the problem of node failures. This requires implementing an automatic recovering strategy by selecting some alternate node for processing (Using two MasterNodes with a NameNode each).

2.6.1.2 Oozie

Apache Oozie is an open-source project of Apache that schedules Hadoop jobs. An efficient process for job handling is required. Analysis of Big Data requires creation of multiple jobs and sub-tasks in a process. Oozie design provisions the scalable processing of multiple jobs. Thus, Oozie provides a way to package and bundle multiple coordinator and workflow jobs, and manage the lifecycle of those jobs.

The two basic Oozie functions are:

- Oozie workflow jobs are represented as Directed Acrylic Graphs (DAGs), specifying a sequence of actions to execute.
- Oozie coordinator jobs are recurrent Oozie workflow jobs that are triggered by time and data availability.

Oozie provisions for the following:

1. Integrates multiple jobs in a sequential manner
2. Stores and supports Hadoop jobs for MapReduce, Hive, Pig, and Sqoop
3. Runs workflow jobs based on time and data triggers
4. Manages batch coordinator for the applications
5. Manages the timely execution of tens of elementary jobs lying in thousands of workflows in a Hadoop cluster.

2.6.1.3 Sqoop

The loading of data into Hadoop clusters becomes an important task during data analytics. Apache Sqoop is a tool that is built for loading efficiently the voluminous amount of data between Hadoop and external data repositories that resides on enterprise application servers or relational databases. Sqoop works

with relational databases such as Oracle, MySQL, PostgreSQL and DB2.

Sqoop provides the mechanism to import data from external Data Stores into HDFS. Sqoop relates to Hadoop eco-system components, such as Hive and HBase. Sqoop can extract data from Hadoop or other ecosystem components.

Sqoop provides command line interface to its users. Sqoop can also be accessed using Java APIs. The tool allows defining the schema of the data for import. Sqoop exploits MapReduce framework to import and export the data, and transfers for parallel processing of sub-tasks. Sqoop provisions for fault tolerance. Parallel transfer of data results in parallel results and fast data transfer.

Sqoop initially parses the arguments passed in the command line and prepares the map task. The map task initializes multiple Mappers depending on the number supplied by the user in the command line. Each map task will be assigned with part of data to be imported based on key defined in the command line. Sqoop distributes the input data equally among the Mappers. Then each Mapper creates a connection with the database using JDBC and fetches the part of data assigned by Sqoop and writes it into HDFS/Hive/HBase as per the choice provided in the command line.

2.6.1.4 Flume

Apache Flume provides a distributed, reliable and available service. Flume efficiently collects, aggregates and transfers a large amount of streaming data into HDFS. Flume enables upload of large files into Hadoop clusters.

The features of flume include robustness and fault tolerance. Flume provides data transfer which is reliable and provides for recovery in case of failure. Flume is useful for transferring a large amount of data in applications related to logs of network traffic, sensor data, geo-location data, e-mails and social-media messages.

Apache Flume has the following four important components:

1. **Sources** which accept data from a server or an application.
2. **Sinks** which receive data and store it in HDFS repository or transmit the data to another source. Data units that are transferred over a **channel** from source to sink are called events.

3. **Channels** connect between sources and sink by queuing event data for transactions. The size of events data is usually 4 KB. The data source is considered to be a source of various set of events. Sources listen for events and write events to a channel. Sinks basically write event data to a target and remove the event from the queue.
4. **Agents** run the sinks and sources in Flume. The interceptors drop the data or transfer data as it flows into the system.

2.6.2 Ambari

Apache Ambari is a management platform for Hadoop. It is open source. Ambari enables an enterprise to plan, securely install, manage and maintain the clusters in the Hadoop. Ambari provisions for advanced cluster security capabilities, such as Kerberos Ambari.

2.6.2.1 Features

Features of Ambari and associated components are as follows:

1. Simplification of installation, configuration and management
2. Enables easy, efficient, repeatable and automated creation of clusters
3. Manages and monitors scalable clustering
4. Provides an intuitive Web User Interface and REST API. The provision enables automation of cluster operations.
5. Visualizes the health of clusters and critical metrics for their operations
6. Enables detection of faulty node links
7. Provides extensibility and customizability.

2.6.2.2 Hadoop Administration

Hadoop large clusters pose a number of configuration and administration challenges. Administrator procedures enable managing and administering Hadoop clusters, resources and associated Hadoop ecosystem components (Figure 2.2). Administration includes installing and monitoring clusters.

Ambari also provides a centralized setup for security. This simplifies the administering complexities and configures security of clusters across the entire

platform. Ambari helps automation of the setup and configuration of Hadoop using Web User Interface and REST APIs.

IBM BigInsights provides an administration console. The console is similar to web UI at Ambari. The console enables visualization of the cluster health, HDFS directory structure, status of MapReduce tasks, review of log records and access application status. Single harmonized view on console makes administering the task easier. Visualization can be up to individual components level on drilling down. Nodes addition and deletion are easy using the console.

The console enables built-in tools for administering. Web console provides a link to server tools and open-source components associated with those.

2.6.3 HBase

Similar to database, HBase is an Hadoop system database. HBase was created for large tables. HBase is an open-source, distributed, versioned and non-relational (NoSQL) database. Features of HBase features are:

1. Uses a partial columnar data schema on top of Hadoop and HDFS.
2. Supports a large table of billions of rows and millions of columns.
3. Provides small amounts of information, called sparse data taken from large data sets which are storing empty or presently not-required data. For example, yearly sales data of KitKats from the data of hourly, daily and monthly sales (Example 2.3).
4. Supports data compression algorithms.
5. Provisions in-memory column-based data transactions.
6. Accesses rows serially and does not provision for random accesses and write into the rows.
7. Provides random, real-time read/write access to Big Data.
8. Fault tolerant storage due to automatic failure support between DataNodes servers.
9. Similarity with Google BigTable.

HBase is written in Java. It stores data in a large structured table. HBase

provides scalable distributed Big Data Store. HBase data store as key-value pairs.

HBase system consists of a set of tables. Each table contains rows and columns, similar to a traditional database. HBase provides a primary key as in the database table. Data accesses are performed using that key.

HBASE applies a partial columnar scheme on top of the Hadoop and HDFS. An HBase column represents an attribute of an object, such as hourly sales of KitKat, Milk, Fruit and Nuts, Nougat and Oreo sold every hour at an ACVM (Example 2.3).

The following example shows a structured table considering Examples 1.6 and 2.3.

EXAMPLE 2.4

Recapitulate Examples 1.6 and 2.3. Consider ACVMs selling KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates. Following is the table for hourly sales of chocolates at multiple ACVMs.

ACVM_ID	Date (DT) mmddyy	Hour (hr)	KitKat Hourly Sale (KKHS)	Milk Hourly Sale(MHS)	Fruit and Nuts Hourly Sale (FNHS)	Nougat Hourly Sale (NHS)	Oreo Hourly Sale (OHS)
2206	121217	16	28	23	38	8	50
10	121217	16	29	14	44	15	38
1224	121217	16	40	41	23	37	8
16	121217	46	13	25	7	8	74
28	121217	16	52	22	16	28	7
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

- How does the HBase store the table?
- How will the records created using shell command ‘put’?

SOLUTION

Format of the HBase that stores rows line by line is:

Row-Key Column-Family: {Column-Qualifier: Version:
Value}

HBase data model specifies the *column qualifiers*. For example, column qualifiers are DT, HR, KKHS, MHS, FNHS, NHS and OHS. Version corresponds to a number reflecting the time stamp which identifies the data of columns uniquely. Version is a number reflecting server time-stamp by default. Value is the *value* in the column field for the qualifier. The first row stores in the HBase as follows:

```
ACVM_id: '2206' { 'DT':1600080000024: '121217', 'HR':  
1600008007319: '16', 'KKHS': 1600081010821: '28',  
'MHS': 1600082010582: '23', 'FNHS': 1600082018001:  
'38', 'NHS': 1600080158868: '8', 'OHS':  
1600038028229: '50'}
```

Write similarly for other rows of hourly sales table.

The records are put in rows and columns as follows:

```
hbase (main) 001:0> put 'ACVM_id', '2206', 'DT',  
'121217', 'HR', '16', 'HourlySales: KKHS', '28' 0  
row(s) in 021120 seconds  
hbase (main) 002:0> put 'ACVM_id', '2206',  
'HourlySales: MHS', '23' 0 row(s) in 001120 seconds  
hbase (main) 003:0> put 'ACVM_id', '2206',  
'HourlySales: FNHS', '38' 0 row(s) in 021120 seconds  
hbase (main) 004:0> put 'ACVM_id', '2206',  
'HourlySales: NHS', '8' 0 row(s) in 001120 seconds  
hbase (main) 005:0> put 'ACVM_id', '2206',  
'HourlySales: OHS', '50' 0 row(s) in 001120 seconds
```

2.6.4 Hive

Apache Hive is an open-source data warehouse software. Hive facilitates reading, writing and managing large datasets which are at distributed Hadoop files. Hive uses SQL. Hive puts a partial SQL interface in front of Hadoop.

Hive design provisions for batch processing of large sets of data. An application of Hive is for managing weblogs. Hive does not process real-time queries and does not update row-based data tables.

Hive also enables data serialization/deserialization and increases flexibility in

schema design by including a system catalog called Hive Metastore. HQL also supports custom MapReduce scripts to be plugged into queries.

Hive supports different storage types, such as text files, sequence files (consisting of binary key/value pairs) and RCFiles (Record Columnar Files), ORC (optimized row columnar) and HBase.

Three major functions of Hive are data summarization, query and analysis. Hive basically interacts with structured data stored in HDFS with a query language known as HQL (Hive Query Language)

which is similar to SQL. HQL translates SQL-like queries into MapReduce jobs executed on Hadoop automatically.

Sections 4.4 and 4.5 will describe the Hive and HiveQL in detail.

2.6.5 Pig

Apache Pig is an open source, high-level language platform. Pig was developed for analyzing large-data sets. Pig executes queries on large datasets that are stored in HDFS using Apache Hadoop. The language used in Pig is known as Pig Latin.

Pig Latin language is similar to SQL query language but applies on larger datasets. Additional features of Pig are as follows:

- (i) Loads the data after applying the required filters and dumps the data in the desired format.
- (ii) Requires Java runtime environment for executing Pig Latin programs.
- (iii) Converts all the operations into map and reduce tasks. The tasks run on Hadoop.
- (iv) Allows concentrating upon the complete operation, irrespective of the individual Mapper and Reducer functions to produce the output results.

Section 4.6 will describe the usages of Pig in detail.

2.6.6 Mahout

Mahout is a project of Apache with library of scalable machine learning algorithms. Apache implemented Mahout on top of Hadoop. Apache used the

MapReduce paradigm. Machine learning is mostly required to enhance the future performance of a system based on the previous outcomes. Mahout provides the learning tools to automate the finding of meaningful patterns in the Big Data sets stored in the HDFS.

Mahout supports four main areas:

- Collaborative data-filtering that mines user behavior and makes product recommendations.
- Clustering that takes data items in a particular class, and organizes them into naturally occurring groups, such that items belonging to the same group are similar to each other.
- Classification that means learning from existing categorizations and then assigning the future items to the best category.
- Frequent item-set mining that analyzes items in a group and then identifies which items usually occur together.

Section 6.9 will describe Mahout architecture and usages.

Self-Assessment Exercise linked to LO 2.5

1. Why is ZooKeeper required to behave as a centralized repository where the distributed applications can write the data?
2. What are the functions which Ambari perform? How does Ambari enable administering of clusters and Hadoop components?
3. Make a table of ecosystem tools and their functions which are required for analyzing performances from SGPs, SGPAAs and CGPAs of each student. Assume that programmes are Master of Science in Computer Science, Master of Computer Applications and Master of Technology in Computer Science.
4. What are the activities which Mahout supports in the Hadoop system?



KEY CONCEPTS

active node
administering cluster
Ambari
application master
AVRO
Chukawa
cluster
columnar data
container
data Block
data node
data replication
Flink
Flume
Hadoop
Hadoop Common
Hadoop pipes
Hadoop streaming
HBase
HDFS
Hive
Mahout
managing cluster
Mapper
MapReduce
node manager
Oozie
parallel tasks

Pig
primary master
Rack
Reducer
resource
resource manager
resource scheduling
row-based data
secondary master
serialization
shell command
slave node
Spark
standby node
synchronization
YARN
ZooKeeper



LO 2.1

- Hadoop is an open-source framework that uses cloud-based utility computing services. Tera Bytes of data processing takes just a few minutes.
- Hadoop system has features of fault tolerant, scalable, flexible, modular design and distributed clusters computing model with data locality.
- Hadoop core components are Hadoop Common, that uses the libraries and

utilities, HDFS, MapReduce and YARN.

- Hadoop ecosystem includes the application support layer and application layer components - AVRO, ZooKeeper, Sqoop, Ambari, Chukwa, Flink and Flume, Pig, Hive, Spark and Mahout.
- HDFS with MapReduce YARN-based system enables parallel processing of large data sets.

LO 2.2

- HDFS is a Java-based distributed file system that can store various types of data.
- Hadoop stores the data in a number of *clusters*. Each cluster has a number of Data Store called *Racks*. Each Rack stores a number of DataNodes. Each DataNode has a large number of *Data Blocks*. The data blocks *replicate by default at least on three DataNodes* in the same or remote nodes.
- Files, data blocks and DataNodes need identification during processing at Hadoop DataNodes. The concept of the NameNode and DataNode associate the HDFS. A NameNode stores meta data for the files.
- Meta data gives information about the file of user application, but does not participate in the computations. DataNode stores the actual data files in the data blocks.
- Provision for multiple NameNodes enables higher resources availability.

LO 2.3

- MapReduce functions are an integral part of Hadoop physical organization.
- MapReduce is a programming model for distributed computing. MapReduce allows writing applications to process huge amounts of data, in parallel, on large clusters of servers reliably.
- The parallel programs of MapReduce are useful for performing large-scale data analysis using multiple CPUs at nodes in a cluster.

LO 2.4

- YARN separates the resource management and processing components. An application consists of a number of tasks and each task can consist of a number of sub-tasks (threads), which run in parallel. YARN enables running of multi-threaded applications. YARN manages and allocates the resources for the application *sub-tasks* and submits the resources for them at the Hadoop.
- YARN schedules and handles the resource requests of large scale, distributed applications.

LO 2.5

- Avro is a data transfer utility which provisions a system which enables data serialization for transfer between the application and processing layers.
- ZooKeeper is a coordination service for the distributed running of applications and tasks.
- Sqoop is a data-transfer software for data-transfer between data stores and relational DBs.
- Ambari is a web-based tool which provisions, monitors, manages, viewing of functioning of clusters, MapReduce, Hive and Pig APIs.
- Cassandra is a database which provisions a scalable and fault-tolerant database for multiple masters.
- Chukwa is a data collection system for large and distributed systems.
- HBase is a structured data store using database that provisions for a scalable and structured database for large tables.
- Hive is a data warehouse system which provisions for queries, data aggregation, summarizing, infrastructure-like enterprise data warehouse, data summarization, ad hoc (unstructured) querying and HiveQL which is SQL-like scripting language.

- Pig is a high-level dataflow language which provisions for dataflow functionality and the execution framework for parallel computations.
- Mahout is a software library for the machine learning algorithms.

Objective Type Questions

Select one correct-answer option for each questions below:

2.1 Programming model for Big Data is (i) centralized computing of input results of the applications from multiple computing nodes, (ii) the distributed computing of an application at the same time. Data sets and the application run at the MPPs at a number of geographic locations and remote servers. (iii) Distributed computing of the data sets using application tasks at the multiple computing nodes. (iv) Computing of the application codes transferred to the multiple nodes which store data sets and compute at cluster.

- (a) i
- (b) ii to iv
- (c) iii and iv
- (d) iv

2.2 Core components of Hadoop are (i) Hadoop Common which contains the libraries and utilities required by other modules of Hadoop, (ii) a Java-based distributed file system, (iii) MapReduce, (iv) YARN, (v) AVRO and ZooKeeper, (vi) Pig, Hive, Sqoop and (v) Ambari.

- (a) all are true
- (b) (i) to (iv)
- (c) all except vi
- (d) ii to vi

2.3 (i) HDFS design is for batch processing and cannot be used for stream analytics. (ii) Spark can be used for Hadoop stream analytics. (iii) YARN has made it possible to process applications, such as interactive queries, text

analytics and streaming analysis. (iv) Flume can be used stream analytics. (v) Spark and Flink technologies are the most suitable for in-stream processing.

- (a) all except iv
- (b) all except ii
- (c) all except i
- (d) all

2.4 Hadoop distributed file system: (i) identifies the file by directories and folders which associate with file system, (ii) identifies the data sources for processing and uses the resource pointers which store in the data dictionary, (iii) identifies the data block using data dictionary master tables stored at a central location, (iv) identifies using centralized tables, and (v) identifies from file meta data at the application.

- (a) none
- (b) ii to v
- (c) ii, iv and v
- (d) all except ii

2.5 (i) HDFS uses the client, master NameNode, primary and secondary MasterNodes and slave nodes. (ii) YARN components use Client, Resource Manager (RM), Node Manager (RM), Application Master (AM) and Containers. (iii) YARN uses the client, master NameNode, primary and secondary MasterNode and slave nodes. (iv) MapReduce v2 when Hadoop uses YARN-based system, which enables parallel processing of large data sets. (v) Slaves are responsible to store the data and process the computation tasks submitted by the clients.

- (a) Only i
- (b) all except ii ad iv.
- (c) all except iii
- (d) all

2.6 (1) Hadoop shell commands are (i) – copyToLocal for copying a file at HDFS to the local and (ii) – cat for copying to standard output (stdout). (2) When file stuData_id96 to be copied at stu_filesdir directory, then command is (iii) \$Hadoop hdfs-put stuData_id96 /user/ stu_filesdir, and (iv) \$Hadoop hdfs-cp stuData_id96 /user/ stu_filesdir.

- (a) ii to iv
- (b) i to iii
- (c) all
- (d) only i and iii

2.7 (i) NM is a slave of the infrastructure. (ii) AM signals whenever it initializes. (iii) All active AMs send the controlling signal periodically to the RM when signaling their presence. (iv) NM accepts the request and queues up the resources for application program or sub-tasks. When the requested resources become available on slave nodes, (v) the RM grants the Application Master usage permission for the specific intervals for the containers on specific slave NMs. The systems do not use the concept of joins (in distributed data storage systems), and (vi) A Client Node submits request of an application to the RM. The RM is the master.

- (a) all except ii to iv
- (b) all
- (c) only iii and iv
- (d) ii to iv

2.8 HBASE (i) applies a partial columnar scheme on top of the MapReduce. It is (ii) an open-source, distributed, versioned, non-relational (NoSQL) database, (iii) written in Java, (iv) stores large unstructured table and (v) provisions for the scalable distributed Big Data Store. Data stores as key-value pairs and (vi) consists of a set of tables. Each table contains rows and columns. Therefore, it is similar to a traditional database and (vii) provisions a primary key as in the database table.

- (a) i to iii

- (b) all except v
- (c) all except i and iv
- (d) all

2.9 (i) Ambari is a structured data store using database that provisions for a scalable and structured database for large tables. (ii) Zookeeper in Hadoop behaves as a centralized repository where distributed applications can put data at a node. (iv) Cassandra is a data collection system for large and distributed systems. (v) Zookeeper is a coordination service that enables synchronization across a cluster in distributed applications.

- (a) all
- (b) only ii and iii
- (c) all except ii and iii
- (d) none

2.10 (i) Ambari is a web-based tool which provisions, monitors, manages, viewing of cluster functioning. (ii) Ambari and BigInsights have provisions for viewing. Their uses are for administering the Hadoop. (iii) Avro is a data transfer utility between application and application support layer. (iii) Cassandra is a database which provisions a scalable and fault-tolerant database for multiple masters. (iv) Chukwa is a data collection system for large and distributed systems. (v) HBase is a structured data store using database that provisions for a scalable and structured database for large tables.

- (a) all correct except iii
- (b) all except iii and v
- (c) all except iv
- (d) all except ii and iii

2.11 (i) Hadoop 1 and 2 provisions for multiple NameNodes. (ii) Each MasterNode (MN) has NameNode, Avro coordination client, and JournalNode (JN). (iii) A JN keeps the records of the state, resources

required, intermediate results or cluster tasks execution. (iv) Application tasks and subtasks at the cluster can write data and read data from a JN.

- (a) all correct except iii
- (b) all except iii and iv
- (c) all except ii
- (d) only iv

2.12 (i) Oozie provides a way to package and bundle multiple coordinator and workflow jobs and manage the lifecycle of those jobs. (ii) Flink provisions for reliable data transfer and provides for recovery in case of failure. Transfers large amount of data in applications. (iii) Chukwa provisions data serialization during data transfer between application and processing layers.

(iv) Sqoop provisions for data transfer between data stores such as relational DBs and Hadoop. (v) Chukwa provisions and manages data collection system for large and distributed systems.

- (a) all correct except iii
- (b) all except iii and v
- (c) all except iv
- (d) all except ii and iii

Review Questions

- 2.1 Why is the application program layer different from support layer in Big Data platform? Explain the Hadoop features. **(LO 2.1)**
- 2.2 List Hadoop two core components. Describe their usages. **(LO 2.1)**
- 2.3 Explain using a diagram the distributed storage, resource manager layer, processing framework and application APIs layers in Hadoop. **(LO 2.1)**
- 2.4 Give the meanings of Hadoop distributed file system, clusters, Racks, DataNodes, Data Blocks, MasterNode, NameNode and metadata of files. Explain these. **(LO 2.2)**

- 2.5 How do multiple NameNodes ensure high availability of Data in HDFS? **(LO 2.2)**
- 2.6 How does MapReduce function as a programming model for distributed computing? **(LO 2.3)**
- 2.7 How does MapReduce enables process huge amounts of data, in parallel, on large clusters of servers reliably. **(LO 2.3)**
- 2.8 List the resources required to run an application. How does the separation of resource management and processing components help the number of tasks and sub-tasks (threads) when running in parallel? **(LO 2.4)**
- 2.9 How does YARN resource manager do the following: (i) keep track of the active node managers and available resources and (ii) allocate the containers to the appropriate sub-tasks and monitors the Application Master? **(LO 2.4)**
- 2.10 Why are AVRO and Zookeeper essential in Hadoop programming for Big Data applications? **(LO 2.5)**

Practice Exercises

- 2.1 Recapitulate Example 2.3. Assume a company collects data from a large number of automatic chocolate vending machines (ACVMs) distributed over a large number of geographic locations in the cities and areas in each city. Each ACVM sells five different flavors of chocolates: KitKat, Milk, Fruit and Nuts, Nougat and Oreo. When will the centralized processing and analytics model and when will Hadoop programming be used? Each machine communicates data for the analytics every hour to the company's central data warehouse. **(LO 2.1)**
- 2.2 Make a data store model using HDFS for SGPs, SGPAAs and CGPAs of each student in 50 UG and 10 PG offered at the university with 5000 students intake capacity each year. Each student information can extend up to 64 MB. **(LO 2.2)**

2.3 Recapitulate Example 1.6 of a Automotive Components and Predictive Automotive Maintenance Services (ACPAMS) company which renders customer services for maintenance and servicing of (Internet) connected cars and its components. Assume that number of centres are 8192 ($=2^{13}$) , number of car serviced by each centre per day equals 32 ($=2^5$). Each car has 256 ($=2^8$) components, which requires maintenance or servicing in the company's car. The service centre also collects feedback after every service and send responses to customer requests. The feedback and responses text takes on average 128 B ($=2^7$ B) and each service or responses records in a report of average 512 B ($=2^9$) text. The company stores the centres data for maximum 10 years and follows last-in first-out data replacement policy. How will the files of ACPAMS be stored using HDFS? What shall be the minimum memory requirement during 10 years? **(LO 2.2)**

2.4 Consider a car company selling five models of car: *Jagaur Land Rover*, *Hexa*, *Zest*, *Nexon* and *Safari Storme*. Assume each day the model sells at 600 show rooms with average 400 car sales average per week in a year. The files for each model are at file_1, file_2,, file_5. Each file stores as the key-value pairs the daily sales log at the company large number of showrooms.

- (i) Calculate how and how much will the showrooms weekly each model sales-log store using HDFS?
- (ii) Write the sample of data collected in a file for day 1, 2,, 365 starting January 1 from the showrooms.
- (iii) What will be the output streams of map tasks for feeding the input streams to the Reducer?
- (iv) What will be the Reducer outputs? **(LO 2.3)**

2.5 Recapitulate a list of actions of YARN resource allocation and scheduling functions in Section 2.5.1 Figure 2.5. Show the directions of sequences and step number over each arrow, to show the sequence of action to allocation of a container. **(LO 2.4)**

2.6 Recapitulate Practice Exercise 2.4, Consider a car company selling *Jaguar Land Rover*, *Hexa*, *Zest*, *Nexon* and *Safari Storme* models of car. Following

is the table for the weekly sales log at the multiple car company showrooms.

<i>CCSR_id</i>	<i>Date (DT) mmddyy</i>	<i>Jagaur Land Rover Weekly Sales (JLRWS)</i>	<i>Hexa Weekly Sales (HWS)</i>	<i>Zest Weekly Sales (ZWS)</i>	<i>Nexon Weekly Sales (NWS)</i>	<i>Safari Strome Weekly Sales (SSWS)</i>
220	121217	28	23	138	148	50
10	121217	49	34	164	115	38
122	121217	40	141	123	37	88
16	121217	13	25	127	158	174
28	121217	12	122	116	128	57
-	-	-	-	-	-	-
-	-	-	-	-	-	-

- (i) How the HBase stores the table in five weeks.
 - (ii) How will the records created using shell command ‘put’ for 35 entries given above?
- (LO 2.5)**
-

Note:

- Level 1 & Level 2 category
- Level 3 & Level 4 category
- Level 5 & Level 6 category

Chapter 3

NoSQL Big Data Management, MongoDB and Cassandra

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 3.1 Get conceptual understanding of NoSQL data stores, Big Data solutions, schema-less models, and increased flexibility for data manipulation
- LO 3.2 Get knowledge of NoSQL data architecture patterns namely, key-value pairs, tabular, column family, BigTable, Record Columnar (RC), Optimized Row Columnar (ORC) and Parquet, document, object and graph data stores, and the variations in architectural patterns
- LO 3.3 Get conceptual understanding of NoSQL data store management, applications and handling problems in Big Data
- LO 3.4 Solve Big Data analytics using shared-nothing architecture, choosing a distribution model among master-slave and peer-to-peer models, and get the knowledge of four ways by which the NoSQL handles the Big Data problems
- LO 3.5 Apply the MongoDB databases and query commands
- LO 3.6 Use the Cassandra databases, data model, clients, and integrate them with Hadoop

RECALL FROM PREVIOUS CHAPTERS

Big Data use new tools for processing and analysis of large volume of data. Big Data sources are Hadoop or Spark compatible file system, structured, unstructured or NoSQL data Store (Table 1.1). Big Data distributed computing uses shared-nothing paradigm, *no in-between data sharing and inter-processor communication.* (Table 1.2)

Chapter 1 introduced NoSQL. NoSQL data stores can store semi-structured or unstructured data. NoSQL stands for No-SQL or Not Only SQL. NoSQL databases can coexists with SQL databases. NoSQL data applications do not integrate with SQL databases applications. NoSQL databases store Big Data. Examples of NoSQL data stores are key-value pairs, hash key, JSON files, BigTable, HBase, MongoDB, Cassandra, and CouchDB (Section 1.6.2.1).

This chapter focuses on providing detailed concepts of NoSQL data architectural patterns, management of Big Data, data distribution models, handling of Big Data problems using NoSQL, MongoDB for document and Cassandra for columnar stores.

3.1 | INTRODUCTION

Big Data uses distributed systems. A distributed system consists of multiple data nodes at clusters of machines and distributed software components. The tasks execute in parallel with data at nodes in clusters. The computing nodes communicate with the applications through a network.

Following are the features of distributed-computing architecture (Chapter 2):

1. *Increased reliability and fault tolerance:* The important advantage of distributed computing system is reliability. If a segment of machines in a cluster fails then the rest of the machines continue work. When the datasets replicate at number of data nodes, the fault tolerance increases further. The dataset in remaining segments continue the same computations as being done at failed segment machines.

2. *Flexibility* makes it very easy to install, implement and debug new services in a distributed environment.
3. *Sharding* is storing the different parts of data onto different sets of data nodes, clusters or servers. For example, university students huge database, on sharding divides in databases, called shards. Each shard may correspond to a database for an individual course and year. Each shard stores at different nodes or servers.
4. *Speed*: Computing power increases in a distributed computing system as shards run parallelly on individual data nodes in clusters independently (no data sharing between shards).
5. *Scalability*: Consider sharding of a large database into a number of shards, distributed for computing in different systems. When the database expands further, then adding more machines and increasing the number of shards provides horizontal scalability. Increased computing power and running number of algorithms on the same machines provides vertical scalability (Section 1.3.1).
6. *Resources sharing*: Shared resources of memory, machines and network architecture reduce the cost.
7. *Open system* makes the service accessible to all nodes.
8. *Performance*: The collection of processors in the system provides higher performance than a centralized computer, due to lesser cost of communication among machines (Cost means time taken up in communication).

The demerits of distributed computing are: (i) issues in troubleshooting in a larger networking infrastructure, (ii) additional software requirements and (iii) security risks for data and resources.

Big Data solutions require a scalable distributed computing model with shared-nothing architecture. A solution is Big Data store in HDFS files. NoSQL data also store Big Data, and facilitate random read/write accesses. The accesses are sequential in HDFS data.

HBase is a NoSQL solution (Section 2.6.3). Examples of other solutions are

MongoDB and Cassandra. MongoDB and Cassandra DBMSs create HDFS compatible distributed data stores and include their specific query processing languages.

Following are selected key terms used in database systems.

Class refers to a template of program codes that is extendable. Class creates instances, called objects. A class consists of initial values for member fields, called *state* (of variables), and implementations of member functions and methods called *behavior*. An implementation means program codes along with values of arguments in the functions and methods (Java Class uses methods, C++ functions.) An abstract class consists of at least one abstract member or method.

Object is an instance of a class in Java, C++, and other object-oriented languages. Object can be an instance of another object (for example, in JavaScript).

Tupple is an ordered set of data which constitutes a record. For example, one row record in a table. A row in a relational database has column fields or attributes. Example of a tupple is (JLRWSale, Week 1, 138, Week 2, 232, ..., week 52, 186) in an RDBMS table. Here, JLRWSale means Jaguar Land Rover Weekly Sale. (JLRWSale, Week 1, 138) is also a tupple, and gives JLR week 1 sales = 138. (Week 2, 232, ..., week 52, 186) means week 2 sales = 232 abd 52 sales = 186 JLRs.

Transaction means execution of instructions in two interrelated entities, such as a query and the database.

Database transactional model refers to a model for transactions, such as the one following the ACID (Section 3.2) or BASE properties (Section 3.2.3).

MySQL refers to a widely used open-source database, which excels as a content management server.

Oracle refers to a widely used object-relational DBMS, written in the C++ language that provides applications integration with service-oriented architectures and has high reliability. Oracle has also released the NoSQL database system.

DB2 refers to a family of database server products from IBM with built-in support to handle advanced Big Data analytics.

Sybase refers to database server based on relational model for businesses, primarily on UNIX. Sybase was the first enterprise-level DBMS in Linux.

MS SQL server refers to a Microsoft-developed RDBMS for enterprise-level databases that supports both SQL and NoSQL architectures.

PostgreSQL refers to an enterprise-level, object-relational DBMS. PostgreSQL uses procedural languages like Perl and Python, in addition to SQL.

This chapter describes NoSQL data architecture patterns, NoSQL for increasing the flexibility in data store architecture, NoSQL usages in Big Data management, and the solutions, such as MongoDB and Cassandra. Section 3.2 describes NoSQL data stores for Big Data usages, schema-less models, and increasing the flexibility for data manipulation. Section 3.3 describes NoSQL data-architecture patterns: Key-value stores, graph stores, column family stores, tabular stores, document stores, object data stores, and variations of NoSQL architectural patterns. Section 3.4 describes NoSQL for managing Big Data, solutions for Big Data, and types of Big Data problems. Section 3.5 describes use of shared-nothing architecture, choosing a distribution model, master-slave versus peer-to-peer, and four ways by which NoSQL handles Big Data problems. Sections 3.6 describes MongoDB and query commands used in the applications. Section 3.7 describes Cassandra databases, data-model, clients and integration with Hadoop and applications.

3.2 | NOSQL DATA STORE

SQL is a programming language based on relational algebra. It is a declarative language and it defines the data schema . SQL creates databases and RDBMSs. RDBMS uses tabular data store with relational algebra, precisely defined operators with relations as the operands. Relations are a set of tuples. Tuples are named attributes. A tuple identifies uniquely by keys called candidate keys.

LO 3.1

NoSQL data store, Big Data solutions, schema-less models, and increasing flexibility for data manipulation

Transactions on SQL databases exhibit ACID properties. ACID stands for atomicity, consistency, isolation and durability.

ACID Properties in SQL Transactions

Following are the meanings of these characteristics during the transactions.

Atomicity of transaction means all operations in the transaction must complete, and if interrupted, then must be undone (rolled back). For example, if a customer withdraws an amount then the bank in first operation enters the withdrawn amount in the table and in the next operation modifies the balance with new amount available. Atomicity means both should be completed, else undone if interrupted in between.

Consistency in transactions means that a transaction must maintain the integrity constraint, and follow the consistency principle. For example, the difference of sum of deposited amounts and withdrawn amounts in a bank account must equal the last balance. All three data need to be consistent.

Isolation of transactions means two transactions of the database must be isolated from each other and done separately.

Durability means a transaction must persist once completed.

Triggers, Views and Schedules in SQL Databases

Trigger is a special stored procedure. Trigger executes when a specific action(s) occurs within a database, such as change in table data or actions such as UPDATE, INSERT and DELETE. For example, a Trigger store procedure inserts new columns in the columnar family data store.

View refers to a logical construct, used in query statements. A View saves a division of complex query instructions and that reduces the query complexity. Viewing of a division is similar to a view of a table. View does not save like data at the table. Query statement when uses references to a view, the statement executes the View. Query (processing) planner combines the information in View definition with the remaining actions on the query. A query planner plans how to break a query into sub-queries for obtaining the required answer. View, hides the query complexity by dividing the query into smaller, more manageable pieces.

Schedule refers to a chronological sequence of instructions which execute concurrently. When a transaction is in the schedule then all instructions of the transaction are included in the schedule. Scheduled order of instructions is maintained during the transaction. Scheduling enables execution of multiple transactions in allotted time intervals.

Join in SQL Databases

SQL databases facilitate combining rows from two or more tables, based on the related columns in them. *Combining* action uses *Join* function during a database transaction. *Join* refers to a clause which combines. Combining the products (AND operations) follows next the selection process. A Join operation does pairing of two tuples obtained from different relational expressions. Joins, if and only if a given Join condition satisfies. Number of Join operations specify using relational algebraic expressions. SQL provides JOIN clause, which retrieves and joins the related data stored across multiple tables with a single command, Join. For example, consider an SQL statement:

```
Select KitKatSales From TransactionsTbl INNER JOIN ACVMSalesTbl ON TransactionsTbl.KitKatSales = TransactionsTbl.KitKatSales;
```

The statement selects those records in a column named KitKatSales which match the values in two tables: one TransactionsTbl and other ACVMSalesTbl.

Relational databases and RDBMS developed using SQL have issues of scalability and distributed design. This is because all tuples need to be on the same data node. The database has an issue of indexing over distributed nodes. They do not model the hierarchical, object-oriented, semi-structured or graph databases.

Database Tables have relationships between them which are represented by related fields. RDBMS allows the Join operations on the related columns. The traditional RDBMS has a problem when storing the records beyond a certain physical storage limit. This is because RDBMS does not support horizontal scalability (Section 1.3.1).

For example, consider sharding a big table in a DBMS into two. Assume writing first 0.1 million records (1 to 100000) in one table and from 100001 in another table. Sharding a database means breaking up into many, much smaller databases that share nothing, and can distribute across multiple servers. Handling of the Joins and managing data in the other related tables are cumbersome processes, when using the sharding.

The problem continues when data has no defined number of fields and

formats. For example, the data associated with the choice of chocolate flavours of the users of ACVM in Example 1.6(i). Some users provide a single choice, while some users provide two choices, and a few others want to fill three best flavours of their choice.

User Id	Choice
1	Dairy Milk
2	Dairy Milk, KitKat
3	KitKat, Snicker, Munch

Defining a field becomes tough when a field in the database offers choice between two or many. This makes RDBMS unsuitable for data management in Big Data environments as well as data in their real forms.

SQL compliant format means that database tables constructed using SQL and they enable processing of the queries written using SQL. ‘NoSQL’ term conveys two different meanings: (i) does not follow SQL compliant formats, (ii) “Not only SQL” use SQL compliant formats with variety of other querying and access methods.

3.2.1 NoSQL

A new category of data stores is NoSQL (means Not Only SQL) data stores. NoSQL is an altogether new approach of thinking about databases, such as schema flexibility, simple relationships, dynamic schemas, auto sharding, replication, integrated caching, horizontal scalability of shards, distributable tuples, semi-structures data and flexibility in approach.

Issues with NoSQL data stores are lack of standardization in approaches, processing difficulties for complex queries, dependence on eventually consistent results in place of consistency in all states.

3.2.1.1 Big Data NoSQL or Not-Only SQL

NoSQL DB does not require specialized RDBMS like storage and hardware for processing. Storage can be on a cloud. Section 1.6.2.1 introduced NoSQL data storage system. NoSQL records are in non-relational data store systems. They use flexible data models. The records use multiple schemas.

NoSQL or Not Only SQL is a Class of non-relational data storage systems, flexible data models and multiple schema.

NoSQL data stores are considered as semi-structured data. Big Data Store uses NoSQL. Figure 1.7 showed co-existence of data store at server or cloud with SQL, RDBMS with NoSQL and Big Data at Hadoop, Spark, Mesos, S3 or compatible Clusters. However, no integration takes place with applications that are based on SQL. NoSQL data store characteristics are as follows:

1. NoSQL is a class of non-relational data storage system with flexible data model. Examples of NoSQL data-architecture patterns of datasets are key-value pairs, name/value pairs, Column family
Big-data store, Tabular data store, Cassandra (used in Facebook/Apache), HBase, hash table [Dynamo (Amazon S3)], unordered keys using JSON (CouchDB), JSON (PNUTS), JSON (MongoDB), Graph Store, Object Store, ordered keys and semi-structured data storage systems.
2. NoSQL not necessarily has a fixed schema, such as table; do not use the concept of Joins (in distributed data storage systems); Data written at one node can be replicated to multiple nodes. Data store is thus fault-tolerant. The store can be partitioned into unshared shards.

Features in NoSQL Transactions NoSQL transactions have following features:

- (i) Relax one or more of the ACID properties.
- (ii) Characterize by two out of three properties (consistency, availability and partitions) of CAP theorem, two are at least present for the application/service/process.
- (iii) Can be characterized by BASE properties (Section 3.2.3).

Big Data NoSQL solutions use standalone-server, master-slave and peer-to-peer distribution models.

Big Data NoSQL Solutions NoSQL DBs are needed for Big Data solutions. They play an important role in handling Big Data challenges. Table 3.1 gives the examples of widely used NoSQL data stores.

Table 3.1 NoSQL data stores and their characteristic features

NoSQL Data store	Description

Apache's HBase	HDFS compatible, open-source and non-relational data store written in Java; A column-family based NoSQL data store, data store providing BigTable-like capabilities (Sections 2.6 and 3.3.3.2); scalability, strong consistency, versioning, configuring and maintaining data store characteristics
Apache's MongoDB	HDFS compatible; master-slave distribution model (Section 3.5.1.3); document-oriented data store with JSON-like documents and dynamic schemas; open-source, NoSQL, scalable and non-relational database; used by Websites Craigslist, eBay, Foursquare at the backend
Apache's Cassandra	HDFS compatible DBs; decentralized distribution peer-to-peer model (Section 3.5.1.4); open source; NoSQL; scalable, non-relational, column-family based, fault-tolerant and tuneable consistency (Section 3.7) used by Facebook and Instagram
Apache's CouchDB	A project of Apache which is also widely used database for the web. CouchDB consists of Document Store. It uses the JSON data exchange format to store its documents, JavaScript for indexing, combining and transforming documents, and HTTP APIs
Oracle NoSQL	Step towards NoSQL data store; distributed key-value data store; provides transactional semantics for data manipulation, horizontal scalability, simple administration and monitoring
Riak	An open-source key-value store; high availability (using replication concept), fault tolerance, operational simplicity, scalability and written in Erlang

CAP Theorem Among C, A and P, two are at least present for the application/service/process. *Consistency* means all copies have the same value like in traditional DBs. *Availability* means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, the other copy in the other partition is available. *Partition* means parts which are active but may not cooperate (share) as in distributed DBs.

1. *Consistency in distributed databases* means that all nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database. Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.

2. *Availability* means that during the transactions, the field values must be available in other partitions of the database so that each request receives a response on success as well as failure. (Failure causes the response to request from the replicate of data). Distributed databases require transparency between one another. Network failure may lead to data unavailability in a certain partition in case of no replication. Replication ensures availability.
3. *Partition* means division of a large database into different databases without affecting the operations on them by adopting specified procedures.

Partition tolerance: Refers to continuation of operations as a whole even in case of message loss, node failure or node not reachable.

Brewer's CAP (Consistency, Availability and Partition Tolerance) theorem demonstrates that any distributed system cannot guarantee C, A and P together.

1. Consistency- All nodes observe the same data at the same time.
2. Availability- Each request receives a response on success/failure.
3. Partition Tolerance-The system continues to operate as a whole even in case of message loss, node failure or node not reachable.

Partition tolerance cannot be overlooked for achieving reliability in a distributed database system. Thus, in case of any network failure, a choice can be:

- Database must answer, and that answer would be old or wrong data (AP).
- Database should not answer, unless it receives the latest copy of the data (CP).

The CAP theorem implies that for a network partition system, the choice of consistency and availability are mutually exclusive. CA means consistency and availability, AP means availability and partition tolerance and CP means consistency and partition tolerance. Figure 3.1 shows the CAP theorem usage in Big Data Solutions.

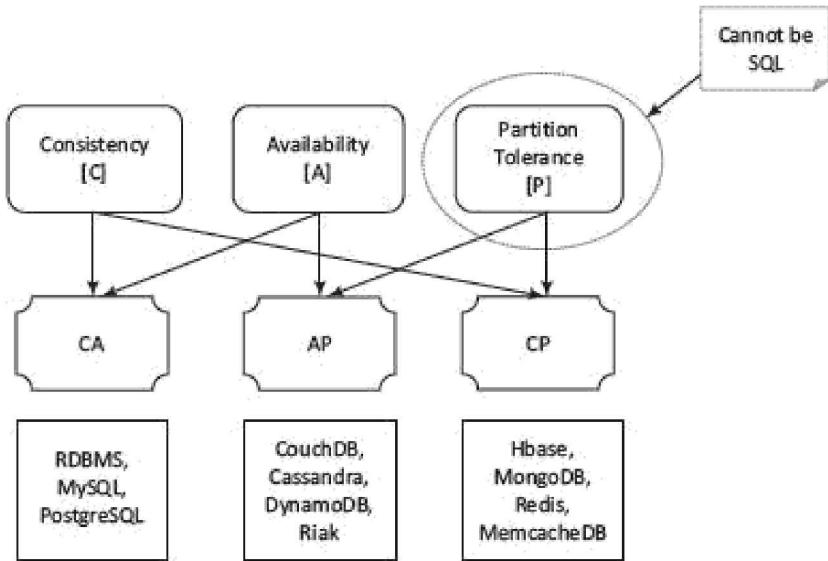


Figure 3.1 CAP theorem in Big Data solutions

3.2.2 Schema-less Models

Schema of a database system refers to designing of a structure for datasets and data structures for storing into the database. NoSQL data not necessarily have a fixed table schema. The systems do not use the concept of Join (between distributed datasets). A cluster-based highly distributed node manages a single large data store with a NoSQL DB.

Data written at one node replicates to multiple nodes. Therefore, these are identical, fault-tolerant and partitioned into shards. Distributed databases can store and process a set of information on more than one computing nodes.

NoSQL data model offers relaxation in one or more of the ACID properties (Atomicity, consistency, isolation and durability) of the database. Distribution follows CAP theorem. CAP theorem states that out of the three properties, two must at least be present for the application/service/process. (Section 3.2.1).

Figure 3.2 shows characteristics of Schema-less model for data stores. ER stands for entity-relation modelling.

Relations in a database build the connections between various tables of data. For example, a table of subjects offered in an academic programme can be connected to a table of programmes offered in the academic institution. NoSQL

data stores use non-mathematical relations but store this information as an aggregate called metadata.

Metadata refers to data describing and specifying an object or objects. Metadata is a record with all the information about a particular dataset and the inter-linkages. Metadata helps in selecting an object, specifications of the data and, usages that design where and when. Metadata specifies access permissions, attributes of the objects and enables additions of an attribute layer to the objects. Files, tables, documents and images are also the objects.

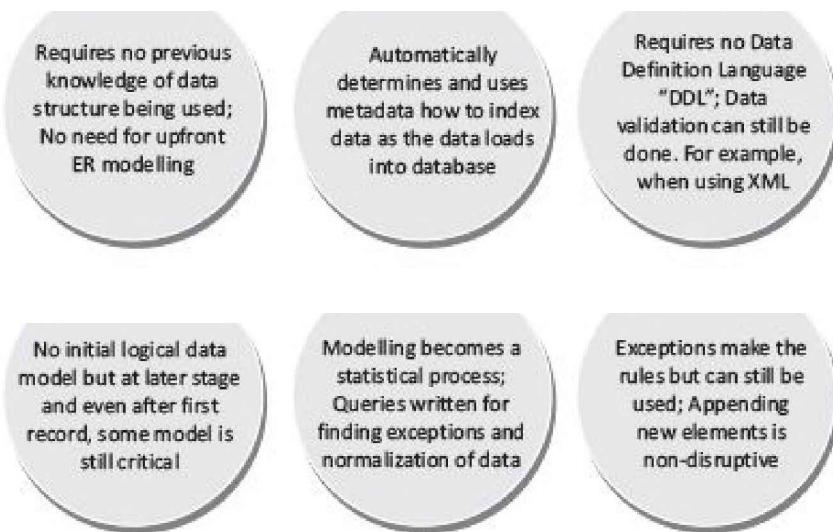


Figure 3.2 Characteristics of Schema-less model

3.2.3 Increasing Flexibility for Data Manipulation

Consider database ‘Contacts’. They follow a fixed schema. Now consider students’ admission database. That also follow a fixed schema. Later, additional data is added as the course progresses. NoSQL data store characteristics are schema-less. The additional data may not be structured and follow fixed schema. The data store consists of additional data, such as documents, blogs, Facebook pages and tweets.

NoSQL data store possess characteristic of increasing flexibility for data manipulation. The new attributes to database can be increasingly added. Late binding of them is also permitted.

BASE is a flexible model for NoSQL data stores. Provisions of BASE increase flexibility.

BASE Properties BA stands for basic availability, S stands for soft state and E stands for eventual consistency.

1. *Basic availability* ensures by distribution of shards (many partitions of huge data store) across many data nodes with a high degree of replication. Then, a segment failure does not necessarily mean a complete data store unavailability.
2. *Soft state* ensures processing even in the presence of inconsistencies but achieving consistency eventually. A program suitably takes into account the inconsistency found during processing. NoSQL database design does not consider the need of consistency all along the processing time.
3. *Eventual consistency* means consistency requirement in NoSQL databases meeting at some point of time in future. Data converges eventually to a consistent state with no time-frame specification for achieving that. ACID rules require consistency all along the processing on completion of each transaction. BASE does not have that requirement and has the flexibility.

BASE model is not necessarily appropriate in all cases but it is flexible and is an alternative to SQL-like adherence to ACID properties. Example 3.11 (Section 3.3.5) explains the concept of BASE in transactions using graph databases.

Schema is not a necessity in NoSQL DB, implying information storage flexibility. Data can store and retrieve without having knowledge of how a database stores and functions internally.

Following is an example to understand the increasing flexibility for data manipulation.

EXAMPLE 3.1

Use examples of database for the students in various university courses to demonstrate the concept of increasing flexibility in NoSQL DBs.

SOLUTION

Figure 3.3 shows increasing flexibility concept using additional data models.

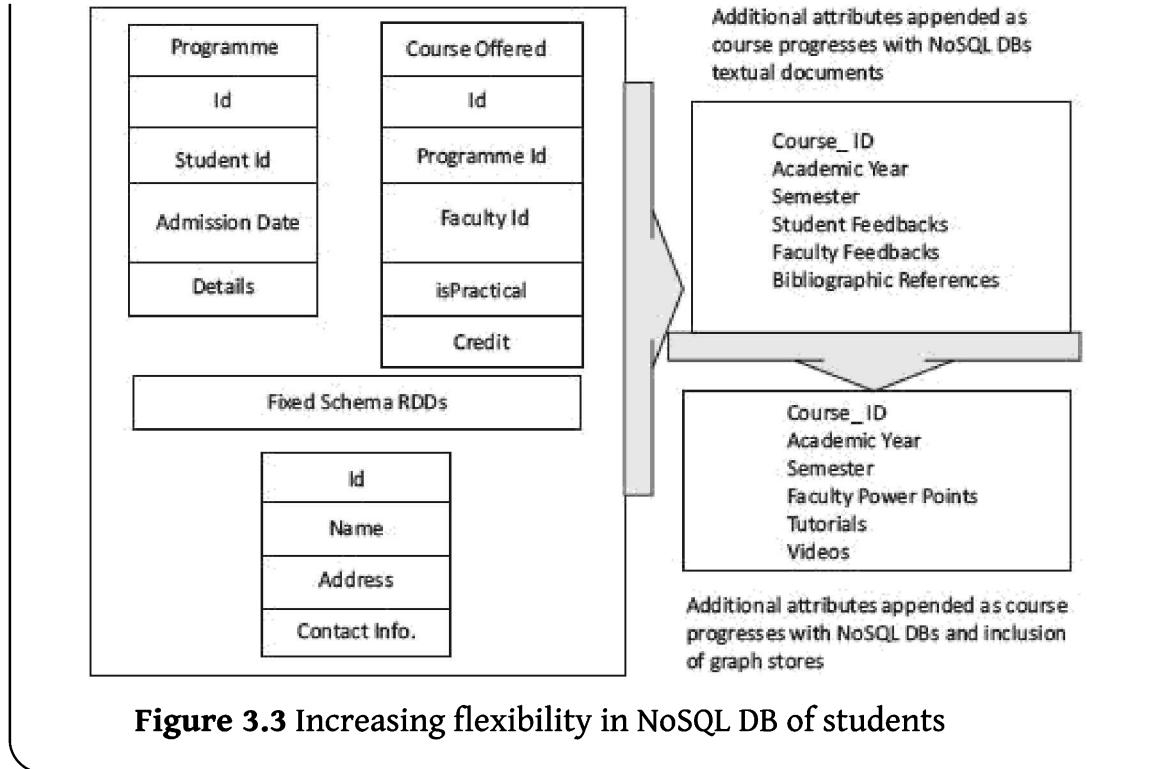


Figure 3.3 Increasing flexibility in NoSQL DB of students

Self-Assessment Exercise linked to LO 3.1

1. Explain when will you use the following: MongoDB, Cassandra, CouchDB, Oracle NoSQL and Riak.
2. How does CAP theorem hold in NoSQL databases?
3. How do ACID and BASE properties differ?
4. Why is the consistency not enforceable in NoSQL distributed databases during a transaction processing?
5. List characteristics of NoSQL data store.
6. Why is metadata a must when using NoSQL data store?

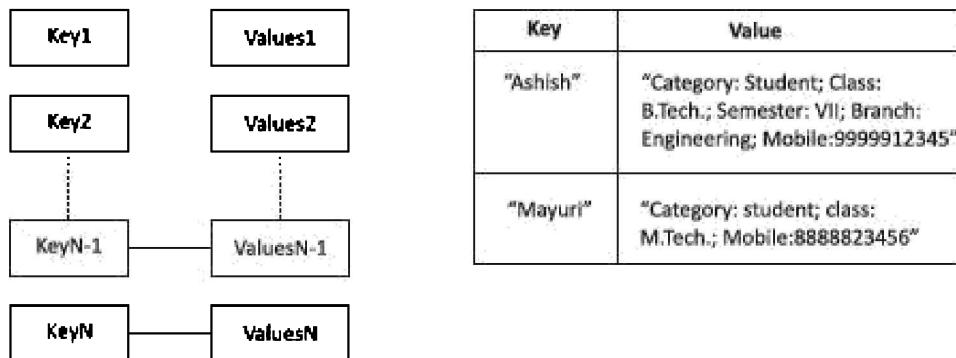
3.3 | NOSQL DATA ARCHITECTURE PATTERNS

NoSQL data stores broadly categorize into architectural patterns described in the following subsections:

NoSQL data-architecture patterns, namely key-value pairs, column family, BigTable, RC, ORC, Parquet, and tabular data stores, document stores, object data stores, graph databases, and data stores with variations in architectural patterns

3.3.1 Key-Value Store

The simplest way to implement a schema-less data store is to use key-value pairs. The data store characteristics are high performance, scalability and flexibility. Data retrieval is fast in key-value pairs data store. A simple string called, key maps to a large data string or BLOB (Basic Large Object). Key-value store accesses use a primary key for accessing the values. Therefore, the store can be easily scaled up for very large data. The concept is similar to a hash table where a unique key points to a particular item(s) of data. Figure 3.4 shows key-value pairs architectural pattern and example of students' database as key-value pairs.



Number of key-values pair, N can be a very large number

Figure 3.4 Example of key-value pairs in data architectural pattern

Advantages of a key-value store are as follows:

1. Data Store can store any data type in a value field. The key-value system stores the information as a BLOB of data (such as text, hypertext, images, video and audio) and return the same BLOB when the data is retrieved. Storage is like an English dictionary. Query for a word retrieves the meanings, usages, different forms as a single item in the dictionary. Similarly, querying for key retrieves the values.

2. A query just requests the values and returns the values as a single item. Values can be of any data type.
3. Key-value store is eventually consistent.
4. Key-value data store may be hierarchical or may be ordered key-value store.
5. Returned values on queries can be used to convert into lists, table-columns, data-frame fields and columns.
6. Have (i) scalability, (ii) reliability, (iii) portability and (iv) low operational cost.
7. The key can be synthetic or auto-generated. The key is flexible and can be represented in many formats: (i) Artificially generated strings created from a hash of a value, (ii) Logical path names to images or files, (iii) REST web-service calls (request response cycles), and (iv) SQL queries.

The key-value store provides client to read and write values using a key as follows:

- (i) Get (key), returns the value associated with the key.
- (ii) Put (key, value), associates the value with the key and updates a value if this key is already present.
- (iii) Multi-get (key1, key2, ..., keyN), returns the list of values associated with the list of keys.
- (iv) Delete (key), removes a key and its value from the data store.

Limitations of key-value store architectural pattern are:

- (i) No indexes are maintained on values, thus a subset of values is not searchable.
- (ii) Key-value store does not provide traditional database capabilities, such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously. The application needs to implement such capabilities.
- (iii) Maintaining unique values as keys may become more difficult when the

volume of data increases. One cannot retrieve a single result when a key-value pair is not uniquely identified.

- (iv) Queries cannot be performed on individual values. No clause like ‘where’ in a relational database usable that filters a result set.

Table 3.2 gives a comparison between traditional relational data model with the key-value store model.

Table 3.2 Traditional relational data model vs. the key-value store model

Traditional relational model	Key-value store model
Result set based on row values	Queries return a single item
Values of rows for large datasets are indexed	No indexes on values
Same data type values in columns	Any data type values

Typical uses of key-value store are: (i) Image store, (ii) Document or file store, (iii) Lookup table, and (iv) Query-cache.

Riak is open-source Erlang language data store. It is a key-value data store system. Data auto-distributes and replicates in Riak. It is thus, fault tolerant and reliable. Some other widely used key-value pairs in NoSQL DBs are Amazon’s DynamoDB, Redis (often referred as Data Structure server), Memcached and its flavours, Berkeley DB, upscaledb (used for embedded databases), project Voldemort and Couchbase.

Concept of Hash Key The following example explains the hash and key-value pairs associated with a hash in traditional data.

EXAMPLE 3.2

Consider an example. Assume that student name is key, k . Each student grade sheet entry has a number of values or set of (secondary) key-value pairs. For example, semester grade point average (SGPAs) values and cumulative grade point average (CGPA) value. How will the hash function be used?

SOLUTION

A hash function generates an index, I_k for k . I_k should ideally be unique and should uniquely map to k . I_k is a number with few digits only, compared to a number of characters (0-255 bytes) in the main key k used as input for the hash function. Assume that total 20 numbers of entries are present between slots indices between 00 to 99. Student name may consist of several characters, but index will be just two digits.

Hash table refers to using associated key-value pairs. A set of pairs retrieve by using a hash key. The hash key is a computed index using hash function for a column. The analytics may use the hash table. The table contains hash keys in the table-columns. The entries (values) across an array of slots (also called buckets). The buckets correspond to the key for the pairs at column. The values are in the associated rows of that column.

3.3.2 Document Store

Characteristics of Document Data Store are high performance and flexibility. Scalability varies, depends on stored contents. Complexity is low compared to tabular, object and graph data stores.

Following are the features in Document Store:

1. Document stores unstructured data.
2. Storage has similarity with object store.
3. Data stores in nested hierarchies. For example, in JSON formats data model [Example 3.3(ii)], XML document object model (DOM), or machine-readable data as one BLOB. Hierarchical information stores in a single unit called *document tree*. Logical data stores together in a unit.
4. Querying is easy. For example, using section number, sub-section number and figure caption and table headings to retrieve document partitions.
5. No object relational mapping enables easy search by following paths from the root of document tree.
6. Transactions on the document store exhibit ACID properties.

Typical uses of a document store are: (i) office documents, (ii) inventory store, (iii) forms data, (iv) document exchange and (v) document search.

The demerits in Document Store are incompatibility with SQL and complexity for implementation. Examples of Document Data Stores are CouchDB and MongoDB.

Real-life Datasets Section 10.3 will describe a very large real-life dataset for Big Data analytics as an examples. An application later analyses the structures in csv, json or other, and creates data stores in new forms (Sections 10.3.2 to 10.3.4). Runs in next step ETL, analytics or other functions. (Sections 10.4 to 10.6). This feature is called late binding (schema-on-read, or schema-on-need).

CSV and JSON File Formats CSV data store is a format for records [Example 1.9 and Example 3.3(i)]. CSV does not represent object-oriented databases or hierarchical data records. JSON and XML represent semistructured data, object-oriented records and hierarchical data records. JSON (Java Script Object Notation) refers to a language format for semistructured data. JSON represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript.

The following example explains the CSV and JSON object concept and aspects of CSV and JSON file formats.

EXAMPLE 3.3

Assume Preeti gave examination in Semester 1 in 1995 in four subjects. She gave examination in five subjects in Semester 2 and so on in each subsequent semester. Another student, Kirti gave examination in Semester 1 in 2016 in three subjects, out of which one was theory and two were practical subjects. Presume the subject names and grades awarded to them.

- (i) Write two CSV files for cumulative grade-sheets for both the students. Point the difficulty during processing of data in these two files.
- (ii) Write a file in JSON format with each student grade-sheet as an object instance. How does the object-oriented and hierarchical data record in JSON make processing easier?

SOLUTION

- (i) Two CSV file for cumulative grade-sheets are as follows:
CSV file for Preeti consists of the following nine lines each with four

columns:

Semester, Subject Code, Subject Name, Grade

1, CS101, ““Theory of Computations””, 7.8.

1, CS102,1, ““Computer Architecture””, 7.8.

.....

2, CS204, ““Object Oriented Programming””, 7.2.

2, CS205, ““Data Analytics””, 8.1.

The CSV file for Kirti consist of following five lines each with five columns: Semester, Subject Type, Subject Code, Subject Name, Grade

1, Theory, EL101, ““Analog Electronics””, 7.6.

1, Theory, EL102,1, ““Principles of Analog Communication””, 7.5.

1, Theory, EL103, ““Digital Electronics””, 7.8.

1, Practical, CS104, ““Analog ICs””, 7.2

1, Practical, CS105, ““Digital ICs””, 8.4

A column head is a key. Number of key-value pairs are $(4 \times 9) = 36$ for preetiGradeSheet.csv and $(5 \times 5) = 25$ for kirtiGradeSheet.csv. Therefore, when processing student records, merger of both files into a single file will need a program to extract the key-value pairs separately, and then prepare a single file.

- (ii) JSON gives an advantage of creating a single file with multiple instances and inheritances of an object. Consider a single JSON file, *studentGradeSheets.json* for cumulative grade-sheets of many students. *Student_Grades* object is top in the hierarchy. Each *student_name* object is next in the hierarchy with object consisting of student name, each with number of instances of subject codes, subject types, subject titles and grades awarded. Each student name object-instance extends in *student grades* object-instances.

Part of the file construct can be as follows:

```
0: {
    _id: 0,
    masterfile: "Students_Grades",
   instancetype: "single",
    mandatory: true,
    "description": "Uniquely identifies student grade master file Object
    Students_Grades "
    "resourcedefs": {
        "1": {
            _id:1,
            name: "studentName",
            instanceof: "multiple",
            "description": "Identifies a semester of the studentName and"
                "resourcedefs": {
                    "200" {
                        _id:200
                        studentName: "Kirti"
                        instanceof: "single"
                        resourcedefs: {
                            "201" {
                                _id:201
                                semester: "1",
                                subjectType: "Theory",
                                subjectCode: "EL101",
                                subjectName: "Analog Electronics"
                                Grade: 7.6
                                type: "string",
                                "description": "instance Grade for a subject Analog Electronics"
                            }
                        }
                    "202": {
                        _id:202,
                        "203" {_id:203
                        semester: "1",
                        subjectType: "Theory"
                        subjectCode: "EL102"
                        subjectName: "Principles of Analog Communication"
                        Grade: 7.5, type = "string"
                    }
                }
            {
                -----
            }
            {..
        }
    }
}
```

XML (eXtensible Markup Language) is an extensible, simple and scalable language. Its self-describing format describes structure and contents in an easy to understand format. XML is widely used. The document model consists of root element and their sub-elements. XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store and data exchanges over the network. An XML document is semi-structured.

Document store appears quite similar to a key-value store and an object store. They are complex in implementation and are SQL incompatible. They have no object-relational layer for mapping and thus enable agile development of text analytics. No sharding of data takes place into the tables. Although the values stored as documents, follows structure and encoding of the managed data

The database stores and retrieves documents, such as XML, JSON, BSON (Binary-encoded Script Object Notation (for objects)). The documents are self-describing, hierarchical tree-structured consisting of maps, collections and scalar values. The documents stored are similar to each other but do not have to be the same. Some of the popular document data stores are CouchDB, MongoDB, Terrastore, OrientDB and RavenDB.

Certain NoSQL DBs enable ACID rule-based transactions also. Examples of document data stores are MongoDB, Apache Couchbase and MarkLogic.

CouchDB uses the JSON store data, HTTP APIs for connectivity, JavaScript for the query language and MapReduce for processing.

Document JSON Format CouchDB Database Apache CouchDB is an open-source database. Its features are:

1. CouchDB provides mapping functions during querying, combining and filtering of information.
2. CouchDB deploys JSON Data Store model for documents. Each document maintains separate data and metadata (schema).
3. CouchDB is a multi-master application. Write does not require field locking when controlling the concurrency during multi-master application.
4. CouchDB querying language is JavaScript. Java script is a language which

documents use to transform.

5. CouchDB queries the indices using a web browser. CouchDB accesses the documents using HTTP API. HTTP methods are Get, Put and Delete (Section 3.3.1).
6. CouchDB data replication is the distribution model that results in fault tolerance and reliability.

Document JSON Format—MongoDB Database MongoDB Document database provides a rich query language and constructs, such as database indexes allowing easier handling of Big Data.

Example of Document in Document Store:

```
{  
    "id": "1001"  
    "Student Name":  
    {  
        "First": "Ashish",  
        "Middle": "Kumar",  
        "Last": "Rai"  
    }  
    "Category": "Student",  
    "Class": "B.Tech.",  
    "Semester": "VII",  
    "Branch": "Computer Engineering",  
    "Mobile": "12345"  
}
```

The document store allows querying the data based on the contents as well. For example, it is possible to search the document where student's first name is "Ashish". Document store can also provide the search value's exact location. The search is by using the document path. A type of key accesses the leaf values in the tree structure. Since the document stores are schema-less, adding fields to documents (XML or JSON) becomes a simple task.

Document Architecture Pattern and Discovering Hierarchical Structure
Following is example of an XML document in which a hierarchical structure discovers later. Figure 3.5 shows an XML document architecture pattern in a document fragment and document tree structure.

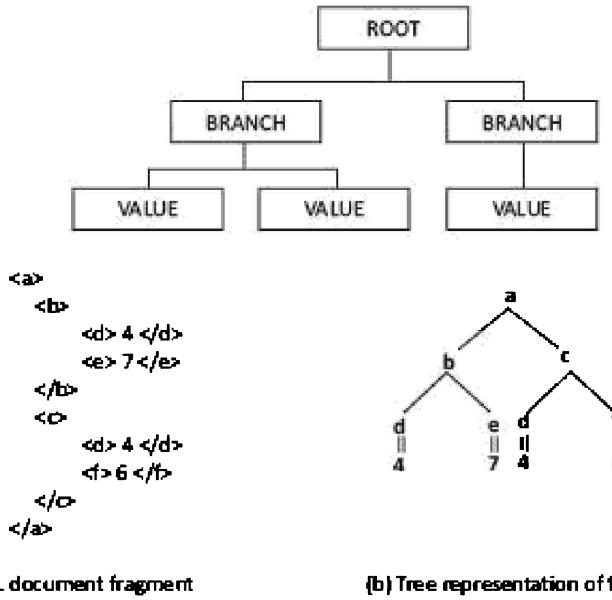


Figure 3.5 XML document architecture pattern

The document store follows a tree-like structure (similar to directory structure in file system). Beneath the root element there are multiple branches. Each branch has a related path expression that provides a way to navigate from the root to any given branch, sub-branch or value.

XQuery and XPath are query languages for finding and extracting elements and attributes from XML documents. The query commands use sub-trees and attributes of documents. The querying is similar as in SQL for databases. XPath treats XML document as a tree of nodes. XPath queries are expressed in the form of XPath expressions. Following is an example of XPath expressions:

EXAMPLE 3.4

Give examples of XPath expressions. Let outermost element of the XML document is *a*.

SOLUTION

An XPath expression */a/b/c* selects *c* elements that are children of *b* elements that are children of element *a* that forms the outermost element of the XML document.

An XPath expression */a/b[c=5]* selects elements *b* and *c* that are children of

a and value of *c* element is 5.

An XPath expression /*a*[*b*/*c*]/*d* selects elements *c* and *d* where *c* is child of *b* and *b* and *d* are children of *a*.

XML and JSON both are designed to form a simple and standard way of describing different kinds of hierarchical data structures. They are popularly used for storing and exchanging data. The following example explains the concept of Document Store in JSON and XML for hierarchical records.

EXAMPLE 3.5

Give the structures of XML and JSON document fragments for a student record.

SOLUTION

Following are the structures:

```
{                                                 <students>
  "students": [                                 <student>
    {                                         <name>Ashish Jain</name>
      "name": "Ashish Jain",                  <rollNo>12345</rollNo>
      "rollNo": "12345"                      </student>
    },                                         <student>
    {                                         <name>Sandeep Joshi</name>
      "name": "Sandeep Joshi",                <rollNo>12346</rollNo>
      "rollNo": "12346"                      </student>
    }                                         </students>
  ]
```

(a) JSON

(b) XML equivalent

When compared with XML, JSON has the following advantages:

- XML is easier to understand but XML is more verbose than JSON.
- XML is used to describe structured data and does not include arrays, whereas JSON includes arrays.
- JSON has basically key-value pairs and is easier to parse from JavaScript.

- The concise syntax of JSON for defining lists of elements makes it preferable for serialization of text format objects.

Document Collection A collection can be used in many ways for managing a large document store. Three uses of a document collection are:

1. Group the documents together, similar to a directory structure in a file-system. (A directory consists of grouping of file folders.)
2. Enables navigating through document hierarchies, logically grouping similar documents and storing business rules such as permissions, indexes and triggers (special procedure on some actions in a database).
3. A collection can contain other collections as well.

3.3.3 Tabular Data

Tabular data stores use rows and columns. Row-head field may be used as a key which access and retrieves multiple values from the successive columns in that row. The OLTP is fast on in-memory row-format data.

Oracle DBs provide both options: columnar and row format storages. Generally, relational DB store is

in-memory row-based data, in which a key in the first column of the row is at a memory address, and values in successive columns at successive memory addresses. That makes OLTP easier. All fields of a row are accessed at a time together during OLTP. Different rows are stored in different addresses in the memory or disk. In-memory row-based DB stores a row as a consecutive memory or disk entry. This strategy makes data searching and accessing faster during *transactions* processing.

In-memory column-based data has the keys (row-head keys) in the first column of each row at successive memory addresses. The next column of each row after the key has the values at successive memory addresses. The values in the third column of each row are at the next memory addresses in succession, and so on up to N columns. The N can be a very large number. The column-based data makes the OLAP easier. All fields of a column access together. All fields of a set of columns may also be accessed together during OLAP. Different rows are stored in different addresses in the memory or disk, but each row values are now not at successive addresses. In-memory column-based DB store a column as

a consecutive memory or disk entry. This strategy makes the analytics processing fast.

Following subsections describe NoSQL format data stores based on tabular formats.

3.3.3.1 Column Family Store

Columnar Data Store A way to implement a schema is the divisions into columns. Storage of each column, successive values is at the successive memory addresses. Analytics processing (AP) In-memory uses columnar storage in memory. A pair of row-head and column-head is a key-pair. The pair accesses a field in the table.

All values in successive fields in a column consisting of multiple rows save at consecutive memory addresses. This enables fast accesses during in-memory analytics, which includes CPU accesses and analyses using memory addresses in which values are cached from the disk before processing. The OLAP (on-line AP) is also fast on in-memory column-format data. An application uses a combination of row head and a column head as a key for access to the value saved at the field.

Column-Family Data Store Column-family data-store has a group of columns as a column family. A combination of row-head, column-family head and table-column head can also be a key to access a field in a column of the table during querying. Combination of row head, column families head, column-family head and column head for values in column fields can also be a key to access fields of a column. A column-family head is also called a super-column head.

Examples of columnar family data stores are HBase, BigTable, HyperTable and Cassandra. The following example explains a column-family data store and why OLAP is fast in-memory column data store in memory:

EXAMPLE 3.6

Consider Example 1.6(i). Assume in-memory columnar storage. Data for a large number of ACVMs with an ACVM_ID each, store in column 1. Data for each day sales at each ACVM for KitKat, Milk, Fruit and Nuts, Nougat and Oreo store in Columns 2 to 6. Each row has six cells (ID +five sales data).

- (i) How do the column key values store in memory?

- (ii) How do the values store in the memory in columnar storage format?
- (iii) How does analytics of each day's sales help?
- (iv) Why do in-memory columnar storage result in fast computations during analytics?
- (v) How are a column family and column-family head (key) specified?
- (vi) How do a column-families group specify?
- (vii) How do row groups form? What is the advantage of division into sub-groups?

SOLUTION

Assume the following columnar storage at memory:

- (i) Column and row keys

Addresses 1000, 2000, 3000,, 6000 save the column keys. Address 1000 stores string 'ACVM_ID'. Then the addresses 1001, 1002,, 1999 store the row keys, means ACVM_IDs.

Chocolate name of five flavours store at addresses 2000, 3000, 4000, 5000, and 6000.

- (ii) Column field values

Column 1 ACVM_IDs store at address 1001 to 1999 for 999 ACVMs. Sales in a day for KitKat, Milk, Fruit and Nuts, Nougat and Oreo store at addresses 2001 to 2999, 3001 to 3999, 4001 to 4999, 5001 to 5999, and 6001 to 6999.

Table 3.3 gives sample values in the columns for a day's sales data. The table also gives the keys for row groups, rows (ACVM_IDs), a column family group, two column families and five column heads for 5 flavours of chocolates. The table gives a row group of just 100 rows, just for the sake of assumption.

Figure 3.6 shows fields in columnar storage and addresses in memory. The figure shows ACVM_IDs as well as each day's sales of each flavour of chocolate at 999 ACVMs. Following are the addresses assigned to the

values in fields of Table 3.3:

Table 3.3 Each day's sales of chocolates on 999 ACVMs

ACVM_ID		Nestle Chocolate Flavours Group					
		Popular Flavours Family			Costly Flavours Family		
		KitKat	Milk	Fruit and Nuts	Nougat	Oreo	
Row-group_1 for IDs 1 to 100		1	360	150	500	101	222
		2	289	175	457	145	317
	
Row-group_m for IDs 901 to 999	
		998	123	201	385	199	310
		999	75	215	560	108	250

Field Value	ACVM_ID	1	2	998	999	Kitkat	360	289	123	75	Milk	150	175	
Address	1000	1001	1002	1998	1999	2000	2001	2002	2998	2999	3000	3001	3002	
<hr/>																
....	201	215	Fruit and Nuts	500	457	385	560	Nougat	101	145	199	108	Oreo	
....	3998	3999	4000	4001	4002	4999	4999	5000	5001	5002	5998	5999	6000	
....	6001	6002	<hr/>													
....	310	250	<hr/>													
....	6998	6999	<hr/>													
<hr/>																
Family1				Family2												
800				801												
<hr/>																
ColumnFamilyGroup 1								7000								

Figure 3.6 Fields in columnar storage and addresses in memory.

- (iii) An analytics application computes the results, such as (a) total sales of each flavour, KitKat, Milk, Fruit and Nuts, Nougat and Oreo, each day, (b) Each ACVM requirement for refilling at each ACVM each day, (c) ID of maximum sales of chocolates each day, (d) cluster of ACVMs showing highest sales, classification of ACVMs as low, moderate and high sales.
- (iv) Consider first address of sales data for KitKat, address_{0, kk} = 1001 of machine ID, ACVMId at address₀ = 1000. Total sales at all 999 ACVMs in a day requires the sum of values between address 1001 to 1999. Increment to the next address is fast when compared to the case when during the execution the value addresses compute from a table of pointers for them. When values are in the row storage format, the chocolate KitKat sales data at the machines will be at addresses 1001, 1007, 1013, ... The table of pointers or computations of address_{0, kk} + n × N + 1, where N is number of columns for each row, is required, and n = 0, 1, 2, ..., 998, 999. The processing takes longer compared to instruction for increment of pointed address to next memory address. Therefore, analytics of (a), (b), (c) and (d) is quicker fast in case of In-memory columnar storage compared to row format storage in memory.
- (v) Columns in Table 3.3 for KitKat and Milk form a group as one family. Columns for Fruit and Nuts, Nougat, and Oreo form a group as second family. The key for one family is 'Popular Flavours Family' and second family is 'Costly Flavours Family'. The keys of column families can save at the addresses 800, 801, ...
- (vi) Two column-families in Table 3.3, Popular Flavours Family and Costly Flavours Family form a super group, 'Nestle Chocolate Flavours'. The key for super group of column-families group is 'Nestle Chocolate Flavours'. The keys of column-family super groups can save at the addresses 700, 701, 702, ...
- (vii) A set of fields in all column families for ACVMs, say of IDs 1 to 100 can

be grouped into row-group_1. Number of row-groups can then be processed as separate sub-tables, parallelly in Big Data environment. The keys of row groups can save at the addresses 600, 601, 602, ...

Columns Families Two or more columns in data-store group into one column family. Table 3.3 considered two families.

Sparse Column Fields A row may associate a large number of columns but contains values in few column fields. Similarly, many column fields may not have data. Columns are logically grouped into column families. Column-family data stores are then similar to sparse matrix data. Most elements of sparse matrix are empty. Data stores at memory addresses is columnar-family based rather than as row based. Metadata provide the column-family indices of not empty column fields.

That facilitates OLAP of not empty column families faster. For example, assume hash key in a column heading field and values in successive rows at one column family. For another key, the values will be in another column family.

Grouping of Column Families Two or more column-families in data store form a super group, called super column. Table 3.3 consists of one such group (super column), 'Nestle Chocolate Flavours Group'.

Grouping into Rows When number of rows are very large then horizontal partitioning of the table is a necessity. Each partition forms one row-group. For example, a group of 1 million rows per partition. A row group thus has all column data store in the memory for in-memory analytics. Practically, row groups are chosen such that memory required for the group is above, say 10 MB and below the maximum size which can cached and buffered in memory, say 1 GB for in-memory analytics.

Data caching, buffering in memory and storing back at disk takes time. So frequent disk accesses remain controlled. Therefore, minimum row-group size of 10 MB is practical (Table 3.3 considered a row group of just 100 rows for the purpose of explaining the addressing and use of keys in a columnar-family data store).

Characteristics of Columnar Family Data Store Columnar family data store imbibes characteristics of very high performance and scalability, moderate level

of flexibility and lower complexity when compared to the object and graph databases. Advantages of column stores are:

1. *Scalability*: The database uses row IDs and column names to locate a column and values at the column fields. The interface for the fields is simple. The back-end system can distribute queries over a large number of processing nodes without performing any Join operations. The retrieval of data from the distributed node can be least complicated by an intelligent plan of row IDs and columns, thereby increasing performance. Scalability means addition of number of rows as the number of ACVMs increase in Example 1.6(i). Number of processing instructions is proportional to the number of ACVMs due to scalable operations.
2. *Partitionability*: For example, large data of ACVMs can be partitioned into datasets of size, say 1 MB in the number of row-groups. Values in columns of each row-group, process in-memory at a partition. Values in columns of each row-group independently parallelly process in-memory at the partitioned nodes.
3. *Availability*: The cost of replication is lower since the system scales on distributed nodes efficiently. The lack of Join operations enables storing a part of a column- family matrix on remote computers. Thus, the data is always available in case of failure of any node.
4. *Tree-like columnar structure* consisting of column-family groups, column families and columns. The columns group into families. The column families group into column groups (super columns). A key for the column fields consists of three secondary keys: column-families group ID, column-family ID and column-head name.
5. *Adding new data at ease*: Permits new column *Insert* operations. Trigger operation creates new columns on an Insert. The column-field values can add after the last address in memory if the column structure is known in advance. New row-head field, row-group ID field, column-family group, column family and column names can be created at any time to add new data.

6. *Querying all the field values* in a column in a family, all columns in the family or a group of column-families, is fast in in-memory column-family data store.
7. *Replication of columns*: HDFS-compatible column-family data stores replicate each data store with default replication factor = 3.
8. *No optimization for Join*: Column-family data stores are similar to sparse matrix data. The data do not optimize for Join operations.

Column-family data store in a format in which store set of column family field-values which are not empty (null or zero). Metadata of the matrix consists of hash keys that reference each set distinctly.

Typical uses of column store are: (i) web crawling, (ii) large sparsely populated tables and (iii) system that has high variance.

HDFS is highly reliable for very long running queries. However, IO operations are slow. Columnar storage is a solution for faster IOs. Columnar storage in memory stores the data actually required for the IOs. Only columns needing the access load during execution. Also, a columnar-object data store can be compressed or encoded. The encoding is according to the data type. Also, the executions of different columns or column partitions can be in parallel at the cluster data-nodes.

3.3.3.2 BigTable Data Store

Examples of widely used column-family data store are Google's BigTable, HBase and Cassandra. Keys for *row key*, *column key*, *timestamp* and *attribute* uniquely identify the values in the fields (Refer Example 2.4)

Following are features of a BigTable:

1. Massively scalable NoSQL. BigTable scales up to 100s of petabytes.
2. Integrates easily with Hadoop and Hadoop compatible systems.
3. Compatibility with MapReduce, HBase APIs which are open-source Big Data platforms.
4. Key for a field uses not only row_ID and Column_ID (for example, ACVM_ID and KitKat in Example 3.6) but also timestamp and attributes. Values are ordered bytes. Therefore, multiple versions of values may be

- present in the BigTable.
5. Handles million of operations per second.
 6. Handle large workloads with low latency and high throughput
 7. Consistent low latency and high throughput
 8. APIs include security and permissions
 9. BigTable, being Google's cloud service, has global availability and its service is seamless.

The following example explains the use of rowID, ColumID and Column attributes in BigTable formats.

EXAMPLE 3.7

Consider Example 3.6. Consider column fields which have keys to access a field not only by row ID and Column ID but also include the timestamp and attributes in a row. Show the column-keys for accessing column fields of a column.

SOLUTION

Table 3.4 gives keys for each day's sales of KitKat chocolates at ACVMs. First row-headings are the column-keys.

Table 3.4 Each day's sales of KitKat chocolates at ACVMs



Column-keys	ACVM_ID	KitKatSalesDate	Timestamp	KitKatSalesNumber

3.3.3.3 RC File Format

Hive uses Record Columnar (RC) file-format records for querying. RC is the best choice for intermediate tables for fast column-family store in HDFS with Hive. Serializability of RC table column data is the advantage. RC file is DeSerializable into column data. A table such as that shown in Example 3.6 can be partitioned into row groups. Values at each column of a row group store as the RC record.

The RC file records store data of a column in the row group (*Serializability* means query or transaction executable by series of instructions such that execution ensures correct results).

The following example explains the use of row groups in the RC file format for column of a row group:

EXAMPLE 3.8

Consider Example 3.6. Practically, row groups have millions of rows and in-memory between 10 MB and 1 GB. Assume two row groups of just two rows each. Consider the following values given in Table 3.3.

Row-group_1 for IDs 1 to 2					
1	360	150	500	101	222
2	289	175	457	145	317
Row-group_m for IDs 998 to 999					
998	123	201	385	199	310
999	75	215	560	108	250

Make a file in RC format.

SOLUTION

The values in each column are the records in file for each row group. Each row-group data is like a column of records which stores in the RC file.

Row group_1	Row group_m
1, 2;	998, 999;
360, 289;	123, 75;
....	...
....	...
....	...
222, 317;	310, 250;

ACVM_ID
 ↪
 KitKat
 ↪
 Milk
 ↪
 Fruit and Nuts
 ↪
 Nougat
 ↪
 Oreo

RC file for row group_1 will consists of records 1, 2; 360, 289; ..., 222, 317; on serialization of column records. RC file for row group_m will consists of

998, 999; 123, 75; ..., 310, 250;

3.3.3.4 ORC File Format

An ORC (Optimized Row Columnar) file consists of row-group data called stripes. ORC enables concurrent reads of the same file using separate RecordReaders. Metadata store uses Protocol Buffers for addition and removal of fields.¹

ORC is an intelligent Big Data file format for HDFS and Hive.² An ORC file stores a collections of rows as a row-group. Each row-group data store in columnar format. This enables parallel processing of multiple row-groups in an HDFS cluster.

An ORC file consists of a stripe the size of the file is by default 256 MB. Stripe consists of indexing (mapping) data in 8 columns, row-group columns data (contents) and stripe footer (metadata). An ORC has two sets of columns data instead of one column data in RC. One column is for each map or list size and other values which enable a query to decide skipping or reading of the mapped columns. A mapped column has contents required by the query. The columnar layout in each ORC file thus, optimizes for compression and enables skipping of data in columns. This reduces read and decompression load.

Lightweight indexing is an ORC feature. Those blocks of rows which do not match a query skip as they do not map on using indices data at metadata. Each index includes the aggregated values of minimum, maximum, sum and count using aggregation functions on the content columns. Therefore, contents-column key for accessing the contents from a column consists of combination of row-group key, column mapping key, min, max, count (number) of column fields of the contents column. Table 3.5 gives the keys used to access or skip a contents column during querying. The keys are Stripe_ID, Index-column key, and contents-column name, min, max and count.

Table 3.5 Keys to access or skip a content column in ORC file format

Stripe_ID	Index Column 1				Index Column 2
	Index column 1 key 1				Index column 2 key 1
	Contents-Column name	Contents Minimum value	Contents Maximum value	Count (number) of content-column fields	
	
	
	Index column 1 key 2				Index column 2 key 2
	Column-name	Minimum value	Maximum value	Count of number of column fields	
	
	

Consider Example 3.6. ORC key to access during a query consist of not only column head ‘KitKat’ (Table 3.3) but also column minimum and maximum sales on an ACVM, count of number of fields in values ‘KitKat’. Analytics operations frequently need these values. Ready availability of these values from the index data itself improves the throughput in Big Data HDFS environment. These values do not need to compute again and again using aggregation functions, such as min, max and count.

An ORC thus, optimizes for reading serially the column fields in HDFS environment. The throughput increases due to skipping and reading of the required fields at contents-column key. Reading less number of ORC file content-columns reduces the workload on the NameNode.

3.3.3.5 Parquet File Formats

Parquet is nested hierarchical columnar-storage concept. Nesting sequence is the table, row group, column chunk and chunk page. Apache Parquet file is columnar-family store file. Apache Spark SQL executes user defined functions (UDFs) which query the Parquet file columns (Section 5.2.1.3). A programmer writes the codes for an UDF and creates the processing function for big long queries.

A Parquet file uses an HDFS block. The block stores the file for processing queries on Big Data. The file compulsorily consists of metadata, though the file need not consist of data.

The Parquet file consists of row groups. A row-group columns data process in-

memory after data cache and buffer at the memory from the disk. Each row group has a number of columns. A row group has N_{col} columns, and row group consists of N_{col} column chunks. This means each column chunk consists of values saved in each column of each row group.

A column chunk can be divided into pages and thus, consists of one or more pages. The column chunk consists of a number of interleaved pages, N_{pg} . A page is a conceptualized unit which can be compressed or encoded together at an instance. The unit is minimum portion of a chunk which is read at an instance for in-memory analytics.

An ORC array <int> has two columns, one for array size and the other for contents. Parquet format file does not consist of extra column per nesting level. Similarly, ORC has two columns, one is for each Map, List size, min, max and the second is for the contents. Parquet format file does not consist of extra column per nesting level, just one column per leaf in the schema.

[Parquet in English means ‘a floor covering made of small rectangular wooden blocks (tiles) fitted together in a pattern. Similarly, Parquet objects have pages as the tiles. Pages build a column chunk. Column chunks build a row group. Row groups build the table. A page is like a tile consisting of column fields. The values read or write at an instance or used for encoding or compression. The values are not read separately from a page.]

Table 3.6 gives the keys used to access or skip the contents page. Three keys are: (i) row-group_ID, (ii) column-chunk key and (iii) page key.

Table 3.6 Combination of keys for content page in the Parquet file format

Row-group_ID	Column Chunk 1 key			
	Page 1 key	Page 2 key	***	Page m key
	**	***	***	***
	**	***	***	***
Row-group_ID	Column Chunk 2 key			
	Page 1key	Page 2 key	***	Page key m'
	**	***	***	***
	**	***	***	***

3.3.4 Object Data Store

An object store refers to a repository which stores the:

1. Objects (such as files, images, documents, folders, and business reports)
2. System metadata which provides information such as filename, creation_date, last_modified, language_used (such as Java, C, C#, C++, Smalltalk, Python), access_permissions, supported query languages)
3. Custom metadata which provides information, such as subject, category, sharing permissions.

Metadata enables the gathering of metrics of objects, searches, finds the contents and specifies the objects in an object data-store tree. Metadata finds the relationships among the objects, maps the object relations and trends. Object Store metadata interfaces with the Big Data. API first mines the metadata to enable mining of the trends and analytics. The metadata defines classes and properties of the objects. Each Object Store may consist of a database. Document content can be stored in either the object store database storage area or in a file storage area. A single file domain may contain multiple Object Stores.

Data definition and manipulation, DB schema design, database browsing, DB administration, application compilation and debugging use a programming language.

Eleven Functions Supporting APIs An Object data store consists of functions supporting APIs for: (i) scalability, (ii) indexing, (iii) large collections, (iv) querying language, processing and optimization (s), (v) Transactions, (vi) data replication for high availability, data distribution model, data integration (such as with relational database, XML, custom code), (vii) schema evolution, (viii) persistency, (ix) persistent object life cycle, (x) adding modules and (xi) locking and caching strategy.

Object Store may support versioning for collaboration. Object Store can be created using IBM ‘Content Platform Engine’. Creation needs installing and configuring the engine (Engine is software which drives forward.). Console of the engine makes creation of process easy. Amazon S3 and Microsoft Azure BLOB support the Object Store.

Amazon S3 (Simple Storage Service) S3 refers to Amazon web service on the cloud named S3. The S3 provides the Object Store. The Object Store differs from the block and file-based cloud storage. Objects along with their metadata store

for each object store as the files. S3 assigns an ID number for each stored object. The service has two storage classes: Standard and infrequent access. Interfaces for S3 service are REST, SOAP and Bit Torrent. S3 uses include web hosting, image hosting and storage for backup systems. S3 is scalable storage infrastructure, same as used in Amazon e-commerce service. S3 may store trillions of objects.

The following example lists Object Store development platforms:

EXAMPLE 3.9

List the functions of Minio, Riak, VERSANT Object Database (VOD), GEMSTONE, Amazon S3 and Microsoft Azure BLOB that support using Object Store APIs.

SOLUTION

1. An open-source multi-clouds object storage server is Minio, which is API compatible with Amazon S3 API and number of widely used public and private clouds. Compatibility enables data export to S3 and usages of APIs.
2. Riak CS (Cloud Storage) is object storage management software on top of Riak. It models on open-source distributed-database which is Amazon-compliant. This means database exports to S3 and use S3 APIs.
3. VOD consists of 11 functions supporting APIs listed above. VOD enables use by multiple concurrent users. VOD supports cross-platform operating systems (OSs), such as Linux, Windows NT, AIX, HP-UX and Solaris (both 32 and 64 bits for all platforms).
4. GEMSTONE Object DB APIs development language is SmallTalk. The platform supports in-memory DBs, object-oriented processing and distributed caches. GEMSTONE provides cross platform support, OSs AIX, Linux, MacOS and Solaris.

3.3.4.1 Object Relational Mapping

The following example explains object relational mapping.

EXAMPLE 3.10

How does an HTML object and XML based web service relate with tabular data stores?

SOLUTION

Figure 3.7 shows the object relational mapping of HTML document and XML web services store with a tabular data store.

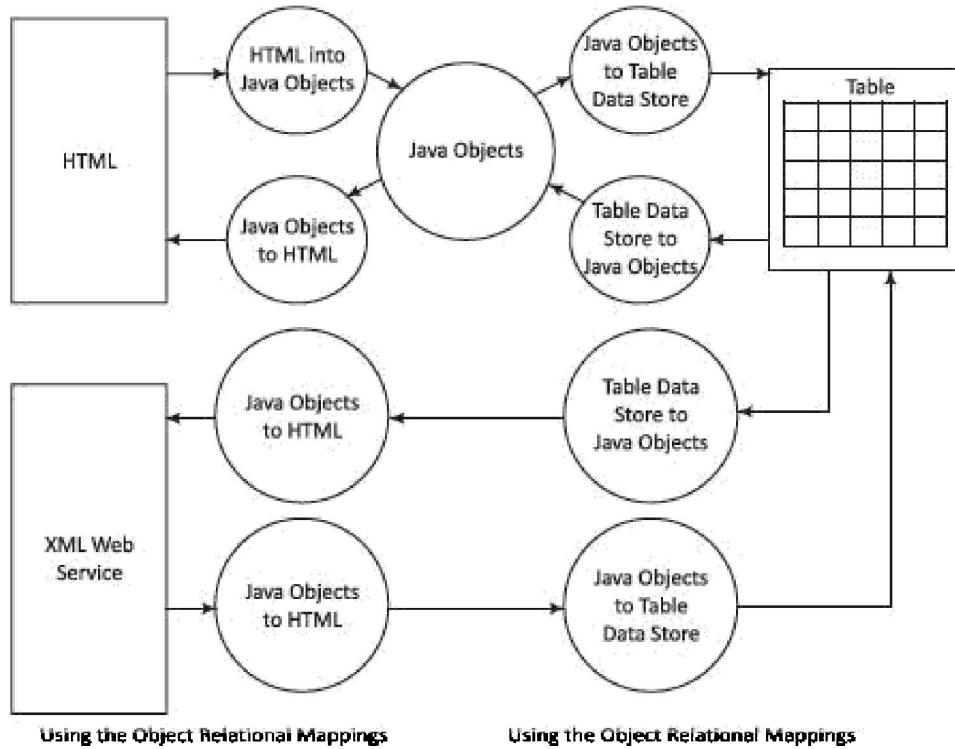


Figure 3.7 HTML document and XML web services

3.3.5 Graph Database

One way to implement a data store is to use graph database. A characteristic of graph is high flexibility. Any number of nodes and any number of edges can be added to expand a graph. The complexity is high and the performance is variable with scalability. Data store as series of interconnected nodes. Graph with data nodes interconnected provides one of the best database system when

relationships and relationship types have critical values.

Data Store focuses on modeling *interconnected* structure of data. Data stores based on graph theory relation $G = (E, V)$, where E is set of edges e_1, e_2, \dots and V is set of vertices, v_1, v_2, \dots, v_n .

Nodes represent entities or objects. Edges encode relationships between nodes. Some operations become simpler to perform using graph models. Examples of graph model usages are social networks of connected people. The connections to related persons become easier to model when using the graph model.

The following example explains the graph database application in describing entities relationships and relationship types.

EXAMPLE 3.11

Let us assume a car company represents a node entity, which has two connected nodes comprising two model entities, namely Hexa and Zest. Draw graph with directed lines, joining the car company with two entities. (i) How do four directed lines relate to four weeks and two directed lines? One directed line corresponds to a car model. Only directed line corresponds to weekly total sales. (ii) How will the yearly sales compute? (iii) Show the path traversals for computations exhibit BASE properties.

SOLUTION

- (i) Figure 3.8 shows section of a graph database for the sales of two car models.

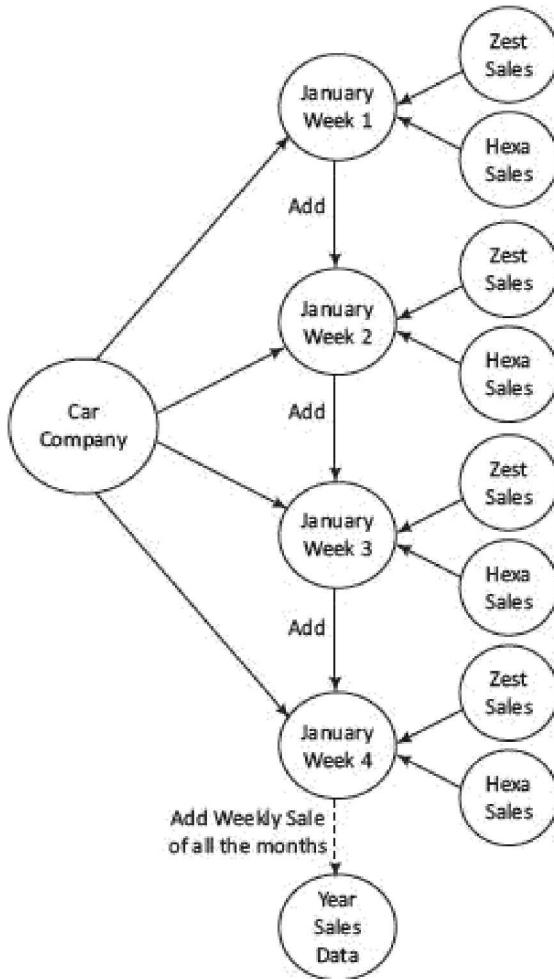


Figure 3.8 Section of the graph database for car-model sales

- (ii) The yearly sales compute by path traversals from nodes for weekly sales to yearly sales data.
- (iv) The path traversals exhibit BASE properties because during the intermediate paths, consistency is not maintained. Eventually when all the path traversals complete, the data becomes consistent.

Graph databases enable fast network searches. Graph uses linked datasets, such as social media data. Data store uses graphs with nodes and edges connecting each other through relations, associations and properties.

Querying for data uses graph traversal along the paths. Traversal may use

single-step, path expressions or full recursion. A relationship represents key. A node possesses property including ID. An edge may have a label which may specify a role.

Characteristics of graph databases are:

1. Use specialized query languages, such as RDF uses SPARQL
2. Create a database system which models the data in a completely different way than the key-values, document, columnar and object data store models.
3. Can have hyper-edges. A hyper-edge is a set of vertices of a hypergraph. A hypergraph is a generalization of a graph in which an edge can join any number of vertices (not only the neighbouring vertices).
4. Consists of a collection of small data size records, which have complex interactions between graph-nodes and hypergraph nodes. Nodes represent the entities or objects. Nodes use Joins. Node identification can use URI or other tree-based structure. The edge encodes a relationship between the nodes.

When a new relationship adds in RDBMS, then the schema changes. The data need transfer from one field to another. The task of adding relations in graph database is simpler. The nodes assign internal identifiers to the nodes and use these identifiers to join the network. Traversing the joins or relationships is fast in graph databases. It is due to the simpler form of graph nodes. The graph data may be kept in RAM only. The relationship between nodes is consistent in a graph store.

Graph databases have poor scalability. They are difficult to scale out on multiple servers. This is due to the close connectivity feature of each node in the graph. Data can be replicated on multiple servers to enhance read and the query processing performance. Write operations to multiple servers and graph queries that span multiple nodes, can be complex to implement.

Typical uses of graph databases are: (i) link analysis, (ii) friend of friend queries, (iii) Rules and inference, (iv) rule induction and (v) Pattern matching. Link analysis is needed to perform searches and look for patterns and relationships in situations, such as social networking, telephone, or email

records (Sections 9.4 and 9.5). Rules and inference are used to run queries on complex structures such as class libraries, taxonomies and rule-based systems.

Examples of graph DBs are Neo4J, AllegroGraph, HyperGraph, Infinite Graph, Titan and FlockDB. Neo4J graph database enable easy usages by Java developers. Neo4J can be designed fully ACID rules compliant. Design consists of adding additional path traversal in between the transactions such that data consistency is maintained and the transactions exhibit ACID properties.

Spark provides a simple and expressive programming model that includes supports to a wide range of applications, including graph computation. Chapter 8 describes Graph Databases.

3.3.6 Variations of NoSQL Architectural Patterns

Six data architectures are SQL-table, key-value pairs, in-memory column-family, document, graph and object. Selected architecture may need variations due to business requirements. Business requirements are ease of using an architecture and long-term competitive advantage. The following example explains the requirements for the database of students of a University that offers multiple courses in their various academic programmes for several years:

EXAMPLE 3.12

List the selection requirements for the database of University students in successive years. The University runs various Under Graduate and Post Graduate programmes. Students are registered to Multiple courses in a programme.

SOLUTION

Following are the selection requirements:

1. Scalability: Since the University archives the data for several years, data store should be scalable.
2. Search ability: Search of required information needs to be fast.
3. Quarrying ability: All applications need to query the data. Query retrieves the required data among the Big Data of several years.
4. Security: Database needs security and fault tolerance.

5. Affordability: Open source is a requirement.
6. Interoperability: Needs ease in search from different platforms. Search from any computer operating system, such as Windows, Mac, Linux, Android and iOS should be feasible.
7. Importability: Database needs to import data from other platforms, such as import of slides, video lectures, tutorials, e-books, webinars should be facilitated in store.
8. Transformability: Queries may be written in one language and may require transformation to another language, such as HTML.

Analysis of the above requirements suggests the document architecture pattern will be more suitable.

Kelly-McCreary, co-founder of ‘NoSQL Now’ suggested that when selecting a NoSQL-pattern, the pattern may need change and require variation to another pattern(s). Some reasons for this are:

1. Focus changing from performance to scalability
2. Changing from modifiability to agility
3. Greater emphasis on Big Data, affording capacity, availability of support, ability for searching and monitoring the actions

Steps for selecting a NoSQL data architectural pattern can be as follows:

1. Select an architecture
2. Perform a use-case driven difficulty analysis for each of the six architectural patterns. Difficulties may be low, medium or high in the following processes: (i) ingestion, (ii) validation of structure and its fields, (iii) updating process using batch or record by record approach, (iv) searching process using full text or by changing the sorting order, and (v) export the reports or application results in HTML, XML or JSON.
3. Estimate the total efforts for each architecture for all business requirements.

Process the choice of architecture using trade-off. For example, between the

MongoDB document data store and Cassandra column-family data store.

Self-Assessment Exercise linked to LO 3.2

1. Compare traditional relational model and key-value pairs model.
2. When will you use the document data store?
3. Why is metadata must in a NoSQL Data Store?
4. How do interactions among graph nodes and hypergraph nodes differentiate?
5. List and compare the features of BigTable, RC, ORC and Parquet data stores.
6. What are the characteristics of the object data store model?
7. Data architecture pattern can be selected from among the six architectures, namely relational SQL table, OLAP-suitable in-memory column, key-value pairs, column-family, document and graph DBs. Explain with an example, how and when each of these is used.

3.4 | NOSQL TO MANAGE BIG DATA

The following subsections describe how to use a NoSQL data store to manage Big Data.

LO 3.3

NoSQL data store management, applications and handling problems in Big Data

3.4.1 Using NoSQL to Manage Big Data

NoSQL (i) limits the support for Join queries, supports sparse matrix like columnar-family, (ii) characteristics of easy creation and high processing speed, scalability and storability of much higher magnitude of data (terabytes and petabytes).

NoSQL sacrifices the support of ACID properties, and instead supports CAP and BASE properties (Sections 3.2.1.1 and 3.2.3). NoSQL data processing scales horizontally as well vertically.

3.4.1.1 NoSQL Solutions for Big Data

Big Data solution needs scalable storage of terabytes and petabytes, dropping of support for database Joins, and storing data differently on several distributed servers (data nodes) together as a cluster. A solution, such as CouchDB, DynamoDB, MongoDB or Cassandra follow CAP theorem (with compromising the consistency factor) to make transactions faster and easier to scale. A solution must also be partitioning tolerant.

Characteristics of Big Data NoSQL solution are:

1. *High and easy scalability:* NoSQL data stores are designed to expand horizontally. Horizontal scaling means that scaling out by adding more machines as data nodes (servers) into the pool of resources (processing, memory, network connections). The design scales out using multi-utility cloud services.
2. *Support to replication:* Multiple copies of data store across multiple nodes of a cluster. This ensures high availability, partition, reliability and fault tolerance.
3. *Distributable:* Big Data solutions permit sharding and distributing of shards on multiple clusters which enhances performance and throughput.
4. *Usages of NoSQL servers:* which are less expensive. NoSQL data stores require less management efforts. It supports many features like automatic repair, easier data distribution and simpler data models that makes database administrator (DBA) and tuning requirements less stringent.
5. *Usages of open-source tools:* NoSQL data stores are cheap and open source. Database implementation is easy and typically uses cheap servers to manage the exploding data and transaction while RDBMS databases are expensive and use big servers and storage systems. So, cost per gigabyte data store and processing of that data can be many times less than the cost of RDBMS.
6. *Support to schema-less data model:* NoSQL data store is schema less, so data can be inserted in a NoSQL data store without any predefined schema. So, the format or data model can be changed any time, without disruption of

application. Managing the changes is a difficult problem in SQL.

7. *Support to integrated caching:* NoSQL data store support the caching in system memory. That increases output performance. SQL database needs a separate infrastructure for that.
8. *No inflexibility* unlike the SQL/RDBMS, NoSQL DBs are flexible (not rigid) and have no structured way of storing and manipulating data. SQL stores in the form of tables consisting of rows and columns. NoSQL data stores have flexibility in following ACID rules.

3.4.1.2 Types of Big Data Problems

Big Data problems arise due to limitations of NoSQL and other DBs. The following types of problems are faced using Big Data solutions.

1. Big Data need the scalable storage and use of distributed servers together as a cluster. Therefore, the solutions must drop support for the database Joins
2. NoSQL database is open source and that is its greatest strength but at the same time its greatest weakness also because there are not many defined standards for NoSQL data stores. Hence, no two NoSQL data stores are equal. For example:
 - (i) No stored procedures in MongoDB (NoSQL data store)
 - (ii) GUI mode tools to access the data store are not available in the market
 - (iii) Lack of standardization
 - (iv) NoSQL data stores sacrifice ACID compliancy for flexibility and processing speed.

A comparison of NoSQL with SQL/RDBMS shows that NoSQL data model are schema-less, no pre-defined schema, multiple data architecture patterns, complex to implement vertical scalability, variable consistency and very weak adherence to ACID rules. Table 3.7 gives a comparison.

Table 3.7 Comparison of NoSQL with SQL/RDBMS

Features	NoSQL Data store	SQL/RDBMS

Model	Schema-less model	Relational
Schema	Dynamic schema	Predefined
Types of data architecture patterns	Key/value based, column-family based, document based, graph based, object based	Table based
Scalable	Horizontally scalable	Vertically scalable
Use of SQL	No	Yes
Dataset size preference	Prefers large datasets	Large dataset not preferred
Consistency	Variable	Strong
Vendor support	Open source	Strong
ACID properties	May not support, instead follows Brewer's CAP theorem or BASE properties	Strictly follows

Self-Assessment Exercise linked to LO 3.3

1. Why does Big Data need scalable storage and uses distributed servers together as a cluster?
2. Why does Big Data solution possess CAP or BASE and may drop support for ACID properties?
3. Why does a Big Data solution drop support for the database Joins?
4. Compare NoSQL data stores with SQL databases.

3.5 | SHARED-NOTHING ARCHITECTURE FOR BIG DATA TASKS

The columns of two tables relate by a relationship. A relational algebraic equation specifies the relation. Keys share between two or more SQL tables in RDBMS. Shared nothing (SN) is a cluster architecture. A node does not share data with any other node.

Big Data store consists of SN architecture. Big Data store, therefore, easily partitions into shards. A partition processes the different queries on data of the different users at each node independently. Thus, data processes run in parallel at the nodes. A node maintains a copy of running-process data. A coordination protocol controls the processing at all SN nodes. An SN architecture optimizes massive parallel data processing.

Shared-nothing architecture, choosing a distribution model, master-slave versus peer-to-peer, and knowledge of four ways by which NoSQL handles the Big Data problems

Data of different data stores partition among the number of nodes (assigning different computers to deal with different users or queries). Processing may require every node to maintain its own copy of the application's data, using a coordination protocol. Examples are using the partitioning and processing are Hadoop, Flink and Spark.

The features of SN architecture are as follows:

1. *Independence*: Each node with no memory sharing; thus possesses computational self-sufficiency
2. *Self-Healing*: A link failure causes creation of another link
3. *Each node functioning as a shard*: Each node stores a shard (a partition of large DBs)
4. No network contention.

3.5.1 Choosing the Distribution Models

Big Data requires distribution on multiple data nodes at clusters. Distributed software components give advantage of parallel processing; thus providing horizontal scalability. Distribution gives (i) ability to handle large-sized data, and (ii) processing of many read and write operations simultaneously in an application. A resource manager manages, allocates, and schedules the resources of each processor, memory and network connection. Distribution increases the availability when a network slows or link fails. Four models for distribution of the data store are given below:

3.5.1.1 Single Server Model

Simplest distribution option for NoSQL data store and access is Single Server Distribution (SSD) of an application. A graph database processes the relationships between nodes at a server. The SSD model suits well for graph DBs. Aggregates of datasets may be key-value, column-family or BigTable data stores which require sequential processing. These data stores also use the SSD model. An application executes the data sequentially on a single server. Figure 3.9(a) shows the SSD model. Process and datasets distribute to a single server which runs the application.

3.5.1.2 Sharding Very Large Databases

Figure 3.9(b) shows sharding of very large datasets into four divisions, each running the application on four i, j, k and l different servers at the cluster. DB_i, DB_j, DB_k and DB_l are four shards.

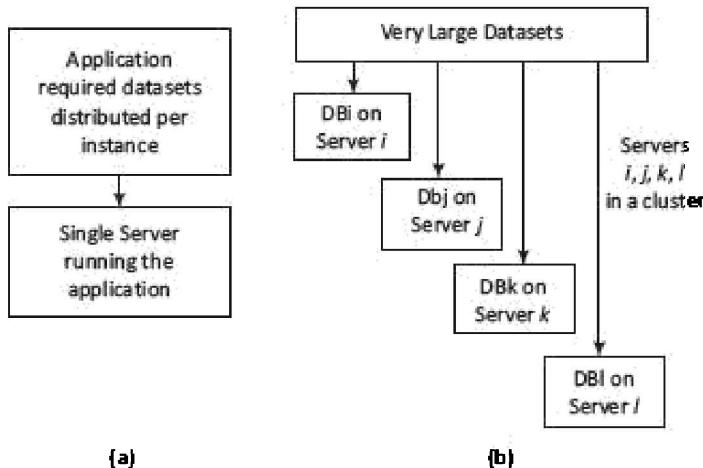


Figure 3.9 (a) Single server model **(b)** Shards distributed on four servers in a cluster.

The application programming model in SN architecture is such that an application process runs on multiple shards in parallel. Sharding provides horizontal scalability. A data store may add an auto-sharding feature. The performance improves in the SN. However, in case of a link failure with the application, the application can migrate the shard DB to another node.

3.5.1.3 Master-Slave Distribution Model

A node serves as a master or primary node and the other nodes are slave nodes. Master directs the slaves. Slave nodes data replicate on multiple slave servers in

Master Slave Distribution (MSD) model. When a process updates the master, it updates the slaves also. A process uses the slaves for read operations. Processing performance improves when process runs large datasets distributed onto the slave nodes. Figure 3.10 shows an example of MongoDB. MongoDB database server is *mongod* and the client is *mongo*.

Master-Slave Replication Processing performance decreases due to replication in MSD distribution model. Resilience for read operations is high, which means if in case data is not available from a slave node, then it becomes available from the replicated nodes. Master uses the distinct write and read paths.

Complexity Cluster-based processing has greater complexity than the other architectures. Consistency can also be affected in case of problem of significant time taken for updating.

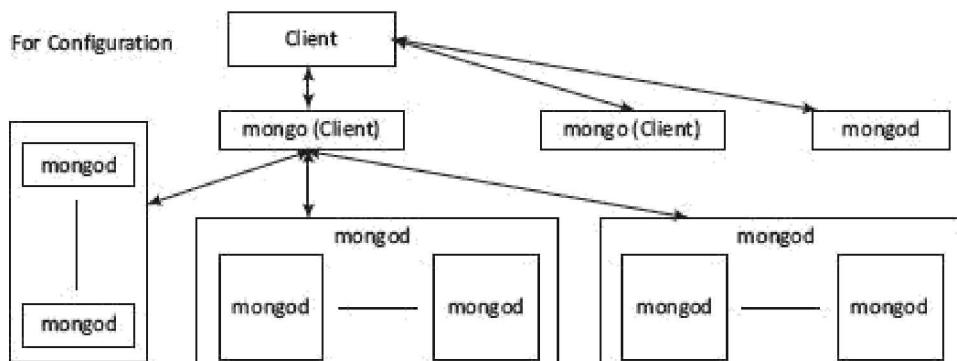


Figure 3.10 Master-slave distribution model. Mongo is a client and mongod is the server

3.5.1.4 Peer-to-Peer Distribution Model

Peer-to-Peer distribution (PPD) model and replication show the following characteristics: (1) All replication nodes accept read request and send the responses. (2) All replicas function equally. (3) Node failures do not cause loss of write capability, as other replicated node responds.

Cassandra adopts the PPD model. The data distributes among all the nodes in a cluster.

Performance can further be enhanced by adding the nodes. Since nodes read and write both, a replicated node also has updated data. Therefore, the biggest advantage in the model is consistency. When a write is on different nodes, then

write inconsistency occurs.

Figure 3.11 shows the PPD model.

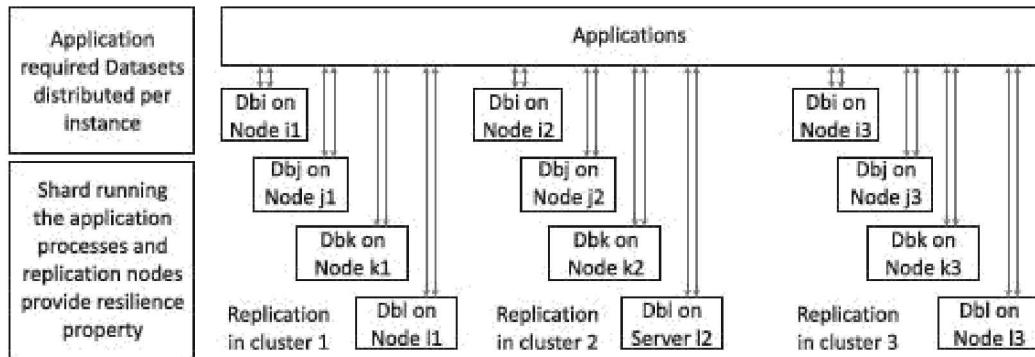


Figure 3.11 Shards replicating on the nodes, which does read and write operations both

3.5.1.5 Choosing Master-Slave versus Peer-to-Peer

Master-slave replication provides greater scalability for read operations. Replication provides resilience during the read. Master does not provide resilience for writes. Peer-to-peer replication provides resilience for read and writes both.

Sharing Combining with Replication Master-slave and sharding creates multiple masters. However, for each data a single master exists. Configuration assigns a master to a group of datasets. Peer-to-peer and sharding use same strategy for the column-family data stores. The shards replicate on the nodes, which does read and write operations both.

3.5.2 Ways of Handling Big Data Problems

Figure 3.12 shows four ways for handling Big Data problems.

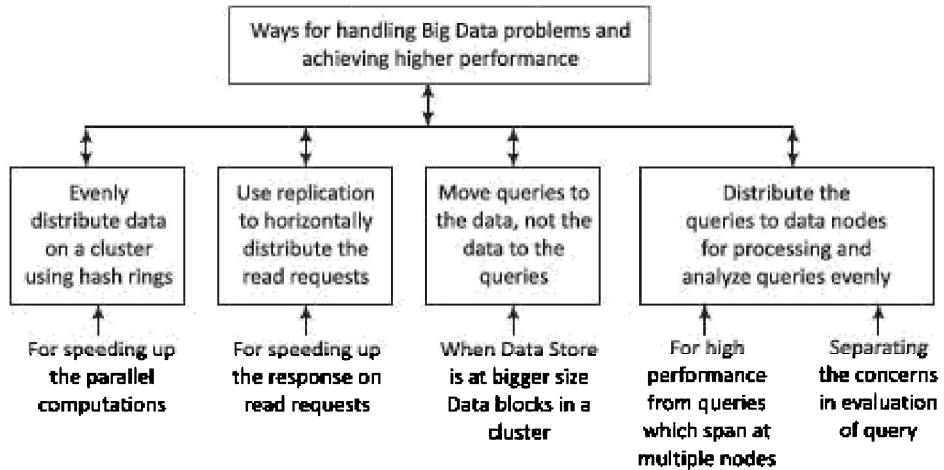


Figure 3.12 Four ways for handling big data problems

Following are the ways:

1. *Evenly distribute the data on a cluster using the hash rings:* Consistent hashing refers to a process where the datasets in a collection distribute using a hashing algorithm which generates the pointer for a collection. Using only the hash of Collection_ID, a Big Data solution client node determines the data location in the cluster. Hash Ring refers to a map of hashes with locations. The client, resource manager or scripts use the hash ring for data searches and Big Data solutions. The ring enables the consistent assignment and usages of the dataset to a specific processor.
2. *Use replication to horizontally distribute the client read-requests:* Replication means creating backup copies of data in real time. Many Big Data clusters use replication to make the failure-proof retrieval of data in a distributed environment. Using replication enables horizontal scaling out of the client requests.
3. *Moving queries to the data, not the data to the queries:* Most NoSQL data stores use cloud utility services (Large graph databases may use enterprise servers). Moving client node queries to the data is efficient as well as a requirement in Big Data solutions.
4. *Queries distribution to multiple nodes:* Client queries for the DBs analyze at

the analyzers, which evenly distribute the queries to data nodes/ replica nodes. High performance query processing requires usages of multiple nodes. The query execution takes place separately from the query evaluation (The evaluation means interpreting the query and generating a plan for its execution sequence).

Self-Assessment Exercise linked to LO 3.4

1. List pros and cons of distribution using sharding.
2. List characteristics of master-slave distribution model.
3. List the benefits of peer-to-peer nodes data distribution model.
4. How is a hash ring used in the distribution of Big Data?

3.6 | MONGODB DATABASE

MongoDB is an open source DBMS. MongoDB programs *create* and *manage* databases. MongoDB manages the collection and document data store. MongoDB functions do querying and accessing the required information. The functions include viewing, querying, changing, visualizing and running the transactions. Changing includes updating, inserting, appending or deleting.

LO 3.5

MongoDB databases and query commands

MongoDB is (i) non-relational, (ii) NoSQL, (iii) distributed, (iv) open source, (v) document based, (vi) cross-platform, (vii) Scalable, (viii) flexible data model, (ix) Indexed, (x) multi-master (Section 3.5.1.3), and (xi) fault tolerant. Document data store in JSON-like documents. The data store uses the dynamic schemas.

The typical MongoDB applications are content management and delivery systems, mobile applications, user data management, gaming, e-commerce, analytics, archiving and logging.

Features Following are features of MongoDB:

1. *MongoDB data store* is a physical container for collections. Each DB gets its own set of files on the file system. A number of DBs can run on a single MongoDB server. DB is default DB in MongoDB that stores within a data folder. The database server of MongoDB is *mongod* and the client is *mongo*.
2. *Collection* stores a number of MongoDB documents. It is analogous to a table of RDBMS. A collection exists within a single DB to achieve a single purpose. Collections may store documents that do not have the same fields. Thus, documents of the collection are schema-less. Thus, it is possible to store documents of varying structures in a collection. Practically, in an RDBMS, it is required to define a column and its data type, but does not need them while working with the MongoDB.
3. *Document model* is well defined. Structure of document is clear, Document is the unit of storing data in a MongoDB database. Documents are analogous to the records of RDBMS table. Insert, update and delete operations can be performed on a collection. Document use JSON (JavaScript Object Notation) approach for storing data. JSON is a lightweight, self-describing format used to interchange data between various applications. JSON data basically has key-value pairs. Documents have dynamic schema.
4. MongoDB is a document data store in which one collection holds different documents. Data store in the form of JSON-style documents. Number of fields, content and size of the document can differ from one document to another.
5. *Storing of data* is flexible, and data store consists of JSON-like documents. This implies that the fields can vary from document to document and data structure can be changed over time; JSON has a standard structure, and scalable way of describing hierarchical data (Example 3.3(ii)).
6. *Storing of documents* on disk is in BSON serialization format. BSON is a binary representation of JSON documents. The mongo JavaScript shell and MongoDB language drivers perform translation between BSON and language-specific document representation.
7. *Querying, indexing, and real time aggregation* allows accessing and analyzing

the data efficiently.

8. *Deep query-ability*—Supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
9. No complex Joins.
10. *Distributed DB* makes availability high, and provides horizontal scalability.
11. *Indexes on any field* in a collection of documents: Users can create indexes on any field in a document. Indices support queries and operations. By default, MongoDB creates an index on the `_id` field of every collection.
12. *Atomic operations on a single document* can be performed even though support of multi-document transactions is not present. The operations are alternate to ACID transaction requirement of a relational DB.
13. *Fast-in-place updates*: The DB does not have to allocate new memory location and write a full new copy of the object in case of data updates. This results into high performance for frequent update use cases. For example, incrementing a counter operation does not fetch the document from the server. Here, the increment operation can simply be set.
14. *No configurable cache*: MongoDB uses all free memory on the system automatically by way of memory-mapped files (The operating systems use the similar approach with their file system caches). The most recently used data is kept in RAM. If indexes are created for queries and the working dataset fits in RAM, MongoDB serves all queries from memory.
15. *Conversion/mapping* of application objects to data store objects not needed

Dynamic Schema Dynamic schema implies that documents in the same collection do not need to have the same set of fields or structure. Also, the similar fields in a document may contain different types of data. Table 3.8 gives the comparison with RDBMS.

Table 3.8 Comparison of RDBMS and MongoDB databases

RDBMS	MongoDB
Database	Data store

Table	Collection
Column	Key
Value	Value
Records / Rows / Tuple	Document / Object
Joins	Embedded Documents
Index	Index
Primary key	Primary key (<code>_id</code>) is default key provided by MongoDB itself

Any relational DB has a typical schema design that shows the number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Replication Replication ensures high availability in Big Data. Presence of multiple copies increases on different database servers. This makes DBs fault-tolerant against any database server failure. Multiple copies of data certainly help in localizing the data and ensure availability of data in a distributed system environment.

MongoDB replicates with the help of a replica set. A replica set in MongoDB is a group of mongod (MongoDb server) processes that store the same dataset. Replica sets provide redundancy but high availability. A replica set usually has minimum three nodes. Any one out of them is called primary. The primary node receives all the write operations. All the other nodes are termed as secondary. The data replicates from primary to secondary nodes. A new primary node can be chosen among the secondary nodes at the time of automatic failover or maintenance. The failed node when recovered can join the replica set as secondary node again. Replica set starts a mongod instance by specifying `-replSet` option before running these commands from mongo (MongoDb Client). Table 3.9 gives the commands used for replication (Recoverability means even on occurrences of failures; the transactions ensure consistency).

Table 3.9 MongoDB Client commands related to replica set

Commands	Description
<code>rs.initiate()</code>	To initiate a new replica set

rs.conf ()	To check the replica set configuration
rs.status ()	To check the status of a replica set
rs.add ()	To add members to a replica set

Figure 3.13 shows a replicated dataset after creating three secondary members from a primary member.

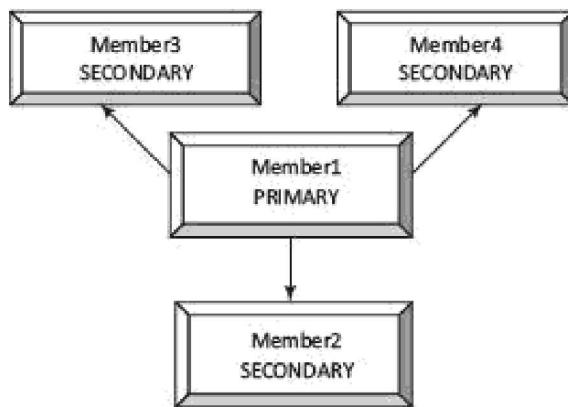


Figure 3.13 Replicated set on creating secondary members

Auto-sharding Sharding is a method for distributing data across multiple machines in a distributed application environment. MongoDB uses sharding to provide services to Big Data applications.

A single machine may not be adequate to store the data. When the data size increases, do not provide data retrieval operation. Vertical scaling by increasing the resources of a single machine is quite expensive. Thus, horizontal scaling of the data can be achieved using sharding mechanism where more database servers can be added to support data growth and the demands of more read and write operations.

Sharding automatically balances the data and load across various servers. Sharding provides additional write capability by distributing the write load over a number of mongod (MongoDB Server) instances.

(Figure 3.10) Basically, it splits the dataset and distributes them across multiple DBs, called shards on the different servers. Each shard is an independent DB. The whole collection of shards forms a single logical DB. If a DB has a 1 terabyte dataset distributed amongst 20 shards, then each shard contains only 50 Giga Byte of data.

A shard stores lesser data than the actual data and handles lesser number of operations in a single instance. For example, to insert data into a collection, the application needs to access only the shard that contains the specified collection. A cluster can thus easily increase its capacity horizontally.

Data Types Table 3.10 gives data types which MongoDB documents support.

Table 3.10 Data types which MongoDB documents support

Type	Description
Double	Represents a float value.
String	UTF-8 format string.
Object	Represents an embedded document.
Array	Sets or lists of values.
Binary data	String of arbitrary bytes to store images, binaries.
Object id	ObjectIds (MongoDB document identifier, equivalent to a primary key) are: small, likely unique, fast to generate, and ordered. The value consists of 12-bytes, where the first four bytes are for timestamp that reflects the instance when ObjectId creates.
Boolean	Represents logical true or false value.
Date	BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch (Jan 1, 1970).
Null	Represents a null value. A value which is missing or unknown is Null.
Regular Expression	RegExp maps directly to a JavaScript RegExp
32-bit integer	Numbers without decimal points save and return as 32-bit integers.
Timestamp	A special timestamp type for internal MongoDB use and is not associated with the regular date type. Timestamp values are a 64-bit value, where first 32 bits are time, t (seconds since the Unix epoch), and next 32 bits are an incrementing ordinal for operations within a given second.

64-bit integer	Number without a decimal point save and return as 64-bit integer.
Min key	MinKey compare less than all other possible BSON element values, respectively, and exist primarily for internal use.
Max key	MaxKey compares greater than all other possible BSON element values, respectively, and exist primarily for internal use.

Rich Queries and Other DB Functionalities MongoDB offers a rich set of features and functionality compared to those offered in simple key-value stores. They can be comparable to those offered by any RDBMS. MongoDB has a complete query language, highly-functional secondary indexes (including text search and geospatial), and a powerful aggregation framework for data analysis. MongoDB provides functionalities and features for more diverse data types than a relational DB, and at scale. Table 3.11 gives a comparison of features.

Table 3.11 Comparison of features MongoDB with respect to RDBMS

Features	RDBMS	MongoDB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Horizontal Scaling	No	Yes

The ability to derive a document-based data model is also a distinct advantage of MongoDB. The method of storing data in the form of BSON (Binary JSON) helps to store the data in a very rich way while can hold arrays and other documents.

MongDB Query Language and Database Commands Table 3.12 gives MongoDB commands for querying the DBs.

Table 3.12 MongoDB querying commands

Command	Functionality
Mongo	Starts MongoDB; (*mongo is MongoDB client). The default database in MongoDB is test.
db.help()	Runs help. This displays the list of all the commands.
db.stats()	Gets statistics about MongoDB server.
Use <database name>	Creates database
Db	Outputs the names of existing database, if created earlier
Dbs	Gets list of all the databases
db.dropDatabase()	Drops a database
db.database.name.insert()	Creates a collection using insert()
db.<database name>.find()	Views all documents in a collection
db.<database name>.update()	Updates a document
db.<database name>.remove()	Deletes a document

Following explains the sample usages of the commands:

To Create database Command use - use command creates a database; For example, Command use lego creates a database named lego. (A sample database is created to demonstrate subsequent queries. The Lego is an international toy brand). Default database in MongoDB is test.

To see the existence of database Command db - db command shows that lego database is created.

To get list of all the databases Command show dbs - This command shows

the names of all the databases.

To drop database Command `db.dropDatabase()` – This command drops a database. Run use `lego` command before the `db.dropDatabase()` command to drop `lego` Database. If no database is selected, the default database `test` will be dropped.

To create a collection Command `insert()` – To create a collection, the easiest way is to insert a record (a document consisting of keys (Field names) and Values) into a collection. A new collection will be created, if the collection does not exist. The following statements demonstrate the creation of a collection with three fields (`ProductCategory`, `ProductId` and `ProductName`) in the `lego`:

```
db.lego.insert
(
{
    "ProductCategory": "Airplane",
    "ProductId": 10725,
    "ProductName": "Lost Temple"
}
)
```

To add array in a collection Command `insert()` - Insert command can also be used to insert multiple documents into a collection at one time.

```

db.lego.insert
(
  [
    {
      "ProductCategory": "Airplane",
      "ProductId": 10725,
      "ProductName": "Lost Temple"
    },
    {
      "ProductCategory": "Airplane",
      "ProductId": 31047,
      "ProductName": "Propeller Plane"
    },
    {
      "ProductCategory": "Airplane",
      "ProductId": 31049,
      "ProductName": "Twin Spin Helicopter"
    }
  ]
)

```

To view all documents in a collection Command db.<database name>.find() - Find command is equivalent to select query of RDBMS. Thus, “Select * from lego” can be written as db.lego.find() in MongoDB. MongoDB created unique objectID (“_id”) on its own. This is the primary key of the collection. Command db.<database name>.find().pretty() gives a prettier look.

To update a document Command db.<database name>.update() - Update command is used to change the field value. By default, multi attribute is false. If {multi: true} is not written then it will update only the first document.

To delete a document Command db.<database name>.remove() - Remove command is used to delete the document. The query db.<database name>.remove((“ProdctID”:10725)) removes the document whose productId is 10725.

Self-Assessment Exercise linked to LO 3.5

1. Compare MongoDB and RDBMS?
2. Give example that demonstrates the uses of various data types of MongoDB.

3. List the functions of MongoDB query language and database commands.
4. How will you consider MongoDB as complete query language, which imbibes highly-functional secondary indices (including text search and geospatial), and provides a powerful aggregation framework for data analysis?

3.7 | CASSANDRA DATABASES

Cassandra was developed by Facebook and released by Apache. Cassandra was named after Trojan mythological prophet Cassandra, who had classical allusions to a curse on oracle. Later on, IBM also released the enhancement of Cassandra, as open source version. The open source version includes an IBM Data Engine which processes No SQL data store. The engine has improved throughput when workload of read-operations is intensive.

Cassandra is basically a column family database that stores and handles massive data of any format including structured, semi-structured and unstructured data.

Apache Cassandra DBMS contains a set of programs. They *create* and *manage* databases. Cassandra provides functions (commands) for querying the data and accessing the required information. Functions do the viewing, querying and changing (update, insert or append or delete), visualizing and perform transactions on the DB.

Apache Cassandra has the distributed design of Dynamo. Cassandra is written in Java. Big organizations, such as Facebook, IBM, Twitter, Cisco, Rackspace, eBay, Twitter and Netflix have adopted Cassandra.

Characteristics of Cassandra are (i) open source, (ii) scalable (iii) non-relational (v) NoSQL (iv) Distributed (vi) column based, (vii) decentralized, (viii) fault tolerant and (ix) tuneable consistency.

Features of Cassandra are as follows:

LO 3.6

Cassandra databases,
data-model and clients,
and integration with the
Hadoop

1. Maximizes the number of writes – writes are not very costly (time consuming)
2. Maximizes data duplication
3. Does not support Joins, group by, OR clause and aggregations
4. Uses Classes consisting of ordered keys and semi-structured data storage systems
5. Is fast and easily scalable with write operations spread across the cluster. The cluster does not have a master-node, so any read and write can be handled by any node in the cluster.
6. Is a distributed DBMS designed for handling a high volume of structured data across multiple cloud servers
7. Has peer-to-peer distribution in the system across its nodes, and the data is distributed among all the nodes in a cluster (Section 3.5.1.4).

Data Replication Cassandra stores data on multiple nodes (data replication) and thus has no single point of failure, and ensures availability, a requirement in CAP theorem. Data replication uses a replication strategy. Replication factor determines the total number of replicas placed on different nodes. Cassandra returns the most recent value of the data to the client. If it has detected that some of the nodes responded with a stale value, Cassandra performs a read repair in the background to update the stale values.

Components at Cassandra Table 3.13 gives the components at Cassandra and their description.

Table 3.13 Components of cassandra

Component	Description
Node	Place where data stores for processing
Data Center	Collection of many related nodes
Cluster	Collection of many data centers
Commit log	Used for crash recovery; each write operation written to commit log

Mem-table	Memory resident data structure, after data written in commit log, data write in mem-table temporarily
SSTable	When mem-table reaches a certain threshold, data flush into an SSTable disk file
Bloom filter	Fast and memory-efficient, probabilistic-data structure to find whether an element is present in a set, Bloom filters are accessed after every query.

Scalability Cassandra provides linear scalability which increases the throughput and decreases the response time on increase in the number of nodes at cluster.

Transaction Support Supports ACID properties (Atomicity, Consistency, Isolation, and Durability).

Replication Option Specifies any of the two replica placement strategy names. The strategy names are Simple Strategy or Network Topology Strategy. The replica placement strategies are:

1. Simple Strategy: Specifies simply a replication factor for the cluster.
2. Network Topology Strategy: Allows setting the replication factor for each data center independently.

Data Types Table 3.14 gives the data types built into Cassandra, their usage and descriptions

Table 3.14 Data types built into Cassandra, their usage and description

CQL Type	Description
ascii	US-ASCII character string
bigint	64-bit signed long integer
blob	Arbitrary bytes (no validation), BLOB expressed in hexadecimal
boolean	True or false
counter	Distributed counter value (64-bit long)
decimal	Variable-precision decimal integer, float
double	64-bit IEEE-754 double precision floating point integer, float

float	32-bit IEEE-754 <i>single precision</i> floating point integer, float
inet	IP address string in IPv4 or IPv6 format, used by the python-cql driver and CQL native protocols
int	32-bit signed integer
list	A collection of one or more ordered elements
map	A JSON-style array of literals: {literal: literal, literal: literal ...}
set	A collection of one or more elements
text	UTF-8 encoded string
timestamp	Date plus time, encoded as 8 bytes since epoch integers, strings
varchar	UTF-8 encoded string
varint	Arbitrary-precision integer

Cassandra Data Model Cassandra Data model is based on Google's BigTable (Section 3.3.3.2). Each value maps with two strings (row key, column key) and timestamp, similar to HBase (Example 2.4). The database can be considered as a sparse distributed multi-dimensional sorted map. Google file system splits the table into multiple tablets (segments of the table) along a row. Each tablet, called META1 tablet, maximum size is 200 MB, above which a compression algorithm used. META0 is the master-server. Querying by META0 server retrieves a META1 tablet. During execution of the application, caching of locations of tablets reduces the number of queries.

Cassandra Data Model consists of four main components: (i) Cluster: Made up of multiple nodes and keyspaces, (ii) Keyspace: a namespace to group multiple column families, especially one per partition, (iii) Column: consists of a column name, value and timestamp and (iv) Column-family: multiple columns with row key reference. Cassandra does keyspace management using partitioning of keys into ranges and assigning different key-ranges to specific nodes.

Following Commands prints a description (typically a series of DDL statements) of a schema element or the cluster:

DESCRIBE CLUSTER
DESCRIBE SCHEMA
DESCRIBE KEYSPACES
DESCRIBE KEYSPACE <keyspace name>
DESCRIBE TABLES
DESCRIBE TABLE <table name>
DESCRIBE INDEX <index name>
DESCRIBE MATERIALIZED VIEW <view name>
DESCRIBE TYPES
DESCRIBE TYPE <type name>
DESCRIBE FUNCTIONS
DESCRIBE FUNCTION <function name>
DESCRIBE AGGREGATES
DESCRIBE AGGREGATE <aggregate function name>

Consistency Command **CONSISTENCY** shows the current consistency level. **CONSISTENCY <LEVEL>** sets a new consistency level. Valid consistency levels are ANY, ONE, TWO, THREE, QUORUM, LOCAL_ONE, LOCAL_QUORUM, EACH_QUORUM, SERIAL AND LOCAL_SERIAL. Following are their meanings:

1. ALL: Highly consistent. A write must be written to commitlog and memtable on all replica nodes in the cluster.
2. EACH_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in all data centers.
3. LOCAL_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in the same center.
4. ONE: A write must be written to commitlog and memtable of at least one replica node.
5. TWO, THREE: Same as One but at least two and three replica nodes, respectively.

6. LOCAL_ONE: A write must be written for at least one replica node in the local data center.
7. ANY: A write must be written to at least one node.
8. SERIAL: Linearizable consistency to prevent unconditional update.
9. LOCAL_SERIAL: Same as Serial but restricted to the local data center.

Keyspaces A keyspace (or key space) in a NoSQL data store is an object that contains all column families of a design as a bundle. Keyspace is the outermost grouping of the data in the data store. It is similar to relational database. Generally, there is one keyspace per application. Keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node.

Create Keyspace Command `CREATE KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor': '<No. of replicas>'} AND durable_writes = '<TRUE/FALSE>;'`

CREATE KEYSPACE statement has attributes replication with option class and replication factor, and durable_write.

Default value of *durable_writes* properties of a table is set to true. That commands the Cassandra to use Commit Log for updates on the current Keyspace true or false. The option is not compulsory.

1. ALTER KEYSPACE command changes (alter) properties, such as the number of replicas and the durable_writes of a keyspace: `ALTER KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor': '<No. of replicas>}';`
2. DESCRIBE KEYSPACE command displays the existing keyspaces.
3. DROP KEYSPACE command drops a keyspace:
4. Re-executing the drop command to drop the same keyspace will result in configuration exception.
5. Use KEYSPACE command connects the client session with a keyspace.

Cassandra Query Language (CQL) Table 3.15 gives the CQL commands and their functionalities.

Table 3.15 CQL commands and their functionalities

Command	Functionality
CQLSH	A command line shell for interacting with Cassandra through CQL
HELP	Runs help. This displays the list of all the commands
CONSISTENCY	Shows the current consistency level
EXIT	Terminate the CQL shell
SHOW HOST	Displays the host
SHOW VERSION	Displays the details of current cqlsh session such as host, Cassandra version, or data type assumptions
CREATE KEYSPACE <Keyspace Name>	Creates keyspace with a name
DESCRIBE KEYSPACE <Keyspace Name>	Displays the keyspace with a name
ALTER KEYSPACE <Keyspace Name>	Modifies keyspace with a name
DROP KEYSPACE <Keyspace Name>	Deletes keyspace with a name
CREATE (TABLE COLUMNFAMILY)	Creates a table or column family
COLLECTIONS	Lists the Collections

The following example provides the sample usages of the commands.

EXAMPLE 3.13

Give the examples of usages of various CQL commands.

SOLUTION

- (1) Create Table Command: CREATE TABLE command creates a table in the current keyspace:

```
CREATE      (TABLE      |      COLUMNFAMILY)      <tablename>
('<column-definition>',      '<column-definition>')
(WITH <option> AND <option>);
```

Primary key is a column used to uniquely identify a row. Therefore, defining a primary key is compulsory while creating a table. A primary key is made of one or more columns of a table.

Example: Create a table *ProductInfo* in the keyspace *lego*, with primary key field *ProductId*.

```
Use lego;
```

```
Create table ProductInfo(ProductId int primary
key, ProductType text);
```

- (2) **Describe Tables Command:** DESCRIBE TABLE Command displays all the tables in the current keyspace:

```
DESCRIBE TABLE <TABLE NAME>;
```

Example: Display the details of a table *ProductInfo*:

```
DESCRIBE TABLE ProductInfo;
```

- (3) **Alter Tables Command:**

```
ALTER TABLE Command ALTER (TABLE | COLUMNFAMILY)
<tablename> (ADD | DROP) <column name>
```

The above command adds a column in the table or to delete a column of the table:

Example: Add a column *dateOfManufacturing* in the table *ProductInfo*:

```
ALTER TABLE ProductInfo add dateOfManufacturing
timestamp;
```

* timestamp is a datatype used for date fields.

- (4) **Cassandra CURD Operations:** (CURD—Create, Update, Read and Delete data into tables) :

- (a) **Insert Command:**

INSERT command creates data in a table:

```
INSERT INTO <tablename> (<column1 name>, <column2  
name>....) VALUES (<value1>, <value2>....) USING  
<option>
```

(b) Update Command:

UPDATE command updates data in a table. The following keywords are used while updating data in a table:

Where – This clause is used to select the row to be updated.

Set – Set the value using this keyword.

Must – Includes all the columns composing the primary key.

If a given row is unavailable, then UPDATE creates a new row.

```
UPDATE <tablename> SET <column name> = <new value>  
<column name> = <value>.... WHERE <condition>
```

[A WHERE clause can be used only on the columns that are a part of primary key or have a secondary index on them.]

(c) Select Command

SELECT command reads the data from a table. The command can read a whole table, a single column, or a particular cell:

```
SELECT <column name (s)> FROM <Table Name>
```

To select all records:

```
SELECT * FROM <Table Name>
```

To select records that fulfils required condition:

```
SELECT <column1, column2, ...> FROM <Table Name>  
where <Condition>
```

Example: Select Product Type, Product Id, Product Name, and Product Cost of Product whose ProductId is 31047:

```
SELECT Product Type, Product Id, Product Name, and  
Product Cost
```

```
from ProductInfo where ProductId = 31047;
```

(d) Delete Command

DELETE command deletes data from a table:

```
DELETE FROM <identifier> WHERE <condition>;
```

Example: Delete row from a table where Product id is 31047:

```
DELETE FROM ProductInfo WHERE ProductId = 31047;
```

(5) Creating a Table with List

CREATE Table command is used for creating a table with a list.

The following query creates a table with two columns, one is the primary key and the other has multiple items (List):

```
CREATE TABLE data (<column name>, <data type>
PRIMARY KEY, <column name list<data type>>);
```

Example : Create a sample table *ContactInfo* with three columns: *Sno*, *name* and *EmailId*. To store multiple Email Ids, use a list:

```
create table ContactInfo (Sno int Primary key,
Name text, emailId list <text>);
```

(6) Insert Command for inserting data into a list

INSERT Command also inserts data into a list. To insert data into the elements in a list, enter all the values separated by a comma within square braces []:

```
INSERT INTO <table name> (column1, column2, )
VALUES (value1, value2, [list value1, list value2,
...])
```

Example: Insert data of three persons into the *ContactInfo* Table:

```
Insert into ContactInfo (Sno, Name, EmailId)
values
(1, 'Rahul', ['rahul@gmail.com',
'rahul@yahoo.com']);
```

```
Insert into ContactInfo (Sno, Name, EmailId)
values (1, 'Geetika', ['geetika@gmail.com',
'geetika@yahoo.com']);

Insert into ContactInfo (Sno, Name, EmailId)
values (1, 'Deepika', ['deepika@gmail.com',
'deepika@yahoo.com']);
```

(7) Update Command for updating Data into a List

UPDATE command also updates data into a list:

```
UPDATE <table Name> SET <New data> where
<condition>.
```

Example : Add one more email Id to the *emailId* list in *ContactInfo* table :

```
UPDATE ContactInfo SET emailId = emailId +
['preeti@ymail.com'] where SNo=1.
```

Cassandra Client A relational database client connects to DB server using drivers. Java JDBC driver API enables storing and retrieving data. Cassandra has peer-to-peer distribution architecture. Several instances require the clients. The driver enables the use of different languages for connecting to DBs. Cassandra does not include the drivers.

A client-generation layer enables the database interactions. AVRO project provides the client generation layer. Third party sources provide Cassandra clients in Java, Ruby, C#, Python, Perl, PHP, C++, Scala and other languages. The Cassandra client can be included in the applications.

Cassandra Hadoop Support Cassandra 2.1 has Hadoop 2 support. The setup and configuration overlays a Hadoop cluster on the Cassandra nodes. A server is configured for the NameNode and JobTracker. Each Cassandra node then installs the TaskTracker and Data Node.

The nodes in the Cassandra cluster can read data from the data in the Data Node in HDFS as well as from Cassandra. A client application sends the MapReduce input to Job Tracker/Resource Manager. RM/JobTracker sends a MapReduce request of job to the Task Trackers/Node Managers and clients such

as MapReduce and Pig. The Reducer output writes to Cassandra. The client gets the results from Cassandra.

Self-Assessment Exercise linked to LO 3.6

1. List the differences between Cassandra, Google BigTable and HBase data models.
2. Compare Cassandra and RDBMS.
3. List the data types used in Cassandra.
4. How are the Cassandra query language and database commands used?
5. List the components in Casandra and their uses.
6. Write the syntax to create keyspace in Cassandra. State when the ALTER keyspace is used.
7. How are the Cassandra CQL collections used?

KEY CONCEPTS

ACID properties

aggregation

application

availability

BASE

BigTable

BLOB

BSON

CAP theorem

Cassandra

client

cluster

column family
consistency
data architecture pattern
database
data model
data node
data tree
data type
deserialization
distributed database
distribution model
document data store
DynamoDB
fixed table schema
hash ring
hierachal record
indexing
Java object
Join
JSON
key-value pair
keyspace
Master-slave
MongoDB
Multi-master DB
NoSQL
object data store
OLAP

pattern
peer-to-peer
Persistency
Querying
RDBMS
relationship map
replication
Scalability
schema-less
serialization
server
sharding
shared nothing
sorted keys
SQL
transaction
XML
XPath



Learning Outcomes

LO 3.1

- A new category of data stores is NoSQL (Not Only SQL) databases. NoSQL is an altogether new approach of thinking about data stores.
- NoSQL data model offers relaxation in one or more of the ACID properties, instead follows CAP theorem and BASE.

- NoSQL DBs possess greater flexibility for data manipulation (compared to SQL).
- NoSQL data does not need fixed schema. The data model may drop support to Joins in Big Data environment.

LO 3.2

- Key-value pairs data store can be used as Big Data NoSQL database.
- Key-value pair is a simplest way to implement a schema-less data store. The pairs can store any data type in the value field. The store uses a primary-key access; therefore, the store can be easily scaled up to large data, and data retrieves fast using keys as the indices.
- NoSQL DB also stores the hierarchical information in a single unit, called *document* store. ‘Document’ stores unstructured data. Data stores in nested hierarchies. Data have no object-relational layer for the mapping. Document data store can be at multiple NameNodes and thus enables higher resources availability.
- Column-family data stores are similar to sparse matrix data. Columns are logically grouped into column families. Column families can logically group as super column. Data stores in memory are column-based than row-based. This facilitates faster OLAP processing. The table can be partitioned into row groups (or stripes). Data stores can be in columnar data RC, ORC and Parquet formats.
- An object data store consists of functions for supporting using APIs.
- Graph database with interconnected data nodes provides one of the best database systems. They enable fast network searches.
- A selected architecture needs variations due to business requirements. Business requirements are easy to use and have long-term competitive advantage.

LO 3.3

- Big Data solution emphasizes on scalable storage of a much higher magnitude of data (of terabytes and petabytes) by dropping support for database Joins, storing data differently and using several distributed servers (data nodes) together as a cluster.
- Big Data NoSQL solution provides high scalability, supports replication, no schema or no fixed data model, support integrated caching: not inflexible like SQL/RDBMS DBs and have flexibility in following ACID rules.

LO 3.4

- SN architecture features are independence, self-healing and no network contention. Each node data functions as a shard of the DB.
- Distribution models are (i) single server, (ii) sharding, (iii) master-slave, (iv) multi-master (v) peer-to-peer, and (vi) hash ring based.

LO 3.5

- MongoDB is (i) non-relational, (ii) distributed, (iii) open source, (iv) NoSQL, (v) document-based, (vi) Cross-platform, (vii) scalable, (viii) flexible, (ix) indexed, (x) deep querying ability, (xi) scalable multi-master data store with no single points of failure and (xii) provisions the data modeling flexibility.
- Querying, indexing and real-time aggregation functions access and analyze the data efficiently, and conversion/mapping of application objects to database objects is not needed.

LO 3.6

- Cassandra is (i) open source, (ii) scalable (iii) non-relational (iv) peer-to-peer distributed system (v) NoSQL (vi) column-based, (vii) decentralized, (viii) fault tolerant and (ix) tunable consistency.

- Cassandra Data model is based on Google's BigTable. Each value maps with two strings (row key, column key) and timestamp, similar to HBase.
- Cassandra data model consists of four main components: (i) Cluster: made up of multiple nodes and keyspaces, (ii) Keyspace: a namespace to group multiple column families, especially one per partition, (iii) Column: consists of a column name, value and timestamp and (iv) Column-family: multiple columns with row key reference.

Objective Type Questions

Select one correct-answer option for each questions below:

3.1 Big Data NoSQL data store (i) transactions show ACID properties, (ii) used in distributed environment, (iii) NoSQL DBs possess increasing flexibility for data manipulation, (iv) follows a fixed data storage schema (v) use the concept of Joins, and (vi) must follow CAP theorem.

- ii and iii
- all
- ii, iii and vi
- iii and iv

3.2 High scalability, flexibility and performance and low complexity are the characteristics of

- key-value pair, (ii) document (iii) column-family, and (iv) graph databases.

- only i and ii
- only i
- only ii
- iii and iv

3.3 The advantages of a key-value store are: (i) can store any data type in the value field, (ii) stores the information as a BLOB of data (such as, text,

hypertext, images, video and audio), but does not return the same BLOB when data is retrieved, (iii) scalability, (iv) reliability, (v) portability, and (vi) low operational cost.

- (a) all except i
- (b) all
- (c) all except ii
- (d) ii to vi

3.4 An object data store consists of functions for supporting (i) scalability, (ii) indexing, (iii) large collections, (iv) querying language, processing and optimization (s), (v) transactions, (vi) data replication for high availability, data distribution model, data integration (such as with relational database, XML, custom code), (vii) schema evolution, (viii) persistency, (ix) persistent object life cycle, (x) adding modules, (xi) locking and caching strategy, and (xii) object store may support versioning for collaboration.

- (a) i to v, ix to xiii
- (b) all
- (c) all except viii, ix ad xii
- (d) all except iii, vi and viii

3.5 Graph database with interconnected data nodes (i) provides one of the best data store system, (ii) provides one of the highly complex data store system, (iii) enables fast network searches, (iv) uses linked datasets, (v) used in social media data, (vi) consists of small data size records with complex interactions between graph nodes but not between the hypergraph nodes

(vii) uses graph with nodes and edges connecting each other through the relations, associations and properties, and (viii) BASE properties.

- (a) i, iii, iv and viii
- (b) ii to v
- (c) ii to vi

(d) all except vi

3.6 Selecting a NoSQL data architectural pattern can be as follows: (i) performing a use-case driven difficulty analysis for all architectural patterns. Assign the difficulties level, such as low, medium or high in the following processes (iii) ingestion, (iv) validation of structure and its fields,

(v) updating process using near real-time approach, (vi) searching process using partial text or by changing the sorting order, (vii) exporting the reports or application results in HTML, XML or JSON, and estimating total efforts for each architecture for all the business requirements.

(a) all

(b) all except v and vi

(c) i to v

(d) all except ii

3.7 Big Data solution emphasizes on scalable storage of a much higher magnitude of data (of terabytes and petabytes) by (i) supporting database Joins, (ii) storing data differently and using several distributed servers (Data Nodes) together as a cluster, (iii) must perform transactions using ACID properties, (iv) implementing CAP theorem without compromising the consistency factor) to make transactions faster and easier to scale, and (v) must be partitioning tolerant.

(a) ii and v

(b) all except i, iii

(c) all except iii and iv

(d) all

3.8 Big Data NoSQL solution: (i) have high and easily scalability, (ii) supports replication, (iii) use multiple copies of data store across multiple nodes of the cluster, (iii) maintains NoSQL Servers, (iv) NoSQL data store implementation is easy and typically uses cheap servers to manage the exploding data and transaction while RDBMS databases are expensive and it uses big servers and storage systems, and (v) storing and processing data

cost per gigabyte in the case of NoSQL can be many times more than the cost of RDBMS.

- (a) i, ii and v
- (b) all except i, iii
- (c) all except iii and iv
- (d) i to iv

3.9 Big Data NoSQL-solution should be (i) schema-less or not fixed data model, (ii) without any predefined schema, (iii) the format or data model can be changed any time, (iv) without application disruption and (v) with change in management, (v) support integrated caching, and (vi) not inflexible like SQL/RDBMS DBs.

- (a) i to v
- (b) all
- (c) all except iii
- (d) i to iv

3.10 A Big Data solution does the following: (i) unevenly distributes data on a cluster, (ii) distributes using token rings, (iii) uses replication and vertical distribution of the client read requests, (iv) creates backup copies of data in batches, (v) moves queries to the data, and data to queries, (vi) distributes queries to multiple nodes, and (vii) query execution takes place separately from query evaluation.

- (a) ii to iv
- (b) vi and vii
- (c) none
- (d) iv to vii

3.11 MongoDB is (i) non-relational, (ii) distributed, (iii) open source, (iv) NoSQL, (v) document-based, (vi) cross-platform, (vii) scalable, (viii) data modeling inflexibility, (ix) indexed, and
(x) scalable multi-master data store with no single point of failure.

- (a) all except viii
- (b) all except v and vi
- (c) i to vi
- (d) i to vii

3.12 MongoDB features are: (i) document model is well defined, (ii) structure of document is clear, stores documents on disk in the (iii) BSON serialization format, (iv) JSON format, and
(v) querying, indexing, real-time aggregation and allows accessing and analyzing the data efficiently.

- (a) all
- (b) i, iii and iv
- (c) all except iv
- (d) iii, iv and v

3.13 Cassandra data model is based on (i) Google's BigTable. Each value maps with two strings (row key, column key) and timestamp, similar to HBase,
(ii) graph database (iii) distributed
(iv) NoSQL (v) column-based, (vi) centralized, (vii) fault tolerant and (vii)
tunable consistency.

- (a) all
- (b) i, iii and iv
- (c) all except i
- (d) all except ii and vi

3.14 Cassandra data model consists of four main components: (i) Cluster: made up of multiple nodes and keyspaces, (ii) Keyspace: a namespace to group multiple column families, especially one per partition, (iii) Column: consists of a column name, value and timestamp, (iv) Column-family: multiple columns with row key reference, (v) provides a prompt in Cassandra query language shell (CQLSh) that allows keying and execution of commands in Cassandra Query Language (CQL).

- (a) i to iii
- (b) ii and iii
- (c) all
- (d) i, ii and iv

3.15 Cassandra features are as follows: (i) maximizes the number of writes – writes are not very costly (time consuming), (ii) maximizes data duplication, (iii) does not support Joins, group by, OR clause and aggregations, (iv) uses Classes consisting of ordered keys and semi-structured data storage systems, (v) is fast and easily scalable with write operations spread across the cluster, and (vi) the cluster does not have a master node, so any read and write can be handled by any node in the cluster.

- (a) all except i and iii
- (b) all
- (c) all except iv
- (d) all except iii and vi

Review Questions

- 3.1 When should data store be NoSQL instead of relational database? Why do Big Data analytics use NoSQL data stores? **(LO 3.1)**
- 3.2 How does NoSQL data store possess increasing flexibility in adding data? **(LO 3.1)**
- 3.3 How does CAP theorem apply in distributed data models? How is it applicable to NoSQL systems? What is eventual consistency in NoSQL stores? **(LO 3.1)**
- 3.4 Compare NoSQL databases with SQL databases in terms of the data model, schema, type of data architecture patterns, scalability, use of SQL Joins, data size preferences, consistency, ACID properties and top IT companies support. **(LO 3.1)**

- 3.5 Describe the pros and cons of (i) key-value data store, (ii) document data store, (iii) object data store, and (iii) graph database. **(LO 3.2)**
- 3.6 Why should the column-family data store be used for the student grade-sheets of a semester examinations showing semester subject grade-points (SGPs) (between 1 and 10) and semester grade point averages (SGPAs)? What does sparse data mean in student grade-sheets columnar data. **(LO 3.2)**
- 3.7 Describe the characteristics of column-family data stores. How do they suit the OLAP operations? How does BigTable store the data? **(LO 3.2)**
- 3.8 Describe the pros and cons of (i) RC, (ii) ORC and (iii) Parquet file format data stores.
- 3.9 Describe graph database characteristics. How are BASE properties exhibited in graph DBs?
(LO 3.2)
- 3.10 How are replication and sharding used? Explain how does sharding help in minimizing the downtime? **(LO 3.3)**
- 3.11 What are the features of shared-nothing (SN) architecture? How does Big Data Store SN system partition? How does a partition process the different queries? **(LO 3.4)**
- 3.12 Describe four ways for handling Big Data problems. **(LO 3.4)**
- 3.13 Explain MongoDB commands for querying the DBs? How can one achieve transaction and locking in MongoDB? How will MongoDB command be used to insert a document in a database called ‘Toys’ and collection called ‘Train’? **(LO 3.5)**
- 3.14 Discuss the cluster and failover model in Cassandra. Compare the peer-to-peer model that Cassandra supports with the master-slave model that other data stores, such as MongoDB support. What are the pros and cons of each model? **(LO 3.5)**
- 3.15 What are the important design considerations when using a column-family data store like Cassandra? List and explain usages of data types built into

Cassandra. (LO 3.6)

3.16 List and explain usages of Cassandra Query Language (CQL) commands and their functionalities. How does the table data store create using CQL? (LO 3.6)

Practice Exercises

3.1 List ten examples where NoSQL data stores are required. (LO 3.1)

3.2 Show the increasing flexibility in NoSQL DB of car company by appending customer post-sales feedbacks, maintenance and service centre feedbacks about the models, and the customer region-wise preference analysis reports. (LO 3.1)

3.3 A company manufactures and sells car through large number of showrooms. Each car showroom records in main table and transaction tables.

Assume that each week company records the car sells. The table data are as follows:

Showroom ID (SR_ID)	Week Number (counting 1.1.2018) (wkNum)	Jagaur Land Rover Sales Number (JLRSNum)	Hexa Sales Number (HSNum)	Zest Sales Number (ZSNum)	Nexon Sales Number (NSNum)	Safari Strome Sales Number (SSSNum)
124	1	2	8	4	7	10
125	1	1	7	9	6	9
126	1	1	9	4	8	3

Jagaur Land Rover Cost Rs. (JLRC)	Hexa Cost Rs. (HC)	Zest Cost Rs. (ZC)	Nexon Cost Rs. (NC)	Safari Strome Cost Rs. (SSC)
20 M	0.8 M	0.75 M	0.7 M	1 M

Write the (key/values) pairs in a week that estimate the total sales per

week per showroom.

(LO 3.2)

- 3.4 Recall Practice Exercise 2.6 and Exercise 3.3. Consider a car company selling Jagaur Land Rover, Hexa, Zest, Nexon and Safari Storme models of Car. Assign IDs as keys in rows 1 to 999999 and column 1, row 0 column head is 'show room ID'

Assume key (at column-head) row 0 corresponds to Jagaur Land Rover Weekly Sales (JLRWS) in column 2. JLRWS values for different showrooms are in successive rows from row 1 to 999999. The values save in same column at successive memory addresses starting from address 1000000. How the total showrooms sales calculation of JLRWS will be faster than row format? How will the addresses be assigned? **(LO 3.2)**

- 3.5 Geographic Information Systems (GIS), like Google Maps stores geographic information in BigTable. How will the values be retrieved during the following: (a) Identification of a location using its longitude and latitude coordinates, (b) Storage of items once, and then provides multiple access paths (queries) to let one view the data. **(LO 3.2)**

- 3.6 Using graph database model, how will the followings store: student id, contact info, admission info, and five courses each in four semesters and SGPs, SGPs and CGPAs at the end of each semester and division awarded? **(LO 3.2)**

- 3.7 Recall Example 3.12 which listed selection requirements for the data architecture pattern the database of University students. Write logical reasons for each. **(LO 3.2)**

- 3.8 Listed selection requirements for the data architecture pattern for the ACVMs Chocolate sales data [Table 3.3]. **(LO 3.3)**

- 3.9 Give two examples each of usages of single server, sharding, master-slave and peer-to-peer distribution models. **(LO 3.4)**

- 3.10 Make a table in which left column gives the outputs using MongoDB querying commands. Fill the right column of the table giving action on the command. **(LO 3.5)**

3.11 Complete the following table fields for actions directed by Cassandra these command.

Actions directed by the command	Code
Create a Table with Set: Table name: ContactInfo Fields: Sno (primary key), Name, set of Contact numbers	
Inserting 3 Datasets in table ContactInfo, and read them	
Updating a set, adding one more contact number to person with Sno = 1, and reading all the data.	

Collections: Map Collection

Actions directed by the command	Code
Create a Table with Map Table name: ContactInfo Fields: Sno (primary key), Name, Map of address	
Inserting 3 Datasets in table ContactInfo, and read them	
Updating a Map, adding one more address to person with Sno = 1, and reading all the data.	

(LO 3.6)

3.12 Recapitulate Practice Exercise 3.3. Consider car company selling *Jagaur Land Rover*, *Hexa*, *Zest*, *Nexon* and *Safari Storme* models of cars. How will the CQL commands be used to create the table for weekly sales log at multiple car company showrooms?

CCSR_id	Date (DT) mmddyy	<i>Jagaur Land Rover</i> Weekly Sales (JLRWS)	<i>Hexa</i> Weekly Sales (HWS)	<i>Zest</i> Weekly Sales (ZWS)	<i>Nexon</i> Weekly Sales (NWS)	<i>Safari Storme</i> Weekly Sales (SSWS)
220	121217	28	23	138	148	50

10	121217	49	34	164	115	38
122	121217	40	141	123	37	88
16	121217	13	25	127	158	174
28	121217	12	122	116	128	57
-	-	-	-	-	-	-
-	-	-	-	-	-	-

(LO 3.6)

1 <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC#LanguageManualORC-ORCFileFormat>

2 <http://www.semantikoz.com/blog/orc-intelligent-big-data-file-format-hadoop-hive/>

Note:

○○● Level 1 & Level 2 category

○●● Level 3 & Level 4 category

●●● Level 5 & Level 6 category

Chapter 4

MapReduce, Hive and Pig

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 4.1 Get understanding of MapReduce, map tasks using the key-value store, grouping by keys, reduce tasks using combiners, and coping with node failures
- LO 4.2 Get knowledge of composing MapReduce programs for calculations, such as counting, summing, and algorithms for relational algebraic operations, projections, unions, intersections, natural joins, grouping, aggregation operations and the matrix multiplication
- LO 4.3 Get understanding of Hive, architecture, installation, comparison of Hive data store with traditional databases
- LO 4.4 Apply HiveQL for querying, sorting, aggregating, querying scripts, MapReduce Joins and sub-queries
- LO 4.5 Get knowledge of Pig architecture, Grunt shell commands, data model, Pig Latin, developing scripts and extensibility using UDFs.

RECALL FROM EARLIER CHAPTERS

Hadoop stores and processes data on large clusters (thousands of nodes) of commodity hardware. Hadoop is a software framework for writing distributed applications. Hadoop processes Big Data (multi-terabyte datasets) in parallel and in a reliable and fault-tolerant way. The Hadoop distribution model is a method in which both computations and the data distribute and handle large-sized data.

(Section 2.3)

MapReduce functions are an integral part of the Hadoop physical organization. MapReduce is a programming model for the distributed computing environment. Applications using MapReduce v2, process huge amounts of data, in parallel, on a large number of data nodes reliably (Sections 2.4 and 2.5).

Figure 2.2 showed the main components and the ecosystem components of Hadoop, such as AVRO, Zookeeper, Ambari, HBase, Hive, Pig and Mahout (Section 2.6).

This chapter focuses on details of MapReduce, Hive and Pig programming and their use in Big Data applications.

4.1 | INTRODUCTION

The data processing layer is the application support layer, while the application layer is the data consumption layer in Big-Data architecture design (Figure 1.2). When using HDFS, the Big Data processing layer includes the APIs of Programs such as **MapReduce** and **Spark**.

The application support layer includes HBase which creates column-family data store using other formats such as key-value pairs or JSON file (Section 3.3.3). HBase stores and processes the columnar data after translating into MapReduce tasks to run in HDFS.

The support layer also includes Hive which creates SQL-like tables. Hive stores and processes table data after translating it into MapReduce tasks to run in HDFS. Hive creates SQL-like tables in Hive shell. Hive uses HiveQL processes queries, ad hoc (unstructured) queries, aggregation functions and summarizing functions, such as functions to compute maximum, minimum, average of selected or grouped datasets. HiveQL is a restricted form of SQL.

The support layer also includes Pig. Pig is a data-flow language and an execution framework (Section 2.6.4). Pig enables the usage of relational algebra in HDFS. MapReduce is the processing framework and YARN is the resource managing framework (Section 2.6.5).

Figure 4.1 shows Big Data architecture design layers: (i) data storage, (ii) data processing and data consumption, (iii) support layer APIs for MapReduce, Hive and Pig running on top of the HDFS Data Store, and (v) application tasks. Pig is a dataflow language, which means that it defines a data stream and a series of transformations.

Hive and Pig are also part of the ecosystem (Figure 4.1). Big Data storage and application-support APIs can use Hive and Pig for processing data at HDFS. Processing needs mapping and finding the source file for data. File is in the distributed data store. Requirement is to identify the needed data-block in the cluster. Applications and APIs run at the data nodes stored at the blocks.

The smallest unit of data that can be stored or retrieved from the disk is a block. HDFS deals with the data stored in blocks. The Hadoop application is responsible for distributing the data blocks across multiple nodes. The tasks, therefore, first convert into map and reduce tasks. This requirement arises because the mapping of stored values is very important. The number of map tasks in an application is handled by the number of blocks of input files.

Suppose stored files have key-value pairs. Mapping tells us whether the key is in file or in the value store, in a particular cluster and rack. Reduce task uses those values for further processing such as counting, sorting or aggregating.

Application sub-task assigned for processing needs only the outputs of reduce tasks. For example, a query needs the required response for a data store. In Example 2.3, a sub-task may just need total daily sales of specific chocolate flavours to compute the analytics and data visualization.

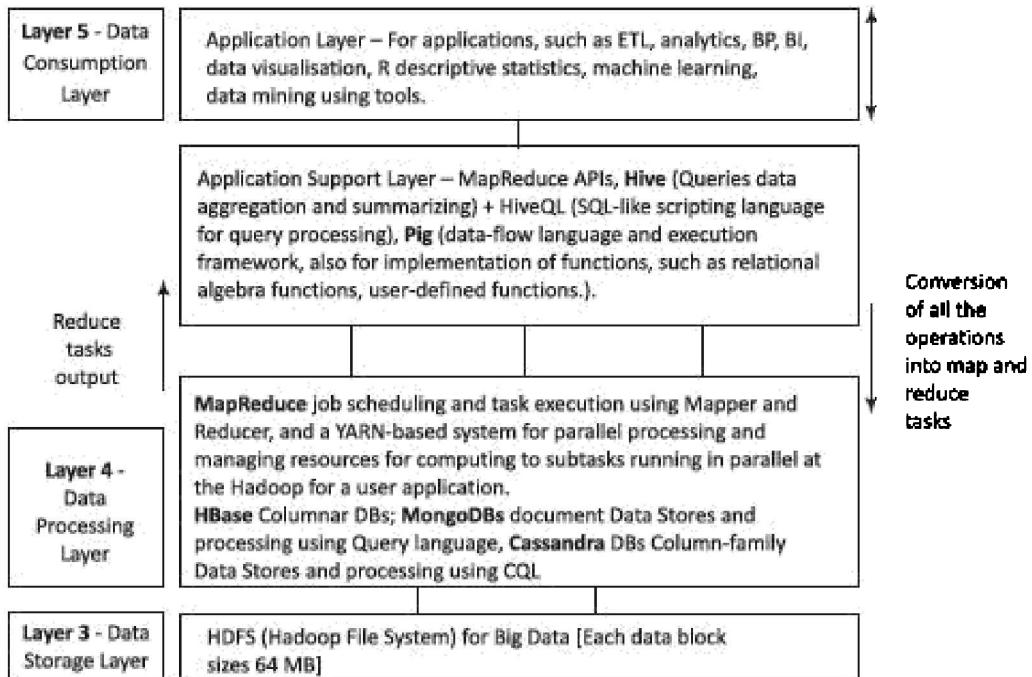


Figure 4.1 Big Data architecture design layers

A reader must learn the following new selected key terms and their meanings besides the key terms given in the previous three chapters.

MapReduce programming model refers to a programming paradigm for processing Big Data sets with a parallel and distributed environment using map and reduce tasks.

YARN refers to provisioning of running and scheduling parallel programs for map and reduce tasks and allocating parallel processing resources for computing sub-tasks running in parallel at the Hadoop for a user application. The YARN resources management enables large-scale data analytics using multiple machines (data nodes) in the HDFS cluster.

Script refers to a small program (codes up to few thousand lines of code) in a language used for purposes such as query processing, text processing, or refers to a small code written in a dynamic high-level general-purpose language, such as Python or PERL.

SQL-like scripting language means a language for writing script that processes queries similar to SQL. SQL lets us: (i) write structured queries for processing in DBMS, (ii) create and modify schema, and control the data access, (iii)

create client for sending query scripts, and create and manage server databases, and (iv) view, query and change (update, insert or append or delete) databases.

NoSQL DBs refers to DBs with no prior fixed schema, schema-less models, and databases which possess increasing flexibility for data manipulation.

NoSQL data model refers to ones offering relaxation in one or more of the ACID properties (Atomicity, Consistency, Isolation and Durability) of the database. A theorem known as CAP (Consistency, Availability and Partitions) states that out of three properties, at least two must be present for the application/service/process. NoSQL relies upon another model known as the BASE model. This model has three principles: Basic availability (the availability of data even in the presence of multiple failures), Soft state (data consistency is the developer's problem and should not be handled by the database), Eventual consistency (when no new changes occur on existing data, eventually all accesses to that data will return the last updated value).

Data-architecture patterns refer to formats used in NoSQL DBs. The examples are Key-Value Data Stores, Object Data Stores, Column family Big Data Stores, Tabular Data Stores and Document Stores.

Key-Value Data Store refers to a simplest way to implement a schema-less database. A string called key maps to values in a large data string or BLOB (basic large object). Key-value stores use primary key access. Therefore, the storage easily scales up and data retrievals are fast.

Object Data Store refers to a repository which stores the (i) objects (such as files, images, documents, folders and business reports), (ii) system metadata which provides information such as filename, creation_date, last_modified, language_used (such as Java, C, C#, C++, Smalltalk, Python), access_permissions, supported Query languages, and (iii) Custom metadata which provides information such as subject, category and sharing permission.

Tabular Data Store refers to table, column-family or BigTable like Data Store.

Column family Big Data store refers to a storage in logical groups of column families. The storage may be similar to columns of sparse matrix. They use a pair of row and column keys to access the column fields.

BigTable Data Store is a popular column-family based Data Store. Row key,

column key and timestamp uniquely identify a value. Google BigTable, HBase and Cassandra DBs use the BigTable Data Store model.

Document Store means a NoSQL DB which stores hierarchical information in a single unit called document. Document stores data in nested hierarchies; for example in XML document object model, JSON formats data model or machine-readable data as one BLOB.

Tuple means an ordered list of elements. An n-tuple relates to set theory, a collection (sequence) of “n” elements. Tuples implement the records.

Collection means a well-defined collection of distinct objects in a set, the objects of a set are the elements. That also means a store within a single DB to achieve a single purpose. A collection may be analogous to a table of RDBMS. A collection in a database also refers to storage of a number of documents. A collection may store documents which do not have the same fields. Thus, documents in the collection are schema-less. Thus, it is possible to store documents of varying structures in a collection.

Aggregate refers to collection of data sets in the key value, column family or BigTable data stores which usually require sequential processing.

Aggregation function refers to a function to find counts, sum, maximum, minimum, other statistical or mathematical function using a collection of datasets, such as column or column-family.

Sequence refers to an enumerated collection of objects, (the repetitions can be there) which contain members similar to a set. Sequence length equals the number of elements (can also be infinite). Sequence should reflect an order which matters, unlike a set.

Document refers to a container for the number of collections. The container can be a unit of storing data in a database, such as MongoDB.

Projection refers to a unary operation (single input or operand) written as $\Pi_{attr_1, attr_2, \dots, attr_n}$ where $(attr_1, attr_2, \dots, attr_n)$ is a set of n attribute names. Projection returns a set obtained by selecting only the n attributes. A generalized projection includes a method using attribute values. $\Pi_{student_Id, sum(GPA), sum(SGPA)}$.

Natural join is where two tables join based on all common columns. Both the tables must have the same column name and the data type.

Inner join is the default natural join. It refers to two tables that join based on common columns mentioned using the ON clause. Inner Join returns all rows from both tables if the columns match.

Node refers to a place for storing data, data block or read or write computations.

Data center in a DB refers to a collection of related nodes. Many nodes form a data center or rack.

Cluster refers to a collection of many nodes.

Keyspace means a namespace to group multiple column families, especially one per partition.

Indexing to a field means providing reference to a field in a document of collections that support the queries and operations using that index. A DB creates an index on the `_id` field of every collection.

This chapter describes MapReduce programming, Hive and Pig APIs in the MapReduce programming model and the HDFS data storage environment. Section 4.2 describes the MapReduce paradigm, map tasks using key-value pairs, grouping by keys and reduce tasks using partitioning and combiners in the application execution framework. Section 4.3 describes algorithms for using MapReduce. Section 4.4 describes Hive, architecture, installation and comparison with traditional databases. Section 4.5 describes HiveQL, querying the data, sorting and aggregating, scripts, joins and sub-queries. Section 4.6 introduces Pig architecture, grunt shell commands, data model, Pig Latin, developing scripts and extensibility using UDFs.

4.2 • MAPREDUCE MAP TASKS, REDUCE TASKS AND MAPREDUCE EXECUTION

Big data processing employs the MapReduce programming model. A Job means a MapReduce program. Each job consists of several smaller units, called MapReduce tasks. A software execution framework in MapReduce programming defines the

LO 4.1

MapReduce, map tasks using the key-value store, grouping by keys, reduce tasks using combiners, and coping with node failures

parallel tasks. The tasks give the required result. The Hadoop MapReduce implementation uses Java framework.

Figure 4.2 shows MapReduce programming model.

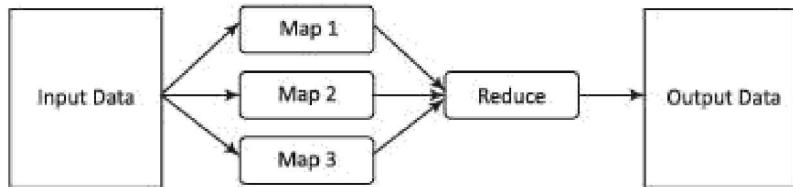


Figure 4.2 MapReduce Programming Model

The model defines two important tasks, namely Map and Reduce. Map takes input data set as pieces of data and maps them on various nodes for parallel processing. The reduce task, which takes the output from the maps as an input and combines those data pieces into a smaller set of data. A reduce task always run after the map task (s).

Many real-world situations are expressible using this model. Such Model describes the essence of MapReduce programming where the programs written are automatically parallelize and execute on a large cluster.

EXAMPLE 4.1

How can a car company quickly compute an aggregation function using the number of cars of a specific car-model sold at the company showrooms as input? Use the concept of division of an application task into a number of sub-tasks (running in parallel).

SOLUTION

The company's showrooms sell a specific model. Assume the analysis requires us to find the aggregate number N. The N computes by counting the number of cars of that model which have been sold over a specific period (Practice Exercise 3.3). N is a very large number. The application process will require a long time to count this sequentially from the sales figures.

The programming-model splits the application task into number of n sub-tasks, running in parallel. Each sub-task thus takes up and counts N/n

sales entries for the car-model. Each sub-task fetches the items containing information of car sales separately. The results of all the application sub-tasks later combine at the end to send the result to the application. High volumes of data (Big Data) need the splitting and parallel processing of the tasks.

MapReduce simplifies software development practice. It eliminates the need to write and manage parallel codes. The YARN resource managing framework takes care of scheduling the tasks, monitoring them and re-executing the failed tasks. Following explains the concept:

EXAMPLE 4.2

How does MapReduce enable query processing quickly in Big Data problems?

SOLUTION

MapReduce provides two important functions for query processing. The distribution of task based on user's query to various nodes within the cluster is the first function. The other function is organizing and reducing the results from each node into a cohesive answer to a query.

The input data is in the form of an HDFS file. The output of the task also gets stored in the HDFS. The compute nodes and the storage nodes are the same at a cluster, that is, the MapReduce program and the HDFS are running on the same set of nodes. This configuration results in effectively scheduling of the sub-tasks on the nodes where the data is already present. This results in high efficiency due to reduction in network traffic across the cluster.

A user application specifies locations of the input/output data and translates into map and reduces functions. A job does implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, together comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker, which then assumes the responsibility of distributing the software/configuration to the slaves by scheduling tasks, monitoring them, and provides status and diagnostic information to the job-client. Figure 4.3 shows MapReduce process when a

client submits a job, and the succeeding actions by the JobTracker and TaskTracker.

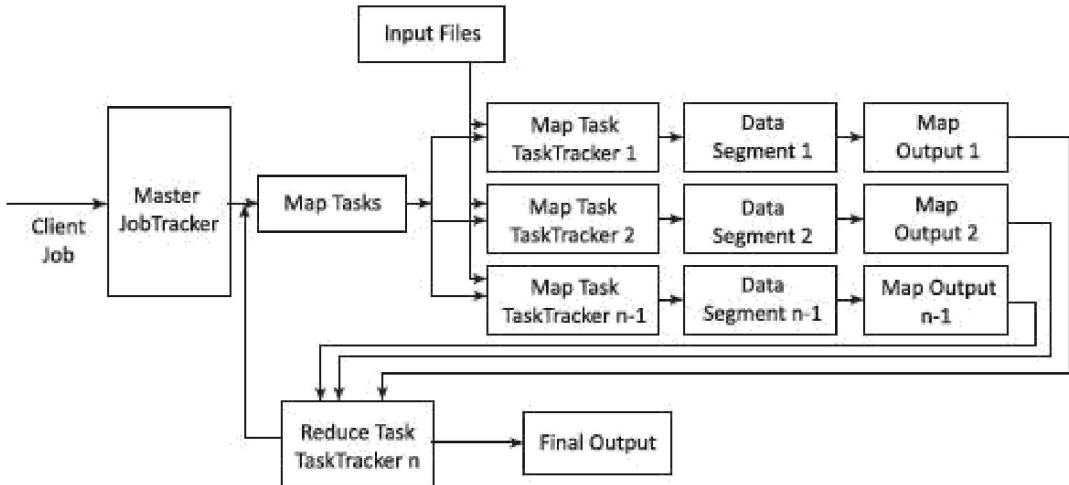


Figure 4.3 MapReduce process on client submitting a job

JobTracker and Task Tracker MapReduce consists of a single master JobTracker and one slave TaskTracker per cluster node. The master is responsible for scheduling the component tasks in a job onto the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

The data for a MapReduce task is initially at input files. The input files typically reside in the HDFS. The files may be line-based log files, binary format file, multi-line input records, or something else entirely different. These input files are practically very large, hundreds of terabytes or even more than it.

Most importantly, the MapReduce framework operates entirely on key, value-pairs. The framework views the input to the task as a set of (key, value) pairs and produces a set of (key, value) pairs as the output of the task, possibly of different types (Section 2.4.2). Example 2.3 explained the process of converting input files into key-values.

4.2.1 Map-Tasks

Map task means a task that implements a `map()`, which runs user application codes for each key-value pair (k_1, v_1). Key k_1 is a set of keys. Key k_1 maps to a

group of data values (Section 3.3.1). Values **v1** are a large string which is read from the input file(s).

The output of `map()` would be zero (when no values are found) or intermediate key-value pairs (**k2, v2**). The value **v2** is the information for the transformation operation at the reduce task using aggregation or other reducing functions.

Reduce task refers to a task which takes the output **v2** from the map as an input and combines those data pieces into a smaller set of data using a *combiner*. The reduce task is always performed after the map task.

The *Mapper* performs a function on individual values in a dataset irrespective of the data size of the input. That means that the Mapper works on a single data set. Figure 4.4 shows logical view of functioning of `map()`.

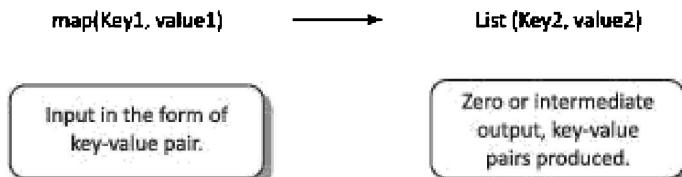


Figure 4.4 Logical view of functioning of `map()`

Hadoop Java API includes `Mapper` class. An abstract function `map()` is present in the `Mapper` class. Any specific `Mapper` implementation should be a subclass of this class and overrides the abstract function, `map()`.

The Sample Code for `Mapper` Class

```
public class SampleMapper extends Mapper<k1, V1, k2, v2>
{
    void map (k1 key, V1 value, Context context) throws IOException,
    InterruptedException
    {..}
}
```

Individual Mappers do not communicate with each other.

Number of Maps The number of maps depends on the size of the input files, i.e., the total number of blocks of the input files. Thus, if the input files are of 1TB in size and the block size is 128 MB, there will be 8192 maps. The number of map task N_{map} can be explicitly set by using `setNumMapTasks(int)`. Suggested number

is nearly 10-100 maps per node. N_{map} can be set even higher.

4.2.2 Key-Value Pair

Each phase (Map phase and Reduce phase) of MapReduce has key-value pairs as input and output. Data should be first converted into key-value pairs before it is passed to the Mapper, as the Mapper only understands key-value pairs of data (Section 3.3.1).

Key-value pairs in Hadoop MapReduce are generated as follows:

InputSplit - Defines a logical representation of data and presents a Split data for processing at individual map().

RecordReader - Communicates with the InputSplit and converts the Split into records which are in the form of key-value pairs in a format suitable for reading by the Mapper. RecordReader uses TextInputFormat by default for converting data into key-value pairs. RecordReader communicates with the InputSplit until the file is read.

Figure 4.5 shows the steps in MapReduce key-value pairing.

Generation of a key-value pair in MapReduce depends on the dataset and the required output. Also, the functions use the key-value pairs at four places: map() input, map() output, reduce() input and reduce() output.

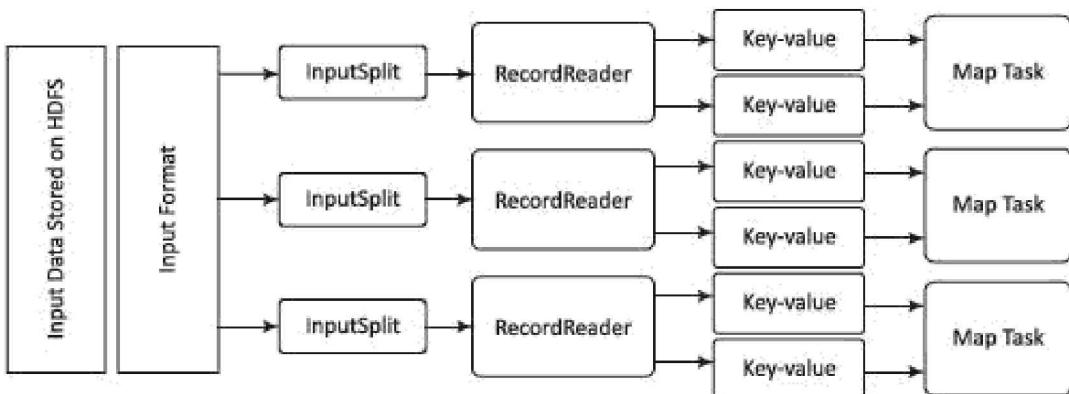


Figure 4.5 Key-value pairing in MapReduce

4.2.3 Grouping by Key

When a map task completes, Shuffle process aggregates (combines) all the

Mapper outputs by grouping the key-values of the Mapper output, and the value **v2** append in a list of values. A “Group By” operation on intermediate keys creates **v2**.

Shuffle and Sorting Phase

Here, all pairs with the same group key (**k2**) collect and group together, creating one group for each key. So, the Shuffle output format will be a List of <**k2**, List (**v2**)>. Thus, a different subset of the intermediate key space assigns to each reduce node. These subsets of the intermediate keys (known as “partitions”) are inputs to the reduce tasks.

Each reduce task is responsible for reducing the values associated with partitions. HDFS sorts the partitions on a single node automatically before they input to the Reducer.

4.2.4 Partitioning

The Partitioner does the partitioning. The partitions are the semi-mappers in MapReduce. Partitioner is an optional class. MapReduce driver class can specify the Partitioner. A partition processes the output of map tasks before submitting it to Reducer tasks. Partitioner function executes on each machine that performs a map task. Partitioner is an optimization in MapReduce that allows **local partitioning** before reduce-task phase. Typically, the same codes implement the Partitioner, Combiner as well as reduce() functions. Functions for Partitioner and sorting functions are at the mapping node. The main function of a Partitioner is to split the map output records with the same key.

4.2.5 Combiners

Combiners are semi-reducers in MapReduce. Combiner is an optional class. MapReduce driver class can specify the combiner. The combiner() executes on each machine that performs a map task. Combiners optimize MapReduce task that locally aggregates before the shuffle and sort phase. Typically, the same codes implement both the combiner and the reduce functions, combiner() on map node and reducer() on reducer node.

The main function of a Combiner is to consolidate the map output records with the same key. The output (key-value collection) of the combiner transfers over the network to the Reducer task as input.

This limits the volume of data transfer between map and reduce tasks, and thus reduces the cost of data transfer across the network. Combiners use grouping by key for carrying out this function. The combiner works as follows:

- (i) It does not have its own interface and it must implement the interface at `reduce()`.
- (ii) It operates on each map output key. It must have the same input and output key-value types as the Reducer class.
- (iii) It can produce summary information from a large dataset because it replaces the original Map output with fewer records or smaller records.

4.2.6 Reduce Tasks

Java API at Hadoop includes Reducer class. An abstract function, `reduce()` is in the Reducer. Any specific Reducer implementation should be subclass of this class and override the abstract `reduce()`.

Reduce task implements `reduce()` that takes the Mapper output (which shuffles and sorts), which is grouped by key-values (k_2, v_2) and applies it in parallel to each group. Intermediate pairs are at input of each Reducer in order after sorting using the key. Reduce function iterates over the list of values associated with a key and produces outputs such as aggregations and statistics. The reduce function sends output zero or another set of key-value pairs (k_3, v_3) to the final the output file. Reduce: $\{(k_2, \text{list}(v_2)) \rightarrow \text{list}(k_3, v_3)\}$

Sample Code for Reducer Class

```
public class ExampleReducer extends Reducer<k2, v2, k3, v3>
{
    void reduce (k2 key, Iterable<v2> values, Context context) throws
    IOException, InterruptedException
    {...}
}
```

4.2.7 Details of MapReduce Processing Steps

Execution of MapReduce job does not consider how the distributed processing implements. Rather, the execution involves the formatting (transforming) of data at each step.

Figure 4.6 shows the execution steps, data flow, splitting, partitioning and sorting on a map node and reducer node.

Example 2.3 described sales data of the five types of chocolates in a large number of ACVMs (Automatic Chocolate Vending Machines). The example gave answers to the following: (i) how hourly data log for each flavor sales on large number of ACVMs save using HDFS, (ii) how the sample of data-collect in a file each for 0-1,1-2, ... 12-13,13-14, 15-16, ... for up to 23-24 specific hour sales, (iii) what will be output streams of map tasks which are the input streams to the reduce() tasks, and (iv) what will be the Reducer outputs.

Let us explore another example, Automotive Components and Predictive Automotive Maintenance Services (ACPAMS). ACPAMS is an application of (Internet) connected cars which renders services to customers for maintenance and servicing of (Internet) connected cars [Chapter 1 Example 1.6(ii)].

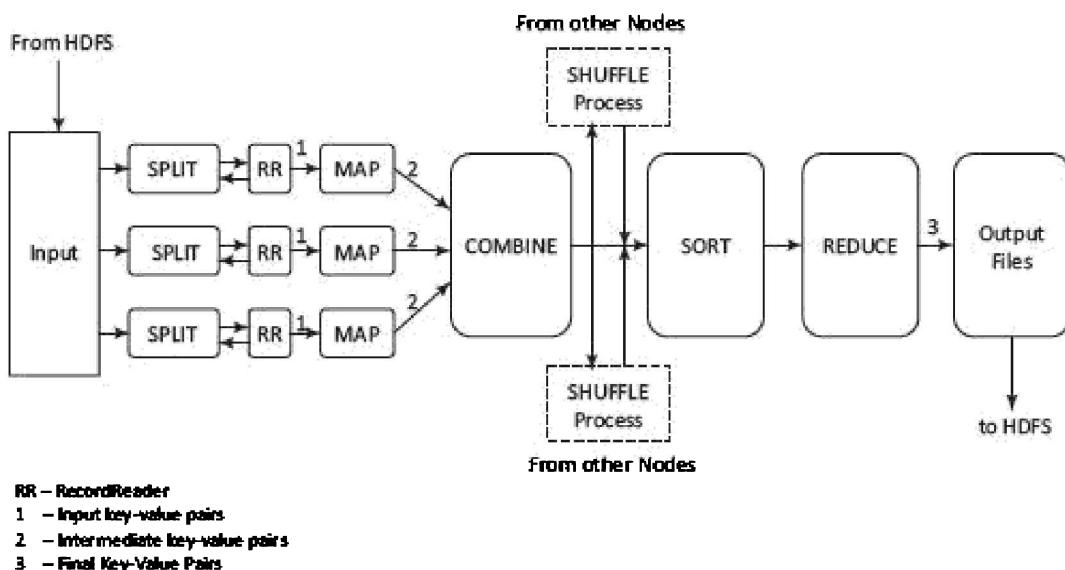


Figure 4.6 MapReduce execution steps

EXAMPLE 4.1

Describe the MapReduce processing steps of a task of ACPAMS.

SOLUTION

Figure 4.7 shows processing steps of an ACPAMS task in MapReduce. Steps

are inputs, mapping, combining, shuffling and reducing for the output to application task.

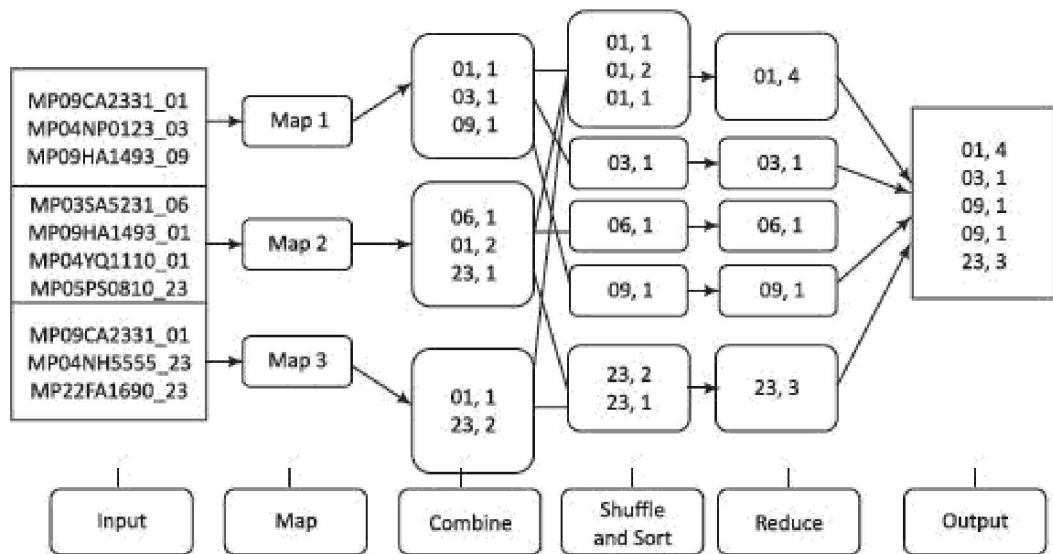


Figure 4.7 MapReduce processing steps in ACPAMS application

The *application* submits the inputs. The execution framework handles all other aspects of distributed processing transparently, on clusters ranging from a single node to a few thousand nodes. The aspects include scheduling, code distribution, synchronization, and error and fault handling.

The following example explains how the ACPAMS company receives alerts/messages.

EXAMPLE 4.4

Describe the MapReduce processing steps to illustrate how the ACPAMS receives alerts/messages.

SOLUTION

The ACPAMS Company can receive the alerts/messages every hour from several sensors of the automotive components installed in the number of cars. A server maps the keys for filling fuel, changing of the coolant, etc. It requires a lot of time to scan the hourly maintenance log sequentially

because there are a million cars registered for the ACPAMS service. Each car is equipped with nearly 50 sensor-based components sending alerts/message every minute. By contrast, the MapReduce programmer can split the application task among multiple subtasks, say one hundred sub-tasks, and each sub-task processes the data of a selected set of a Service users.

The results of all the sub-tasks aggregate and produce the final result for hourly maintenance requirement of each component of the cars registered at ACPAMS Company. Finally, the aggregated hourly results appear from the hourly log of transactions files at Hadoop data nodes. The whole system maintains transparency, without knowing the presence of a distributed parallel system processing data of a hundred million records. Table 4.1 gives examples of assigning Ids to alerts/messages.

Table 4.1 Examples of Alert/MessageId (say, total 50 Ids, i.e., one for each sensor component)

Alert/ MessageId	Details
01	Fuel fill-in request
02	Filter requirement
03	Alignment and balancing requirement
04	Engine oil less
05	Coolant Change request

Assumption: Let 60 files for each hour create every day. The files are file_1, file_2, ..., file_60. Each file saves as a key-value pair at the large number of the company's machines. The log contains information in the following format:

maintenance service Id:(<CarRegistrationNumber>_<alert/messsageId>)

Thus, every line entry becomes the key, and the value will be the instance number.

Sample data of one of such file out of 60 files (file_10) saved as hour-maintenance-service log for the maintenance service during 15:00-16:00 will be as follows:

MP09CA2331_01

MP04NP0123_03

MP09HA1493_01

MP03SA5231_06

.

.

MP09CA2331_04

MP09CC4614_01

.

- (i) The input files are using *NLineInputFormat* input format.
- (ii) Map tasks will map the input streams of key values at files, file_1, file_2, file_59, file_60 every hour.
- (iii) The map function extracts the alert/message Id from each line of the input files. They are the values after underscore in each line.
- (iv) The resulting $1 \text{ million} \times 50$ key-value pairs (since there are 50 sensors assumed per car) map each hour with keys for RegistrationNumber_N ($N = 1 \text{ to } 50$). The output stream from Mapper will be as follows:

(01, 1), (03, 1), (01, 1), ..., ..., ...,

The (key, value) contains (alert/message Id, the instance number)

Mapper in ACPAMS

Input	Set of data	MP09CA2331_01 MP04NPO123_03 MP09HA1493_09 MP03SA5231_06 MP09HA1493_01 MP04YQ1110_01 MP05PS0810_23 MP09CA2331_01 MP04NH5555_23 MP22FA1690_23
Output	Convert into another set of data (Key, Value) Key – Alert/Message Id Value – Number of instance	(01,1), (03,1), (09,1), (06,1), (01,1), (01,1), (23,1), (01,1), (23,1), (23,1)

Combiner in ACPAMS

Input	Key-Value Pairs (output of Mapper)	(01,1), (03,1), (09,1), (06,1), (01,1), (01,1), (23,1), (01,1), (23,1), (23,1)
Output	Key-Value Pairs (Group-By-key at node itself)	(01,1), (03,1), (09,1), (06,1), (01,2), (23,1), (01,1), (23,2)

Shuffle in ACPAMS

Input	Key-Value Pairs (output of Combiner)	(01,1), (03,1), (09,1), (06,1), (01,2), (23,1), (01,1), (23,2)
Output	Key-Value Pairs (moved to other nodes based on key value)	(01,1), (01,2), (01,1) (03,1) (09,1) (06,1) (23,1), (23,2)

Sort in ACPAMS

Input	Key-Value Pairs (output of Shuffler)	(01,1), (01,2), (01,1) (03,1) (09,1) (06,1) (23,1), (23,2)
Output	Key-Value Pairs (Sort as per key)	(01,1), (01,2), (01,1) (03,1) (06,1) (09,1) (23,1), (23,2)

Reducer – Takes the output from Map (after the sort phase) as an input and combines the data tuples into a smaller set of tuples. Following are some examples of Reduce function in Alert/Message Count.

Input (output of map())	Set of Tuples	(01,1), (01,2), (01,1) (03,1) (06,1) (09,1) (23,1), (23,2)
Output	Converts into a smaller set of tuples (By using the aggregate function)	(01,4) (03,1) (06,1) (09,1) (23,3)

The output of the reduce(), which is the final result of MapReduce Job provides a number of alert or messages at an hourly basis for the complaint raised by each component of the cars registered at the ACPAMS Company. Then the analyst decides which components need maintenance on high to low priority. The report of ACAMPS helps the company in improving the manufacturing of its car components.

The following example gives pseudocodes for an algorithm:

EXAMPLE 4.5

Write pseduocodes for MapReduce algorithm for the ACPAMS.

SOLUTION

Figure 4.8 gives pseudocodes for the ACPAMS algorithm in MapReduce.

```

class Mapper {

    method Map (file id a; file f) {
        for all term i ∈ file f do {
            t = Substring (i, 2, After_)
            Emit (term t, count 1)}}}

class Reducer {

    method Reduce (term t, counts [c1, c2,...]) {
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do {
            sum ← sum + c}
        Emit (term t, count sum)}
    }

```

Figure 4.8 Pseudocodes for the ACPAMS algorithm in MapReduce

Emit() function is for output in MapReduce. The Mapper emits an intermediate key-value pair for each alert/message in a document. The Reducer sums up all counts for each alert/message.

4.2.8 Coping with Node Failures

The primary way using which Hadoop achieves fault tolerance is through restarting the tasks. Each task nodes (TaskTracker) regularly communicates with the master node, JobTracker. If a TaskTracker fails to communicate with the JobTracker for a pre-defined period (by default, it is set to 10 minutes), a task node failure by the JobTracker is assumed. The JobTracker knows which map and reduce tasks were assigned to each TaskTracker.

Node failure occurs when the TaskTracker fails to communicate with the JobTracker for a pre-defined period. The JobTracker restarts the TaskTracker for coping with the node failure.

If the job is currently in the mapping phase, then another TaskTracker will be assigned to re-execute all map tasks previously run by the failed TaskTracker. All completed map tasks also need to be assigned for re-execution if they belong to incomplete jobs. This is because the intermediate results residing in the failed TaskTracker file system may not be accessible to the reduce task.

If the job is in the reducing phase, then another TaskTracker will re-execute all reduce tasks that were in progress on the failed TaskTracker.

Once reduce tasks are completed, the output writes back to the HDFS. Thus, if a TaskTracker has already completed nine out of ten reduce tasks assigned to it, only the tenth task must execute at a different node.

Map tasks are slightly more complicated. A node may have completed ten map tasks but the Reducers may not have copied all their inputs from the output of those map tasks. Now if a node fails, then its Mapper outputs are inaccessible. Thus, any complete map tasks must also be re-executed to make their results available to the remaining reducing nodes. Hadoop handles all of this automatically. MapReduce does not use any task identities to communicate between nodes or which reestablishes the communication with other task node. Each task focuses on only its own direct inputs and knows only its own outputs. The failure and restart processes are clean and reliable.

The failure of JobTracker (if only one master node) can bring the entire process down; Master handles other failures, and the MapReduce job eventually completes. When the Master compute-node at which the JobTracker is executing fails, then the entire MapReduce job must restart. Following points summarize the coping mechanism with distinct Node Failures:

(i) Map TaskTracker failure:

- Map tasks completed or in-progress at TaskTracker, are reset to idle on failure
- Reduce TaskTracker gets a notice when a task is rescheduled on another TaskTracker

(ii) Reduce TaskTracker failure:

- Only in-progress tasks are reset to idle

(iii) Master JobTracker failure:

- Map-Reduce task aborts and notifies the client (in case of one master node).

Self-Assessment Exercise linked to LO 4.1

1. Show MapReduce process diagrammatically to depict a client submitting a job, the workflow of JobTracker and TaskTracker, and TaskTrackers creating the outputs.
2. Assume an input file size of 10 TB and a data block size of 128 MB. How many map tasks are required? Assume that each node does 100 maps. How many nodes are involved in processing? How will you change the number of map tasks per node to 120 using a Java statement?
3. Explain function of Group By, partitioning and combining using one example for each.
4. How does the data convert to (key, value) pairs before passing to the Mapper? How do the InputSplit and RecordReader function?
5. How are the failures of Map TaskTracker, Reduce TaskTracker and Master JobTracker handled in MapReduce?
6. How does the execution framework handle all aspects of distributed processing after a client node submits the job to the designated node of a cluster (the JobTracker)? Explain the concept using a diagram.

4.3 COMPOSING MAPREDUCE FOR CALCULATIONS AND ALGORITHMS

The following subsections describe the use of MapReduce program composition in counting and summing, algorithms for relational algebraic operations, projections, unions, intersections, natural joins, grouping and aggregation, matrix multiplication and other computations.

LO 4.2

Composing MapReduce programs for calculations, such as counting and summing; and algorithms for relational algebraic operations, such as projections, unions, intersections, natural joins, grouping, aggregation operations and matrix multiplication

4.3.1 Composing Map-Reduce for Calculations

The calculations for various operations compose are:

Counting and Summing Assume that the number of alerts or messages generated during a specific maintenance activity of vehicles need counting for a month. Figure 4.8 showed the pseudocode using `emit()` in the `map()` of *Mapper* class. *Mapper* emits 1 for each message generated. The reducer goes through the list of 1s and sums them. Counting is used in the data querying application. For example, count of messages generated, word count in a file, number of cars sold, and analysis of the logs, such as number of tweets per month. Application is also in business analytics field.

Sorting Figure 4.6 illustrated MapReduce execution steps, i.e., dataflow, splitting, partitioning and sorting on a map node and reduce on a reducer node. Example 4.3 illustrated the sorting method. Many applications need sorted values in a certain order by some rule or process. *Mappers* just emit all items as values associated with the sorting keys which assemble as a function of items. *Reducers* combine all emitted parts into a final list.

Finding Distinct Values (Counting unique values) Applications such as web log analysis need counting of unique users. Evaluation is performed for the total number of unique values in each field for each set of records that belongs to the same group. Two solutions are possible:

- (i) The *Mapper* emits the dummy counters for each pair of field and `groupId`, and the *Reducer* calculates the total number of occurrences for each such pair.
- (ii) The *Mapper* emits the values and `groupId`, and the *Reducer* excludes the duplicates from the list of groups for each value and increments the counter for each group. The final step is to sum all the counters emitted at the *Reducer*. This requires only one MapReduce job but the process is not scalable, and hence has limited applicability in large data sets.

Collating Collating is a way to collect all items which have the same value of function in one document or file, or a way to process items with the same value of the function together. Examples of applications are producing inverted indexes and extract, transform and load operations.

Mapper computes a given function for each item, produces value of the function as a key, and the item itself as a value. *Reducer* then obtains all item values using group-by function, processes or saves them into a list and outputs

to the application task or saves them.

Filtering or Parsing Filtering or parsing collects only those items which satisfy some condition or transform each item into some other representation. Filtering/parsing include tasks such as text parsing, value extraction and conversion from one format to another. Examples of applications of filtering are found in data validation, log analysis and querying of datasets.

Mapper takes items one by one and accepts only those items which satisfy the conditions and emit the accepted items or their transformed versions. *Reducer* obtains all the emitted items, saves them into a list and outputs to the application.

Distributed Tasks Execution Large computations divide into multiple partitions and combine the results from all partitions for the final result. Examples of distributed running of tasks are physical and engineering simulations, numerical analysis and performance testing.

Mapper takes a specification as input data, performs corresponding computations and emits results. *Reducer* combines all emitted parts into the final result.

Graph Processing using Iterative Message Passing Graph is a network of entities and

relationships between them. A node corresponds to an entity. An edge joining two nodes corresponds to a relationship. Path traversal method processes a graph. Traversal from one node to the next generates a result which passes as a message to the next traversal between the two nodes. Cyclic path traversal uses iterative message passing.

Web indexing also uses iterative message passing. Graph processing or web indexing requires calculation of the state of each entity. Calculated state is based on characteristics of the other entities in its neighborhood in a given network. (State means present value. For example, assume an entity is a course of study. The course may be Java or Python. Java is a state of the entity and Python is another state.)

A set of nodes stores the data and codes at a network. Each node contains a list of neighbouring node IDs. MapReduce jobs execute iteratively. Each node in an iteration sends messages to its neighbors. Each neighbor updates its state based on the received messages. Iterations terminate on some conditions, such as

completion of fixed maximal number of iterations or specified time to live or negligible changes in states between two consecutive iterations.

Mapper emits the messages for each node using the ID of the adjacent node as a key. All messages thus group by the incoming node. *Reducer* computes the state again and rewrites a node new state.

Cross Correlation Cross-correlation involves calculation using number of tuples where the items co-occur in a set of tuples of items. If the total number of items is N , then the total number of values = $N \times N$. Cross correlation is used in text analytics. (Assume that items are words and tuples are sentences). Another application is in market-analysis (for example, to enumerate, the customers who buy item x tend to also buy y). If $N \times N$ is a small number, such that the matrix can fit in the memory of a single machine, then implementation is straightforward.

Two solutions for finding cross correlations are:

- (i) The *Mapper* emits all pairs and dummy counters, and the *Reducer* sums these counters. The benefit from using combiners is little, as it is likely that all pairs are distinct. The accumulation does not use in-memory computations as N is very large.
- (ii) The *Mapper* groups the data by the first item in each pair and maintains an associative array (“stripe”) where counters for all adjacent items accumulate. The *Reducer* receives all stripes for the leading item, merges them and emits the same result as in the pairs approach.

[Stripe means a set of arrays associated with a dataset or a set of rows that belong to a common key with each row having a number of columns.]

The grouping:

- Generates fewer intermediate keys. Hence, the framework has less sorting to do.
- Greatly benefits from the use of combiners.
- In-memory accumulation possible.
- Enables complex implementations.
- Results in general, faster computations using stripes than “pairs”.

4.3.2 Matrix–Vector Multiplication by MapReduce

Numbers of applications need multiplication of $n \times n$ matrix **A** with vector **B** of dimension **n**. Each element of the product is the element of vector **C** of dimension **n**. The elements of **C** calculate by relation,

$$c_i = \sum_{j=1}^n a_{ij} b_j. \text{ An example of calculations is given below.}$$

Assume $\mathbf{A} = \begin{vmatrix} 1 & 5 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{vmatrix}$ and $\mathbf{B} = \begin{vmatrix} 4 \\ 1 \\ 3 \end{vmatrix}$ (4.1)

$$\text{Multiplication } \mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1 \times 4 + 5 \times 1 + 4 \times 3 \\ 2 \times 4 + 1 \times 1 + 3 \times 3 \\ 4 \times 4 + 2 \times 1 + 1 \times 3 \end{bmatrix}$$

Hence, $\mathbf{C} = \begin{bmatrix} 21 \\ 18 \\ 21 \end{bmatrix}$... (4.2)

Algorithm for using MapReduce: The Mapper operates on **A** and emits row-wise multiplication of each matrix element and vector element ($a_{ij} \times b_j \forall i$). The Reducer executes sum() for summing all values associated with each *i* and emits the element c_i . Application of the algorithm is found in linear transformation.

4.3.3 Relational-Algebra Operations

Explained ahead are the some approaches of algorithms for using MapReduce for relational algebraic operations on large datasets.

4.3.3.1 Selection

Example of Selection in relational algebra is as follows: Consider the attribute names (ACVM_ID, Date, chocolate_flavour, daily_sales). Consider relation

$$R = \{(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)\}.$$

Selection $\text{ACVM_ID} \leq 525 (R)$ selects the subset $R = \{(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)\}$.

Selection $\chi_{chocolate_flavour = Oreo}$ selects the subset $\{(524, 12122017, Oreo, 72), (525, 12122017, Oreo, 72), (526, 12122017, Oreo, 72)\}$.

The test() tests the attribute values used for a selection after the binary operation of an attribute with the value(s) or value in an attribute name with value in another attribute name and the binary operation by which each tuple selects. Selection may also return *false* or *unknown*. The test condition then does not select any.

The *Mapper* calls test() for each tuple in a row. When test satisfies the selection criterion then emits the tuple. The *Reducer* transfers the received input tuple as the output.

4.3.3.2 Projection

Example of *Projection* in relational algebra is as follows:

Consider attribute names (ACVM_ID, Date, chocolate_flavour, daily_sales).

Consider relation $R = \{(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)\}$.

Projection $\Pi_{ACVM_ID}(R)$ selects the subset $\{(524)\}$.

Projection, $\Pi_{chocolate_flavour, 0.5 * daily_sales}$ selects the subset $\{(KitKat, 0.5 \times 82), (Oreo, 0.5 \times 72)\}$.

The test() tests the presence of attribute (s) used for projection and the factor by an attribute needs projection.

The *Mapper* calls test() for each tuple in a row. When the test satisfies, the predicate then emits the tuple (same as in selection). The *Reducer* transfers the received input tuples after eliminating the possible duplicates. Such operations are used in analytics.

4.3.3.3 Union

Example of *Union* in relations is as follows: Consider,

$$R1 = \{(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)\}$$

$$R2 = \{(525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)\}$$

and $R3 = \{(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)\}$

Result of *Union* operation between R1 and R3 is:

$$R1 \cup R3 = \{(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (526,$$

12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

The *Mapper* executes all tuples of two sets for union and emits all the resultant tuples. The *Reducer* class object transfers the received input tuples after eliminating the possible duplicates.

4.3.3.4 Intersection and Difference

Intersection Example of Interaction in relations is as follows: Consider,

$$R1 = \{(524, 12122017, Oreo, 72)\}$$

$$R2 = \{(525, 12122017, KitKat, 82)\}$$

and $R3 = \{(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)\}$

Result of Intersection operation between R1 and R3 are

$$R1 \cap R3 = \{(12122017, Oreo)\}$$

The *Mapper* executes all tuples of two sets for intersection and emits all the resultant tuples. The *Reducer* transfers only tuples that occurred twice. This is possible only when tuple includes primary key and can occur once in a set. Thus, both the sets contain this tuple.

Difference Consider:

$$R1 = \{(12122017, KitKat, 82), (12122017, Oreo, 72)\}$$

and $R3 = \{(12122017, KitKat, 82), (12122017, Oreo, 25)\}$

Difference means the tuple elements are not present in the second relation. Therefore, difference set_1 is

$$R1 - R3 = (12122017, Oreo, 72) \text{ and set_2 is } R3 - R1 = (12122017, Oreo, 25).$$

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_1 or set_2 to which a tuple belongs to). The *Reducer* transfers only tuples that belong to set_1.

Symmetric Difference Symmetric difference (notation is $A \Delta B$ (or $A \ominus B$)] is another relational entity. It means the set of elements in exactly one of the two relations A or B. $R3 \ominus R1 = (12122017, Oreo, 25)$.

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_1 or set_2 this tuple belongs to). The *Reducer* transfers only tuples that belong to neither set_1 or set_2.

4.3.3.5 Natural Join

Consider two relations R1 and R2 for tuples a, b and c. Natural Join computes for R1 (a, b) with R2 (b, c). Natural Join is R (a, b, c). Tuples b joins as one in a Natural Join. The *Mapper* emits the key-value pair (b, (R1, a)) for each tuple (a, b) of R1, similarly emits (b, (R2, c)) for each tuple (b, c) of R2.

The *Mapper* is mapping both with Key for b. The *Reducer* transfers all pairs consisting of one with first component R1 and the other with first component R2, say (R1, a) and (R2, c). The output from the key and value list is a sequence of key-value pairs. The key is of no use and is irrelevant. Each value is one of the triples (a, b, c) such that (R1, a) and (R2, c) are present in the input list of values.

The following example explains the concept of join, how the data stores use the INNER Join and NATURAL Join of two tables, and how the Join compute quickly.

EXAMPLE 4.6

An SQL statement “Transactions INNER JOIN KitKatStock ON Transactions.ACVM_ID = KitKatStock.ACVM_ID”; selects the records that have matching values in two tables for transactions of KitKat sales at a particular ACVM. One table is KitKatStock with columns (KitKat_Quantity, ACVM_ID) and second table is Transactions with columns (ACVM_ID, Sales_Date and KitKat_SalesData).

1. What will be INNER Join of two tables KitKatStock and Transactions?
2. What will be the NATURAL Join?

SOLUTION

1. The INNER JOIN gives all the columns from the two tables (thus the common columns appear twice). The INNER JOIN of two tables will return a table with five column: (i) KitKatStock.Quantity, (ii) KitKatStock.KitKat_ACVM_ID, (iii) Transactions.ACVM_ID, (iv) Transactions.KitKat_SalesDate, and (v) Transactions.KitKat_SalesData.
2. The NATURAL JOIN gives all the unique columns from the two tables.

The NATURAL JOIN of two tables will return a table with four columns:

- (i) KitKatStock.Quantity, (ii) KitKatStock.ACVM_ID, (iii) Transactions.KitKat_SalesDate, and (iv) Transactions.KitKat_SalesData.

Values accessible by key in the first table KitKatStock merges with Transactions table accessible by the common key ACVM_ID.

NATURAL JOIN gives the common column once in the output of a query, while INNER JOIN gives common columns of both tables.

Join enables fast computations of the aggregate of the number of chocolates of specific flavour sold.

4.3.3.6 Grouping and Aggregation by MapReduce

Grouping means operation on the tuples by the value of some of their attributes after applying the aggregate function independently to each attribute. A Grouping operation denotes by <grouping attributes> ; <function-list> (R). Aggregate functions are count(), sum(), avg(), min() and max().

Assume $R = \{(524, 12122017, \text{KitKat}, 82), (524, 12122017, \text{Oreo}, 72), (525, 12122017, \text{KitKat}, 82), (525, 12122017, \text{Oreo}, 72), (526, 12122017, \text{KitKat}, 82), (526, 12122017, \text{Oreo}, 72)\}$. Chocolate_flavour ; count ACVM_ID, sum (daily_sales(chocolate_flavour)) will give the output (524, KitKat, sale_month), (525, KitKat, sale_month), and (524, Oreo, sale_month), (525, Oreo, sale_month), for all ACVM_IDs.

The Mapper finds the values from each tuple for grouping and aggregates them. The Reducer receives the already grouped values in input for aggregation.

4.3.4 Matrix Multiplication

Consider matrices named A (i rows and j columns) and B (j rows and k columns) to produce the matrix C(i rows and k columns). Consider the elements of matrices A, B and C as follows:

$$\begin{array}{l} A = \begin{matrix} a_{11} & a_{12} & \dots & a_{1j} \\ a_{21} & a_{22} & \dots & a_{2j} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{ij} \end{matrix} \quad B = \begin{matrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{j1} & b_{j2} & \dots & b_{jk} \end{matrix} \quad C = \begin{matrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \dots & \dots \\ c_{i1} & c_{i2} & \dots & c_{ik} \end{matrix} \end{array} \dots (4.3)$$

$A \cdot B = C$; Each element evaluates as follow:

$$C_{ik} = \text{Sum } (a_{ij} \times b_{jk}) \text{ for } j = 1 \text{ to } n. v_a = a_{ij} \text{ and } v_b = b_{jk} \quad \dots (4.4)$$

First row of C

C first column element = $(a_{11}b_{11} + a_{12}b_{21} + \dots + a_{1j}b_{j1})$. Second column element = $(a_{11}b_{12} + a_{12}b_{22} + \dots + a_{1j}b_{j2})$.

The kth column element = $(a_{11}b_{1k} + a_{12}b_{2k} + \dots + a_{1j}b_{jk})$.

Second row of C

C first column element = $(a_{21}b_{11} + a_{22}b_{21} + \dots + a_{2j}b_{j1})$. Second column element = $(a_{21}b_{12} + a_{22}b_{22} + \dots + a_{2j}b_{j2})$.

The kth column element = $(a_{21}b_{1k} + a_{22}b_{2k} + \dots + a_{2j}b_{jk})$.

The ith row of C

C first column element = $(a_{i1}b_{11} + a_{i2}b_{21} + \dots + a_{ij}b_{j1})$. Second column element = $(a_{i1}b_{12} + a_{i2}b_{22} + \dots + a_{ij}b_{j2})$. The kth column element = $(a_{i1}b_{1k} + a_{i2}b_{2k} + \dots + a_{ij}b_{jk})$.

Consider two solutions of matrix multiplication.

Matrix Multiplication with Cascading of Two MapReduce Steps

Table 4.2 gives the names, attributes, relations R_A , R_B and R_C , tuples in A, B, C, natural Join of R_A and R_B , keys and values, and seven steps for multiplication of A and B.

Table 4.2 Seven steps for multiplication of A and B for cascading of two MapReduce Steps

Step	Step description	Matrix A	Matrix B	Matrix C = A \cdot B
1	Name	A	B	C
2	Specify attributes of (Key, Value pairs of each element [row number, column number, value])	(I, J, v _a)	(J, K, v _b)	(I, K, v _c)
3	Specify relations	$R_A = A(I, J, v_a)$	$R_B = B(J, K, v_b)$	$R_C = C(I, K, v_c)$

4	Consider tuples of A, B and C	(i, j, a _{ij})	(j, k, b _{jk})	(i, k, c _{ik})
5	Find natural Join of R _A and R _B = Matrix elements (a _{ij} , b _{jk}) [j is common in both]	-	-	tuples (i, j, k, v _a , v _b)
6	Get tuples for finding Product C			Four-component tuple (i, j, k, v _a × v _b)
7	Grouping and aggregation of tuples with attributes I and K			<I, K> ; SUM (v _a × v _b)

The product A.B = Natural join of tuples in the relations R_A and R_B followed by grouping and aggregation. Natural Join of A (I, J, v_a) and B (J, K, v_b), having only attribute J in common = Tuples (i, j, k, v_a, v_b) from each tuple (i, j, v_a) in A and tuples (j, k, v_b) in B.

1. *MapReduce tasks for Steps 5 and 6:* Five-component tuple represents the pair of matrix elements

(a_{ij}, b_{jk}). Requirement is product of these elements. That means four-component tuple (i, j, k, v_a × v_b),

from equation (4.4) for elements C_{ik} = Sum (a_{ij}. b_{jk})_{j=1 to j}.

- (a) Mapper Function: (i) *Mapper* emits the key-value pairs (j, (A, i, a_{ij})) for each matrix element a_{ij}, and (ii) *Mapper* emits the key-value pair (j, (B, k, b_{jk})) for each matrix element a_{ij}.

- (b) Reduce Function: Consider the tuples of A = (A, i, a_{ij}) for each key j, consider tuples of

B = (B, k, b_{jk}) for each key j. Produce a key-value pair with key equal to (i, k) and

value = a_{ij} × b_{jk}. A and B are just the names, may be represented by 0101 and 1010.

2. *Next MapReduce Steps 7:* Perform <I, K> ; SUM (v_a × v_b). That means do grouping and aggregation, with I and K as the grouping attributes and the sum of v_a × v_b as the aggregation.

- (c) The *Mapper* emits the key-value pairs (i, k, v_c) for each matrix element of C inputs with key i and k, and v_c from earlier task of the reducer v_a ×

v_b .

- (d) Reducer groups (i, k, v_c) in C using $[C, i, k, \text{sum } (v_c)]$ from aggregated values of v_c from sum (v_c) . Aggregation uses the same memory locations as used by elements v_c . C is just the name, may be represented by 1111.

Matrix Multiplication with One MapReduce Step

MapReduce tasks for Steps 5 to 7 in a single step.

- (e) Map Function: For each element a_{ij} of A , the Mapper emits all the key-value pairs $[(i, k), (A, j, a_{ij})]$ for $k = 1, 2, \dots$, up to the number of columns of B . Similarly, emits all the key-value pairs $[(i, k), (B, j, b_{jk})]$ for $i = 1, 2, \dots$, up to the number of rows of A . for each element b_{jk} of B .
- (f) Reduce Function: Consider the tuples of $A = (A, i, a_{ij})$ for each key j . Consider tuples of $B = (B, k, b_{jk})$ for each key j . Emits the key-value pairs with key equal to (i, k) and value = sum of $(a_{ij} \times b_{jk})$ for all values j .

Memory required in one step MapReduce is large as compared to two steps in cascade. This is due to the need to store intermediate values of v_c and then sum them in the same Reducer step.

Self-Assessment Exercise linked to LO 4.2

1. How does MapReduce program implement counting, filtering and parsing?
2. How does MapReduce collate all items which have the same value?
3. How does MapReduce perform graph analysis in a network of computing nodes to build a spanning tree information at a particular node?
4. How does MapReduce program collate and process items with the same value of the function together in an ETL operation?
5. How does MapReduce program implement <grouping attributes> \S <function-list> (R)?

4.4 | HIVE

LO 4.3

Hive was created by Facebook. Hive is a data warehousing tool and is also a data store on the top of Hadoop. An enterprise uses a data warehouse as large data repositories that are designed to enable the tracking, managing, and analyzing the data. (Section 1.6.1.7) Hive processes structured data and integrates data from multiple heterogeneous sources. Additionally, also manages the constantly growing volumes of data.

Figure 4.9 shows the main features of Hive.

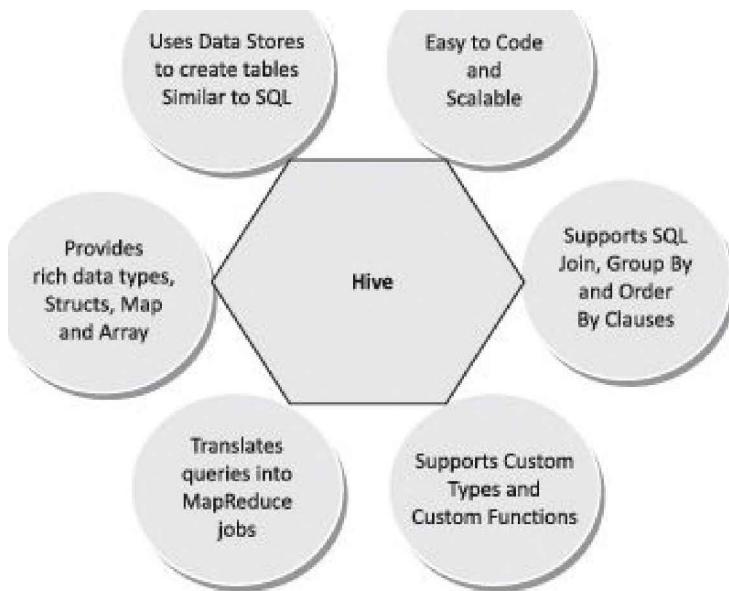


Figure 4.9 Main features of Hive

Hive Characteristics

1. Has the capability to translate queries into MapReduce jobs. This makes Hive scalable, able to handle data warehouse applications, and therefore, suitable for the analysis of static data of an extremely large size, where the

fast response-time is not a criterion.

2. Supports web interfaces as well. Application APIs as well as web-browser clients, can access the Hive DB server.
3. Provides an SQL dialect (Hive Query Language, abbreviated HiveQL or HQL).

Results of HiveQL Query and the data load in the tables which store at the Hadoop cluster at HDFS.

Limitations

Hive is:

1. Not a full database. Main disadvantage is that Hive does not provide update, alter and deletion of records in the database.
2. Not developed for unstructured data.
3. Not designed for real-time queries.
4. Performs the partition always from the last column.

4.4.1 Hive Architecture

Figure 4.10 shows the Hive architecture.

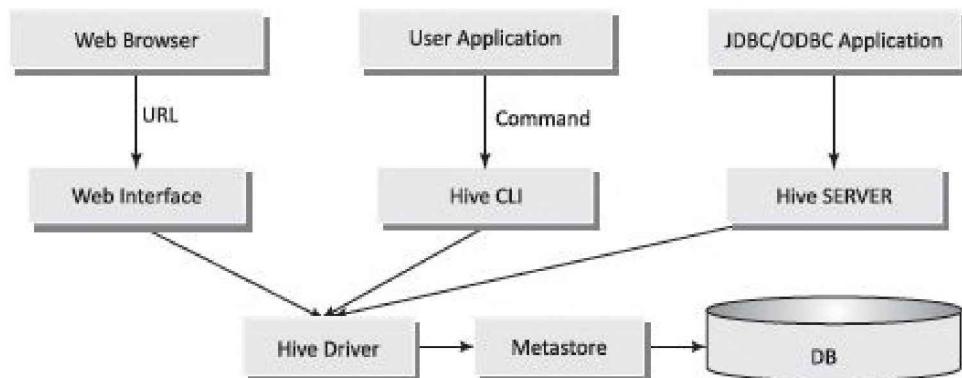


Figure 4.10 Hive architecture

Components of Hive architecture are:

- **Hive Server (Thrift)** – An optional service that allows a remote client to submit requests to Hive and retrieve results. Requests can use a variety of

programming languages. Thrift Server exposes a very simple client API to execute HiveQL statements.

- **Hive CLI (Command Line Interface)** – Popular interface to interact with Hive. Hive runs in local mode that uses local storage when running the CLI on a Hadoop cluster instead of HDFS.
- **Web Interface** – Hive can be accessed using a web browser as well. This requires a HWI Server running on some designated code. The URL `http://hadoop:<port no.> / hwi` command can be used to access Hive through the web.
- **Metastore** – It is the system catalog. All other components of Hive interact with the Metastore. It stores the schema or metadata of tables, databases, columns in a table, their data types and HDFS mapping.
- **Hive Driver** – It manages the life cycle of a HiveQL statement during compilation, optimization and execution.

4.4.2 Hive Installation

Hive can be installed on Windows 10, Ubuntu 16.04 and MySQL. It requires three software packages:

- Java Development kit for Java compiler (Javac) and interpreter
- Hadoop
- Compatible version of Hive with Java– Hive 1.2 onward supports Java 1.7 or newer.

Steps for installation of Hive in a Linux based OS are as follows:

1. Install Javac and Java from Oracle Java download site. Download jdk 7 or a later version from
<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>, and extract the compressed file.

All users can access Java by Make java available to all users. The user has to move it to the location “/usr/local/” using the required commands

2. Set the path by the commands for jdk1.7.0_71, export JAVA_HOME=/usr/local/jdk1.7.0_71, export PATH=\$PATH:\$JAVA_HOME/bin
(Can use alternative install /usr/bin/java usr/local/java/bin/java 2)
3. Install Hadoop <http://apache.claz.org/hadoop/common/hadoop-2.4.1/>
4. Make shared HADOOP, MAPRED, COMMON, HDFS and all related files, configure HADOOP and set property such as replication parameter.
5. Name the yarn.nodemanager.aux-services. Assign value to mapreduce_shuffle. Set namenode and datanode paths.
6. Download <http://apache.petsads.us/hive/hive-0.14.0/>. Use ls command to verify the files \$ tar zxvf apache-hive-0.14.0-bin.tar.gz, \$ ls
OR
Hive archive also extracts by the command apache-hive-0.14.0-bin apache-hive-0.14.0-bin.tar.gz. , \$ cd \$HIVE_HOME/conf, \$ cp hive-env.sh.template hive-env.sh, export HADOOP_HOME=/usr/local/hadoop
7. Use an external database server. Configure metastore for the server.

4.4.3 Comparison with RDBMS (Traditional Database)

Hive is a DB system which defines databases and tables. Hive analyzes structured data in DB. Hive has certain differences with RDBMS. Table 4.3 gives a comparison of Hive database characteristics with RDBMS.

Table 4.3 Comparison of Hive database characteristics with RDBMS

Characteristics	Hive	RDBMS
Record level queries	No Update and Delete	Insert, Update and Delete
Transaction support	No	Yes
Latency	Minutes or more	In fractions of a second
Data size	Petabytes	Terabytes

Data per query	Petabytes	Gigabytes
Query language	HiveQL	SQL
Support JDBC/ODBC	Limited	Full

4.4.4 Hive Data Types and File Formats

Hive defines various primitive, complex, string, date/time, collection data types and file formats for handling and storing different data formats. Table 4.4 gives primitive, string, date/time and complex Hive data types and their descriptions.

Table 4.4 Hive data types and their descriptions

Data Type Name	Description
TINYINT	1 byte signed integer. Postfix letter is Y.
SMALLINT	2 byte signed integer. Postfix letter is S.
INT	4 byte signed integer
BIGINT	8 byte signed integer. Postfix letter is L.
FLOAT	4 byte single-precision floating-point number
DOUBLE	8 byte double-precision floating-point number
BOOLEAN	True or False
TIMESTAMP	UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff”
DATE	YYYY-MM-DD format
VARCHAR	1 to 65355 bytes. Use single quotes (‘ ’) or double quotes (“ ”)
CHAR	255 bytes
DECIMAL	Used for representing immutable arbitrary precision. DECIMAL (precision, scale) format

UNION	Collection of heterogeneous data types. Create union
NULL	Missing values representation

Table 4.5 gives Hive three Collection data types and their descriptions.

Table 4.5 Collection data-types and their descriptions

Name	Description
STRUCT	Similar to 'C' struc, a collection of fields of different data types. An access to field uses dot notation. For example, struct ('a', 'b')
MAP	A collection of key-value pairs. Fields access using [] notation. For example, map ('key1', 'a', 'key2', 'b')
ARRAY	Ordered sequence of same types. Accesses to fields using array index. For example, array ('a', 'b')

Table 4.6 gives the file formats and their descriptions.

Table 4.6 File formats and their descriptions

File Format	Description
Text file	The default file format, and a line represents a record. The delimiting characters separate the lines. Text file examples are CSV, TSV, JSON and XML (Section 3.3.2).
Sequential file	Flat file which stores binary key-value pairs, and supports compression.
RCFile	Record Columnar file (Section 3.3.3.3).
ORCFILE	ORC stands for Optimized Row Columnar which means it can store data in an optimized way than in the other file formats (Section 3.3.3.4).

Record columnar file means one that can be partitioned in rows and then partitioned with columns. Partitioning in this way enables serialization.

4.4.5 Hive Data Model

Table 4.7 below gives three components of Hive data model and their descriptions.

Table 4.7 Components (also called data units) of Hive Data Model

Name	Description
Database	Namespace for tables
Tables	Similar to tables in RDBMS Support filter, projection, join and union operations The table data stores in a directory in HDFS
Partitions	Table can have one or more partition keys that tell how the data stores
Buckets	Data in each partition further divides into buckets based on hash of a column in the table. Stored as a file in the partition directory.

4.4.6 Hive Integration and Workflow Steps

Hive integrates with the MapReduce and HDFS. Figure 4.11 shows the dataflow sequences and workflow steps between Hive and Hadoop.

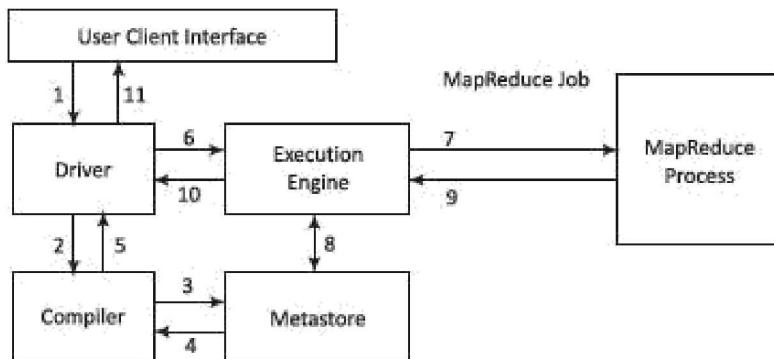


Figure 4.11 Dataflow sequences and workflow steps

Steps 1 to 11 are as follows:

STEP

No.	OPERATION
1	Execute Query: Hive interface (CLI or Web Interface) sends a query to Database Driver to execute the query.
2	Get Plan: Driver sends the query to query compiler that parses the query to check the syntax and query plan or the requirement of the query.
3	Get Metadata: Compiler sends metadata request to Metastore (of any database, such as MySQL).
4	Send Metadata: Metastore sends metadata as a response to compiler.
5	Send Plan: Compiler checks the requirement and resends the plan to driver. The parsing and compiling of the query is complete at this place.
6	Execute Plan: Driver sends the execute plan to execution engine.
7	Execute Job: Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Then, the query executes the job.
8	Metadata Operations: Meanwhile the execution engine can execute the metadata operations with Metastore.
9	Fetch Result: Execution engine receives the results from Data nodes.
10	Send Results: Execution engine sends the result to Driver.
11	Send Results: Driver sends the results to Hive Interfaces.

4.4.7 Hive Built-in Functions

Hive supports a number of built-in functions. Table 4.8 gives the return types, syntax and descriptions of the examples of these functions.

Table 4.8 Return types, syntax, and descriptions of the functions

Return Type	Syntax	Description

BIGINT	round(double a)	Returns the rounded BIGINT (8 Byte integer) value of the 8 Byte double-precision floating point number a
BIGINT	floor(double a)	Returns the maximum BIGINT value that is equal to or less than the double.
BIGINT	ceil(double a)	Returns the minimum BIGINT value that is equal to or greater than the double.
double	rand(), rand(int seed)	Returns a random number (double) that distributes uniformly from 0 to 1 and that changes in each row. Integer seed ensured that random number sequence is deterministic.
string	concat(string str1, string str2, ...)	Returns the string resulting from concatenating str1 with str2,
string	substr(string str, int start)	Returns the substring of str starting from a start position till the end of string str.
string	substr(string str, int start, int length)	Returns the substring of str starting from the start position with the given length.
string	upper(string str), ucase (string str)	Returns the string resulting from converting all characters of str to upper case.
string	lower(string str), lcase(string str)	Returns the string resulting from converting all characters of str to lower case.
string	trim(string str)	Returns the string resulting from trimming spaces from both ends. trim ('12A34 56') returns '12A3456'
string	ltrim(string str); rtrim(string str)	Returns the string resulting from trimming spaces (only one end, left or right hand side or right-handside spaces trimmed). ltrim('12A34 56') returns '12A3456' and rtrim(' 12A34 56 ') returns '12A3456'.
string	rtrim(string str)	Returns the string resulting from trimming spaces from the end (right hand side) of str.

int	year(string date)	Returns the year part of a date or a timestamp string.
int	month(string date)	Returns the month part of a date or a timestamp string.
int	day(string date)	Returns the day part of a date or a timestamp string.

Following are the examples of the returned output:

SELECT floor(10.5) from marks; Output = 10.0

SELECT ceil(10.5) from marks; Output = 11.0

Self-Assessment Exercise linked to LO 4.3

1. How does Hive install? What are the features of Hive? What are the components of the Hive architecture?
2. Give reasons for Hive provided with distinct integer types: TINYINT, SMALLINT, INT and BIGINT.
3. How does Hive use text, sequential, RC and ORC files?
4. How does the Hive use Collection data types: STRUCT, MAP and ARRAY?
5. How does Hive integrate with MapReduce and HDFS?
6. Give one example each of usages of round(), floor(), ceil(), rand() and upper() built-in functions in Hive.

4.5 | HIVEQL

Hive Query Language (abbreviated HiveQL) is for querying the large datasets which reside in the HDFS environment. HiveQL script commands enable data definition, data manipulation and query processing. HiveQL supports a large base of SQL users who are acquainted with SQL to extract information from data

LO 4.4

HiveQL for querying, sorting, aggregating, querying scripts, and MapReduce scripts for Joins and sub-queries

warehouses.

HiveQL Process Engine	HiveQL is similar to SQL for querying on schema information at the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The bridge between HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results same as MapReduce results. It uses the flavor of MapReduce.

The subsections ahead give the details of data definition, data manipulation and querying data examples.

4.5.1 HiveQL Data Definition Language (DDL)

HiveQL database commands for data definition for DBs and Tables are CREATE DATABASE, SHOW DATABASE (list of all DBs), CREATE SCHEMA, CREATE TABLE. Following are HiveQL commands which create a table:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [<database name>.]  
<table name>  
[(<column name> <data type> [COMMENT <column comment>], ...)]  
[COMMENT <table comment>]  
[ROW FORMAT <row format>]  
[STORED AS <file format>]
```

Table 4.9 gives the row formats in a Hive table.

Table 4.9 Hive Table Row Formats

DELIMITED	Specifies a delimiter at the table level for structured fields. This is default. Syntax: FIELDS TERMINATED BY, LINES TERMINATED BY
SERDE	Stands for Serializer/Deserializer. SYNTAX: SERDE 'serde.class.name'

HiveQL database commands for data definition for the DBs and Tables are CREATE DATABASE, SHOW DATABASE (list of all DBs), CREATE SCHEMA, CREATE TABLE.

The following example uses HiveQL commands to create a database

toys_companyDB.

—

EXAMPLE 4.7

How do you create a database named toys_companyDB and table named toys_tbl?

SOLUTION

```
$HIVE_HOME/binhive - service cli  
hive>set hive.cli.print.current.db=true;  
hive> CREATE DATABASE toys_companyDB  
hive>USE toys_companyDB  
hive (toys_companyDB)> CREATE TABLE toys_tbl (  
>puzzle_code STRING,  
>pieces SMALLINT  
>cost FLOAT);  
hive (toys_companyDB)> quit;  
&ls/home/binadmin/Hive/warehouse/toys_companyDB.db
```

The following example uses the command CREATE TABLE to create a table toy_products.

—

EXAMPLE 4.8

How do you create a table toy_products with the following fields?

Field	Data type
ProductCategory	string
ProductId	int
ProductName	string
ProductPrice	float

SOLUTION

```
CREATE TABLE IF NOT EXISTS toy_products (ProductCategory String,
```

```
ProductId int, ProductName String, ProductPrice float)
COMMENT 'Toy details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

The option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

Consider the following command:

A command is

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>;
```

IF NOT EXISTS is an optional clause. The clause notifies the user that a database with the same name already exists. SCHEMA can be also created in place of DATABASE using this command

A command is written to get the list of all existing databases.

```
SHOW DATABASES;
```

A command is written to delete an existing database.

```
DROP (DATABASE | SCHEMA) [IF EXISTS] <database name>
[RESTRICT | CASCADE];
```

The following example gives the sample usages of the commands.

EXAMPLE 4.9

Give examples of usages of database commands for CREATE, SHOW and DROP.

SOLUTION

```
CREATE DATABASE IF NOT EXISTS toys_companyDB;
SHOW DATABASES;
default toys_companyDB
```

*Default database is test.

Delete database using the command:

Drop Database toys_companyDB.

4.5.2 HiveQL Data Manipulation Language (DML)

HiveQL commands for data manipulation are USE <database name>, DROP DATABASE, DROP SCHEMA, ALTER TABLE, DROP TABLE, and LOAD DATA.

The following is a command for inserting (loading) data into the Hive DBs.

```
LOAD DATA [LOCAL] INPATH '<file path>' [OVERWRITE] INTO  
TABLE <table name> [PARTITION (partcol1=val1,  
partcol2=val2 ...)]
```

LOCAL is an identifier to specify the local path. It is optional. OVERWRITE is optional to overwrite the data in the table. PARTITION is optional. val1 is value assigned to partition column 1 (partcol1) and val2 is value assigned to partition column 2 (partcol2).

Command	Functionality	Script Example
LOAD DATA	Insert data in a table	LOAD DATA LOCAL INPATH '/home/user/jigsaw_puzzle_info.txt' OVERWRITE INTO TABLE toy_tbl;

The following is an example for usages of data manipulation commands, INSERT, ALTER, and DROP.

EXAMPLE 4.10

Consider an example of a toy company selling Jigsaws. Consider a text file named jigsaw_puzzle_info.txt in /home/user directory. The file is text file with four fields: Toy-category, toy-id, toy-name, and Price in US\$ as follows:

```
puzzle_Garden 10725 Fantasy 1.35  
puzzle_Jungle 31047 Animals 2.85  
puzzle_School 81049 Nursery 4.45
```

How will you use (i) LOAD (insert), (ii) ALTER and (iii) DROP commands?

SOLUTION

- (i) Insert the data of this file into a table using the following commands:

```
LOAD DATA LOCAL INPATH '/home/user/jigsaw_puzzle_info.txt'  
OVERWRITE INTO TABLE jigsaw_puzzle;
```

- (ii) Alter the table using the following commands:

```
ALTER TABLE <name> RENAME TO <new name>
```

```
ALTER TABLE <name> ADD COLUMNS (<col spec> [, <col spec> ...])
```

```
ALTER TABLE <name> DROP [COLUMN] <column name>
```

```
ALTER TABLE <name> CHANGE <column name> <new name> <new type>
```

```
ALTER TABLE <name> REPLACE COLUMNS (<col spec> [, <col spec> ...])
```

The following query renames the table from jigsaw_puzzle to toy_tbl:

```
ALTER TABLE jigsaw_puzzle RENAME TO toy_tbl;
```

The following query renames the column name ProductCategory to ProductCat:

```
ALTER TABLE toy_tbl CHANGE ProductCategory ProductCat String;
```

The following query renames data type of ProductPrice from float to double:

```
ALTER TABLE toy_tbl CHANGE ProductPrice ProductPrice Double;
```

The following query adds a column named ProductDesc to the toy_tbl table:

```
ALTER TABLE toy_tbl ADD COLUMNS (ProductDesc String COMMENT  
'Product Description');
```

The following query deletes all the columns from the toy_tbl table and replaces it with ProdCat and ProdName columns:

```
ALTER TABLE toy_tbl REPLACE COLUMNS (ProductCategory INT  
ProdCat Int, ProductName STRING ProdName String);
```

- (iii) The following query deletes a column named ProductDesc from the toy_tbl table:

```
ALTER TABLE toy_tbl DROP COLUMN ProductDesc;
```

A table DROP using the following command: `DROP TABLE [IF EXISTS] table_name;`

The following query drops a table named `jigsaw_puzzle`:

```
DROP TABLE IF EXISTS jigsaw_puzzle;
```

4.5.3 HiveQL For Querying the Data

Partitioning and storing are the requirements. A data warehouse should have a large number of partitions where the tables, files and databases store. Querying then requires sorting, aggregating and joining functions.

Querying the data is to `SELECT` a specific entity *satisfying* a condition, *having* presence of an entity or selecting specific entity using `GroupBy`.

```
SELECT [ALL | DISTINCT] <select expression>, <select expression>, ...
FROM <table name>
[WHERE <where condition>]
[GROUP BY <column List>]
[HAVING <having condition>]
[CLUSTER BY <column List>| [DISTRIBUTE BY <column List>] [SORT BY <column List>]]
[LIMIT number];
```

4.5.3.1 Partitioning

Hive organizes tables into partitions. Table partitioning refers to dividing the table data into some parts based on the values of particular set of columns. Partition makes querying easy and fast. This is because `SELECT` is then from the smaller number of column fields. Section 3.3.3.3 described RC columnar format and serialized records. The following example explains the concept of partitioning, columnar and file records formats.

EXAMPLE 4.11

Consider a table T with eight-column and four-row table. Partition the table, convert in RC columnar format and serialize.

SOLUTION

Firstly, divide the table in four parts, t_{r1} , t_{r2} , t_{r3} and t_{r4} horizontally row-wise. Each sub-table has one row and eight columns. Now, convert each sub-table t_{r1} , t_{r2} , t_{r3} and t_{r4} into columnar format, or RC File records [Recall Example 3.7 on how RC file saves each row-group data in a format using SERDE (serializer/des-serializer)].

Each sub-table has eight rows and one column. Each column can serially send data one value at an instance. A column has eight key-value pairs with the same key for all the eight.

Table Partitioning

Create a table with Partition using command:

```
CREATE [EXTERNAL] TABLE <table name> (<column name 1>
<data type 1>, ....)
PARTITIONED BY (<column name n> <data type n> [COMMENT
<column comment>], ...);
```

Rename a Partition in the existing Table using the following command:

```
ALTER TABLE <table name> PARTITION partition_spec
RENAME TO PARTITION partition_spec;
```

Add a Partition in the existing Table using the following command:

```
ALTER TABLE <table name> ADD [IF NOT EXISTS] PARTITION
partition_spec
[LOCATION 'location1'] partition_spec [LOCATION
'location2'] ...
partition_spec: (p_column = p_col_value, p_column =
p_col_value, ...)
```

Drop a Partition in the existing Table using the following command:

```
ALTER TABLE <table name> DROP [IF EXISTS] PARTITION
partition_spec, PARTITION partition_spec;
```

The following example explains concept of add, rename and drop a partition.

EXAMPLE 4.12

How will you add, rename and drop a partition to a table, toys_tbl?

SOLUTION

- (i) Add a partition to the existing toy table using the command:

```
ALTER TABLE toy_tbl ADD PARTITION  
(category='Toy_Airplane')  
'/Toy_Airplane/partAirplane';
```

- (ii) The following query renames a partition:

```
ALTER TABLE toy_tbl PARTITION  
(category='Toy_Airplane') RENAME TO PARTITION  
(name='Fighter');
```

- (iii) Drop a Partition in the existing Table using the command:

```
ALTER TABLE toy_tbl DROP [IF EXISTS] PARTITION  
(category='Toy_Airplane');
```

The following example explains how querying is facilitated by using partitioning of a table. A query processes faster when using partition. Selection of a product of a specific category from a table during query processing takes lesser time when the table has a partition based on a category.

EXAMPLE 4.13

Assume that following file contains toys_tbl.

```
/table/toy_tbl/file1  
Category, id, name, price  
Toy_Airplane, 10725, Lost Temple, 1.25  
Toy_Airplane, 31047, Propeller Plane, 2.10  
Toy_Airplane, 31049, Twin Spin Helicopter, 3.45
```

```
Toy_Train, 31054, Blue Express, 4.25
```

```
Toy_Train, 10254, Winter Holiday Toy_Train, 2.75
```

A table *toy_tbl* contains many values for categories of toys. Query is required to identify all toy_airplane fields. Give reasons why partitioning reduces query processing time.

SOLUTION

Here, a table named *toy_tbl* contains several toy details (category, id, name and price). Suppose it is required to identify all the airplanes. A query searches the whole table for the required information. However, if a partition is created on the *toy_tbl*, based on category and stores it in a separate file, then it will reduce the query processing time.

Let the data partitions into two files, file 2 and file 3, using category.

```
/table/toys/toy_airplane/file2
toy_airplane, 10725, Lost Temple, TP, 1.25
toy_airplane, 31047, Propeller Plane, 2.10
toy_airplane, 31049, Lost Temple, 3.45
/table/toys/toy_train/file3
Toy_Train, 31054, Blue Express, 4.25
Toy_Train, 10254, Winter Holiday Toy_Train, 2.75
```

Advantages of Partition

1. Distributes execution load horizontally.
2. Query response time becomes faster when processing a small part of the data instead of searching the entire dataset.

Limitations of Partition

1. Creating a large number of partitions in a table leads to a large number of files and directories in HDFS, which is an overhead to NameNode, since it must keep all metadata for the file system in memory only.
2. Partitions may optimize some queries based on Where clauses, but they may be less responsive for other important queries on grouping clauses.

3. A large number of partitions will lead to a large number of tasks (which will run in separate JVM) in each MapReduce job, thus creating a lot of overhead in maintaining JVM start up and tear down (A separate task will be used for each file). The overhead of JVM start up and tear down can exceed the actual processing time in the worst case.

4.5.3.2 Bucketing

A partition itself may have a large number of columns when tables are very large. Tables or partitions can be sub-divided into buckets. Division is based on the hash of a column in the table.

Consider bucketed column $C_{\text{bucket_}i}$. First, define a hash_function $H()$ according to type of the bucketed column. Let the total number of buckets = N_{buckets} . Let $C_{\text{bucket_}i}$ denote i^{th} bucketed column. The hash value h_i = hashing function $H(C_{\text{bucket_}i}) \bmod (N_{\text{buckets}})$.

Buckets provide an extra structure to the data that can lead to more efficient query processing when compared to undivided tables or partition. Buckets store as a file in the partition directory. Records with the same bucketed column will always be stored in the same bucket. Records kept in each bucket provide sorting ease and enable Map task Joins. A Bucket can also be used as a sample dataset.

CLUSTERED BY clause divides a table into buckets. A coding example on Buckets is given below:

EXAMPLE 4.14

A table `toy_tbl` contains many values for categories of toys. Assume the number of buckets to be

`created = 5`. Assume a table for `Toy_Airplane` of product code 10725.

1. How will the bucketing enforce?
2. How will the bucketed table partition `toy_airplane_10725` create five buckets?
3. How will the bucket column load into `toy_tbl`?
4. How will the bucket data display?

SOLUTION

#Enforce bucketing

```
set hive.enforce.bucketing=true;
```

#Create bucketed Table for toy_airplane of product code 10725 and create cluster of 5 buckets

```
CREATE TABLE IF NOT EXISTS  
toy_airplane_10725(ProductCategory STRING,  
ProductId INT, ProductName STRING, PrdocutMfgDate  
YYYY-MM-DD, ProductPrice_US$ FLOAT) CLUSTERED BY  
(Price) into 5 buckets;
```

Load data to bucketed table.

```
FROM toy_airplane_10725 INSERT OVERWRITE TABLE  
toy_tbl SELECT ProductCategory, ProductId,  
ProductName, PrdocutMfgDate, ProductPrice;
```

- To display the contents for Price_US\$ selected for the ProductId from the second bucket.

```
SELECT DISTINCT ProductId FROM toy_tbl_buckets  
TABLE FOR 10725(BUCKET 2 OUT OF 5 ON Price_US$);
```

4.5.3.3 Views

A program uses functions or objects. Constructing an object instance enables layered design and encapsulating the complexity due to methods and fields. Similarly, Views provide ease of programming. Complex queries simplify using reusable Views. A HiveQL View is a logical construct.

A View provisions the following:

- Saves the query and reduces the query complexity
- Use a View like a table but a View does not store data like a table
- Hive query statement when uses references to a view, the Hive executes the View and then the planner combines the information in View definition with the remaining actions on the query (Hive has a query planner, which plans how a query breaks into sub-queries for obtaining

the right answer.)

- Hides the complexity by dividing the query into smaller, more manageable pieces.

4.5.3.4 Sub-Queries (Using Views)

Consider the following query with a nested sub-query.

EXAMPLE 4.15

A table *toy_tbl* contains many values for categories of toys. Assume a table for Toy_Airplane of product code 10725. Consider a nested query:

```
FROM (
  SELECT * toy_tbl_Join people JOIN Toy_Airplane
    ON (Toy_Airplane.ProductId= productId.id) WHERE productId=10725
      ) toys_ catalog SELECT prdocutMfgDate WHERE prdocutMfgDate = '2017-10-23';
```

Create a View for using that in a nested query.

SOLUTION

```
# create a view named toy_tbl_MiniJoin
CREATE VIEW toy_tbl_MiniJoin AS
SELECT * toy_tbl_Join people JOIN Toy_Airplane
  ON (Toy_Airplane.ProductId= productId.id) WHERE productId=10725
    ) toys_ catalog SELECT prdocutMfgDate WHERE prdocutMfgDate = '2017-10-23';
```

4.5.4 Aggregation

Hive supports the following built-in aggregation functions. The usage of these functions is same as the SQL aggregate functions. Table 4.10 lists the functions, their syntax and descriptions.

Table 4.10 Aggregate functions, their return type, syntax and descriptions

Return Type	Syntax	Description
BIGINT	count(*), count(expr)	Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	Returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg (col), avg (DISTINCT col)	Returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min (col)	Returns the minimum value of the column in the group.
DOUBLE	max(col)	Returns the maximum value of the column in the group.

Usage examples are:

Example: `SELECT ProductCategory, count (*) FROM toy_tbl GROUP BY ProductCategory;`

Example: `SELECT ProductCategory, sum(ProductPrice) FROM toy_tbl GROUP BY ProductCategory;`

4.5.5 Join

A JOIN clause combines columns of two or more tables, based on a relation between them. HiveQL Join is more or less similar to SQL JOINS. Following uses of two tables show the Join operations.

Table 4.11 gives an example of a table named `toy_tbl` of Product categories, ProductId and Product name.

Table 4.11 Table of Product categories, Product Id and Product name

ProductCategory	ProductId	ProductName
Toy_Airplane	10725	Lost temple
Toy_Airplane	31047	Propeller plane
Toy_Airplane	31049	Twin spin helicopter

Toy_Train	31054	Blue express
Toy_Train	10254	Winter holiday Toy_Train

Table 4.12 gives an example of a table named *price* of ID or Product ID and ProductCost.

Table 4.12 Table of ID and Product Cost

Id	ProductPrice
10725	100.0
31047	200.0
31049	300.0
31054	450.0
10254	200.0

Different types of joins are follows:

JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

JOIN Join clause combines and retrieves the records from multiple tables. Join is the same as OUTER JOIN in SQL. A JOIN condition uses primary keys and foreign keys of the tables.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice
FROM toy_tbl t JOIN price p
ON (t.ProductId = p.Id);
```

LEFT OUTER JOIN A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice
FROM toy_tbl t LEFT OUTER JOIN price p
ON (t.ProductId = p.Id);
```

RIGHT OUTER JOIN A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice  
FROM toy_tbl t RIGHT OUTER JOIN price p  
ON (t.ProductId = p.Id);
```

FULL OUTER JOIN HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfill the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice  
FROM toy_tbl t FULL OUTER JOIN price p  
ON (t.ProductId = p.Id);
```

4.5.6 Group by Clause

GROUP BY, HAVING, ORDER BY, DISTRIBUTIVE BY, CLUSTER BY are HiveQL clauses. An example of using the clauses is given below:

EXAMPLE 4.16

How do SELECT statement uses GROUP BY, HAVING, DISTRIBUTIVE BY, CLUSTER BY? How does clause GROUP BY used in queries on toy_tbl?

SOLUTION

- (i) Use of SELECT statement with WHERE clause is as follows:

```
SELECT [ALL | DISTINCT] <select expression>,  
<select expression>, ...  
FROM <table name>  
[WHERE <where condition>]  
[GROUP BY <column List>]  
[HAVING <having condition>]  
[CLUSTER BY <column List>| [DISTRIBUTE BY <column  
List>] [SORT BY <column List>]]
```

```
[LIMIT number];
```

- (ii) Use of the clauses in queries to toy_tbl is as follows:

```
SELECT * FROM toy WHERE ProductPrice > 1.5;
```

```
SELECT ProductCategory, count (*) FROM toy_tbl  
GROUP BY ProductCategory;
```

```
SELECT ProductCategory, sum(ProductPrice) FROM  
toy_tbl GROUP BY ProductCategory;
```

Self-Assessment Exercise linked to LO 4.4

1. What are the results after execution of the following command?

```
CREATE TABLE IF NOT EXISTS toy_products  
(ProductCategory String, ProductId int, ProductName  
String, ProductPrice float)  
COMMENT 'Toy details'  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;
```

2. What do the following statements mean?

```
ALTER TABLE <puzzle_info> RENAME TO  
<jigsaw_puzzle_info>  
ALTER TABLE <jigsaw_puzzle_info> ADD COLUMNS  
(<puzzle_code_name>[,<pieces> [,<puzzle_cost> [, ...]])  
ALTER TABLE <jigsaw_puzzle_info> DROP [COLUMN]  
<puzzle_cost_US$>
```

3. How do you create partitions and buckets in a Hive database?

4. Consider sales table for all five car models at a large number of showrooms. How are the sales figures of a specific model queried?

5. Explain the meaning of the following statements:

```
SELECT [ALL | DISTINCT] <select expression>, <select
```

```

expression>, ...
FROM <table name>
[WHERE <where condition>]
[GROUP BY <column List>]
[HAVING <having condition>]
[CLUSTER BY <column List>| [DISTRIBUTE BY <column List>] [SORT BY <column List>]]
[LIMIT number];

```

4.6 | PIG

Apache developed Pig, which:

LO 4.5

Pig, architecture, Grunt shell commands, data model, Pig Latin, developing scripts, and extensibility using UDFs

- Is an abstraction over MapReduce
- Is an execution framework for parallel processing
- Reduces the complexities of writing a MapReduce program
- Is a high-level dataflow language. Dataflow language means that a Pig operation node takes the inputs and generates the output for the next node
- Is mostly used in HDFS environment
- Performs data manipulation operations at files at data nodes in Hadoop.

1. Applications of Apache Pig

Applications of Pig are:

- Analyzing large datasets
- Executing tasks involving adhoc processing
- Processing large data sources such as web logs and streaming online data
- Data processing for search platforms. Pig processes different types of data

- Processing time sensitive data loads; data extracts and analyzes quickly. For example, analysis of data from twitter to find patterns for user behavior and recommendations.

2. Features

- (i) Apache PIG helps programmers write complex data transformations using scripts (without using Java). Pig Latin language is very similar to SQL and possess a rich set of built-in operators, such as group, join, filter, limit, order by, parallel, sort and split. It provides an interactive shell known as Grunt to write Pig Latin scripts. Programmers write scripts using Pig Latin to analyze data. The scripts are internally converted to Map and Reduce tasks with the help of the component known as Execution Engine, that accepts the Pig Latin scripts as input and converts these scripts into MapReduce jobs. Writing MapReduce tasks was the only way to process the data stored in HDFS before the Pig.
- (ii) Creates user defined functions (UDFs) to write custom functions which are not available in Pig. A UDF can be in other programming languages, such as Java, Python, Ruby, Jython, JRuby. They easily embed into Pig scripts written in Pig Latin. UDFs provide extensibility to the Pig.
- (iii) Process any kind of data, structured, semi-structured or unstructured data, coming from various sources.
- (iv) Reduces the length of codes using multi-query approach. Pig code of 10 lines is equal to MapReduce code of 200 lines. Thus, the processing is very fast.
- (v) Handles inconsistent schema in case of unstructured data as well.
- (vi) Extracts the data, performs operations on that data and dumps the data in the required format in HDFS. The operation is called ETL (Extract Transform Load).
- (vii) Performs automatic optimization of tasks before execution.
- (viii) Programmers and developers can concentrate on the whole operation without a need to create mapper and reducer tasks separately.

- (ix) Reads the input data files from HDFS or the data files from other sources such as local file system, stores the intermediate data and writes back the output in HDFS.
- (x) Pig characteristics are data reading, processing, programming the UDFs in multiple languages and programming multiple queries by fewer codes. This causes fast processing.
- (xi) Pig derives guidance from four philosophies, live anywhere, take anything, domestic and run as if flying. This justifies the name Pig, as the animal pig also has these characteristics. Table 4.13 gives differences between Pig and MapReduce.

Table 4.13 Differences between Pig and MapReduce

Pig	MapReduce
A dataflow language	A data processing paradigm
High level language and flexible	Low level language and rigid
Performing Join, filter, sorting or ordering operations are quite simple	Relatively difficult to perform Join, filter, sorting or ordering operations between datasets
Programmer with a basic knowledge of SQL can work conveniently	Complex Java implementations require exposure to Java language
Uses multi-query approach, thereby reducing the length of the codes significantly	Require almost 20 times more the number of lines to perform the same task
No need for compilation for execution; operators convert internally into MapReduce jobs	Long compilation process for Jobs
Provides nested data types like tuples, bags and maps	No such data types

Table 4.14 gives differences between Pig and SQL.

Table 4.14 Differences between Pig and SQL

Pig	SQL
Pig Latin is a procedural language	A declarative language

Schema is optional, stores data without assigning a schema	Schema is mandatory
Nested relational data model	Flat relational data model
Provides limited opportunity for Query optimization	More opportunity for query optimization

Pig and Hive codes, both create MapReduce jobs when execute. Hive in some cases, operates on HDFS in a similar way Apache Pig does. Table 4.15 gives a few significant points that set Pig apart from Hive.

Table 4.15 Differences between Pig and Hive

Pig	Hive
Originally created at Yahoo	Originally created at Facebook
Exploits Pig Latin language	Exploits HiveQL
Pig Latin is a dataflow language	HiveQL is a query processing language
Pig Latin is a procedural language and it fits in pipeline paradigm	HiveQL is a declarative language
Handles structured, unstructured and semi-structured data	Mostly used for structured data

3. Pig Architecture

Firstly, Pig Latin scripts submit to the Apache Pig Execution Engine. Figure 4.12 shows Pig architecture for scripts dataflow and processing in the HDFS environment.

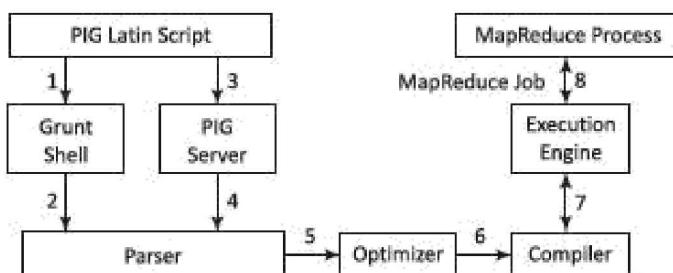


Figure 4.12 Pig architecture for scripts dataflow and processing

The three ways to execute scripts are:

1. **Grunt Shell:** An interactive shell of Pig that executes the scripts.
2. **Script File:** Pig commands written in a script file that execute at Pig Server.
3. **Embedded Script:** Create UDFs for the functions unavailable in Pig built-in operators. UDF can be in other programming languages. The UDFs can embed in Pig Latin Script file.

Parser A parser handles Pig scripts after passing through Grunt or Pig Server. The Parser performs type checking and checks the script syntax. The output is a Directed Acyclic Graph (DAG). Acyclic means only one set of inputs are simultaneously at a node, and only one set of output generates after node operations. DAG represents the Pig Latin statements and logical operators. Nodes represent the logical operators. Edges between sequentially traversed nodes represent the dataflows.

Optimizer The DAG is submitted to the logical optimizer. The optimization activities, such as split, merge, transform and reorder operators execute in this phase. The optimization is an automatic feature. The optimizer reduces the amount of data in the pipeline at any instant of time, while processing the extracted data. It executes certain functions for carrying out this task, as explained as follows:

PushUpFilter: If there are multiple conditions in the filter and the filter can be split, Pig splits the conditions and pushes up each condition separately. Selecting these conditions at an early stage helps in reducing the number of records remaining in the pipeline.

PushDownForEachFlatten: Applying flatten, which produces a cross product between a complex type such as a tuple, bag or other fields in the record, as late as possible in the plan. This keeps the number of records low in the pipeline.

ColumnPruner: Omits never used columns or the ones no longer needed, reducing the size of the record. This can be applied after each operator, so that the fields can be pruned as aggressively as possible.

MapKeyPruner: Omits never used map keys, reducing the size of the record.

LimitOptimizer: If the limit operator is immediately applied after *load* or *sort* operator, Pig converts the load or sort into a limit-sensitive implementation, which does not require processing the whole dataset. Applying the limit earlier reduces the number of records.

Compiler The compiler compiles after the optimization process. The optimized codes are a series of MapReduce jobs.

Execution Engine Finally, the MapReduce jobs submit for execution to the engine. The MapReduce jobs execute and it outputs the final result.

4.6.1 Apache Pig – Grunt Shell

Main use of Grunt shell is for writing Pig Latin scripts. Any shell command invokes using sh and ls. Syntax of sh command is:

```
grunt> sh shell command parameters
```

Syntax of ls command:

```
grunt> sh ls
```

Grunt shell includes a set of utility commands. Included utility commands are clear, help, history, quit and set. The shell includes commands such as exec, kill and run to control the Pig from the Grunt shell.

4.6.2 Installing Pig

Following are the steps for installing Pig:

1. Download the latest version from - <https://pig.apache.org/>
2. Download the tar files and create a Pig directory

```
$ cd Downloads/  
$ tar zxvf pig-0.15.0-src.tar.gz  
$ tar zxvf pig-0.15.0.tar.gz  
$ mv pig-0.15.0-src.tar.gz/* /home/Hadoop/Pig/
```

3. Configure the Pig

```
export PIG_HOME = /home/Hadoop/Pig
```

```

export PATH = $PATH:/home/Hadoop/pig/bin
export PIG_CLASSPATH = $HADOOP_HOME/conf

```

4.6.3 Pig Latin Data Model

Pig Latin supports primitive data types which are atomic or scalar data types. Atomic data types are int, float, long, double, char [], byte []. The language also defines complex data types. Complex data types are tuple, bag and map. Table 4.16 gives data types and examples.

Table 4.16 Data types and examples

Data type	Description	Example
bag	Collection of tuples	{(1,1), (2,4)}
tuple	Ordered set of fields	(1,1)
map (data map)	Set of key-value pairs	[Number#1]
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L or 10l
float	32-bit floating point	22.7F or 22.7f
double	64-bit floating point	3.4 or 3.4e2 or 3.4E2
chararray	Char [], Character array	data analytics
bytearray	BLOB (Byte array)	ff00

A simple atomic value is known as a field. For example, ‘Oreo’ or ‘10’ are fields. Atomic means non-divisible. NULL denotes an unknown or non-existent value in Pig Latin.

Tuple Tuple is a record of an ordered set of fields. A tuple is similar to a row in a table of RDBMS. The elements inside a tuple do not necessarily need to have a schema associated to it. A tuple represents by ‘()’ symbol. For example, (1, Oreo, 10, Cadbury)

Indices of the fields access the fields in each tuple. Tuples are ordered, like \$1 from above tuple will return a value ‘Oreo’.

Figure 4.13 shows Pig data model with fields, Tuple and Bag.

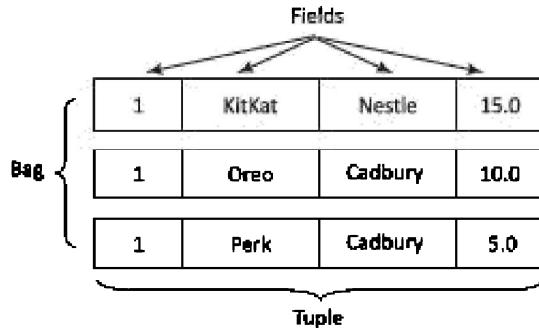


Figure 4.13 Pig Data Model with fields, Tuple and Bag

Bag A bag is an unordered set of tuples. A bag can contain duplicate tuples as it is not mandatory that they need to be unique. Each tuple can have any number of fields (flexible schema). A bag can also have tuples with different data types.

{ } symbol represents a bag. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contains the same number of fields or that the fields in the same position (column) have the same type. For example, {(Oreo, 10), (KitKat, 15, Cadbury)}

There are two types of bag: outer bag or relations and inner bag. Outer bag or relation is a bag of tuples. Here relations are similar as relations in relational databases. To understand it better let us take an example: {(Oreo, Cadbury), (KitKat, Nestle), (Perk, Cadbury)}. This bag explains the relation between the Chocolate brand and their brand company.

A bag can be a field in a relation; in that context, it is known as an inner bag. Thus, an inner bag contains a bag inside a tuple. Figure 4.14 shows a relation and keys and their values: (Cadbury, {(Oreo, 10), (Perk, 5)}) (Nestle {(Kitkat, 15)})

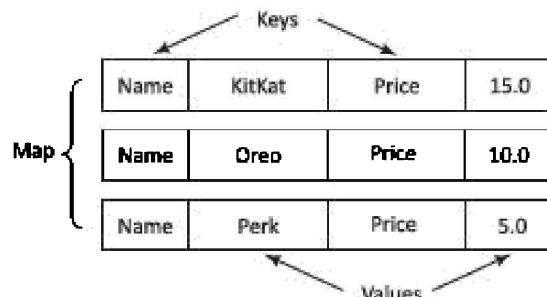


Figure 4.14 Relation and corresponding keys and their values (key-value pairs)

Relation A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

Map A map (or data map) is a set of key-value pairs. The key needs to be of type chararray and should be unique (similar to a column name). Map can be indexed and value associated with it can be accessed from the keys. The value might be of any type. [] symbol represents Map. The key and value separate by '#' symbol. For example, [type#Oreo, price#10]

4.6.4 Pig Latin and Developing Pig Latin Scripts

Pig Latin enables developing the scripts for data analysis. A number of operators in Pig Latin help to develop their own functions for reading, writing and processing data. Pig Latin programs execute in the Pig run-time environment.

Pig Latin

Statements in Pig Latin:

1. Basic constructs to process the data.
2. Include schemas and expressions.
3. End with a semicolon.
4. LOAD statement reads the data from file system, DUMP displays the result and STORE stores the result.
5. Single line comments begin with -- and multiline begin with /* and end with */
6. Keywords (for example, LOAD, STORE, DUMP) are not case-sensitive.
7. Function names, relations and paths are case-sensitive.

Figure 4.15 shows the order of processing Pig statements—Load, dump and store.

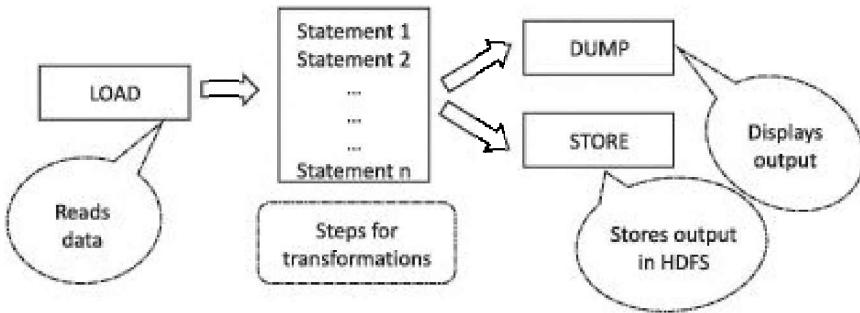


Figure 4.15 Order of processing Pig statements—Load, dump, and store

Operators In Pig Latin

Arithmetic Operators		+	-	*	/	%
Used for		Addition	Subtraction	Multiplication	Division	Remainder
Comparison Operators	==	!=	<	>	≤	≥
Used for	Equality	Not equal	Less than	Greater than	Less than and equal to	Greater than and equal to
Boolean Operators	AND		OR	NOT		
Used for	Logical AND	Logical OR		Logical NOT		

4.6.4.1 Apache Pig Execution

Pig Execution Modes Local Mode: All the data files install and run from a local host using the local file system. Local mode is mostly used for testing purpose.

COMMAND: `pig -x local`

MapReduce Mode: All the data files load or process that exists in the HDFS. A MapReduce job invokes in the back-end to perform a particular operation on the data that exists in the HDFS when a Pig Latin statement executes to process the data.

COMMAND: `pig -x mapreduce or pig`

Pig Latin Script Execution Modes

- Interactive Mode - Using the Grunt shell.
- Batch Mode - Writing the Pig Latin script in a single file with .pig extension.

- Embedded Mode - Defining UDFs in programming languages such as Java, and using them in the script.

4.6.4.2 Commands

- To get the list of pig commands: `pig -help`;
- To get the version of pig: `pig -version`.
- To start the Grunt shell, write the command: `pig`

LOAD Command The first step to a dataflow is to specify the input. Load statement in Pig Latin loads the data from PigStorage.

To load data from HBase: `book = load 'MyBook' using HBaseStorage();`

For reading CSV file, PigStorage takes an argument which indicates which character to use as a separator. For example, `book = LOAD 'PigDemo/Data/Input/myBook.csv' USING PigStorage (,);`

For reading text data line by line: `book = LOAD 'PigDemo/Data/Input/myBook.txt' USING PigStorage() AS (lines: chararray);`

To specify the data-schema for loading: `book = LOAD 'MyBook' AS (name, author, edition, publisher);`

Store Command Pig provides the store statement for writing the processed data after the processing is complete. It is the mirror image of the load statement in certain ways.

By default, Pig stores data on HDFS in a tab-delimited file using PigStorage:

`STORE processed into '/PigDemo/Data/Output/Processed';`

To store in HBaseStorage with a using clause: `STORE processed into 'processed' using HBaseStorage();`

To store data as comma-separated text data, PigStorage takes an argument to indicate which character to use as a separator: `STORE processed into 'processed' using PigStorage(',') ;`

Dump Command Pig provides dump command to see the processed data on the screen. This is particularly useful during debugging and prototyping sessions. It

can also be useful for quick adhoc jobs.

The following command directs the output of the Pig script on the display screen:

```
DUMP processed;
```

Relational Operations

The relational operations provided at Pig Latin operate on data. They transform data using sorting, grouping, joining, projecting and filtering. Followings are the basic relational operators:

Foreach FOREACH gives a simple way to apply transformations based on columns. It is Pig's projection operator. Table 4.17 gives examples using FOREACH.

Table 4.17 Applying transformations on columns using FOREACH operator

Load an entire record, but then remove all but the name and phone fields from each record	A = load 'input' as (name: chararray, rollno: long, address: chararray, phone: chararray, preferences: map []); B = foreach A generate name, phone;
Tuple projection using dot operator	A = load 'input' as (t:tuple (x:int, y:int)); B = foreach A generate t.x, t.\$1;
Bag projection	A = load 'input' as (b:bag{t:(x:int, y:int)}); B = foreach A generate b.x;
Bag projection	A = load 'input' as (b:bag{t:(x:int, y:int)}); B = foreach A generate b.(x, y);
Add all integer values	A = load 'input' as (x:chararray, y:int, z:int); A1 = foreach A generate x, y + z as yz; B = group A1 by x; C = foreach B generate SUM(A1.yz);

Filter FILTER gives a simple way to select tuples from a relation based on some

specified conditions (predicate). It is Pig's *select* command.

Loads an entire record, then selects the tuples with marks more than 75 from each record	<pre>A = load 'input' as (name:chararray, rollno:long, marks:float); B = filter A by marks > 75.0;</pre>
Find name (chararray) that do not match a regular expression by preceding the text without a given character string. Output is all names that do not start with P.	<pre>A = load 'input' as (name:chararray, rollno:long, marks:float); B = filter A by not name matches 'P.*';</pre>

Group GROUP statement collects records with the same key. There is no direct connection between group and aggregate functions in Pig Latin unlike SQL.

Collects all records with the same value for the provided key into a bag. Then it can pass to aggregate function, if required or do other things with that.	<pre>A = load 'input' as (name: chararray, rollno:long, marks: float); grpds = group A by marks; B = foreach grpds generate name, COUNT(A);</pre>
---	---

Order by ORDER statement sorts the data based on a specific field value, producing a total order of output data.

The syntax of order is similar to group. Key indicates by which the data sort.	<pre>A = load 'input' as (name: chararray, rollno: long, marks: float); B = order A by name;</pre>
To sort based on two or more keys (For example, first sort by, then sort by), indicate a set of keys by which the data sort. No parentheses around the keys when multiple keys indicate in order	<pre>A = load 'input' as (name:chararray, rollno:long, marks:float); B = order A by name, marks;</pre>

Distinct DISTINCT removes duplicate tuples. It works only on entire tuples, not

on individual fields:

Removes the tuples having the same name and city.	A = load 'input' as (name: chararray, city: chararray); B = distinct A;
---	--

Join JOIN statement joins two or more relations based on values in the common field. Keys indicate the inputs. When those keys are equal, two tuples are joined. Tuples for which no match is found are dropped.

Join selects tuples from one input to put together with tuples from another input.	A = load 'input1' as (name:chararray, rollno:long); B = load 'input2' as (rollno:long, marks:float); C = join A by rollno, B by rollno
Also based on multiple keys join. All cases must have the same number of keys, and they must be of the same or compatible types.	A = load 'input1' as (name: chararray, fathername: chararray, rollno: long); B = load 'input2' as (name: chararray, rollno: long, marks: float); C = join A by (name, rollno), B by (name, rollno)

Pig also supports *outer joins*. Tuples which do not have a match on the other side are included, with null values being filled for the missing fields in outer joins. Outer joins can be *left*, *right* or *full*. A left outer join means tuples from the left side will be included even when they do not have a match on the right side. Similarly, a *right* outer join means tuples from the right side will be included even when they do not have a match on the left side. A full outer join implies tuples from both sides are taken even when they do not have matches.

Limit LIMIT gets the limited number of results.

Outputs only first five tuples from the relation.	A = load 'input' as (name: chararray, city: chararray); B = Limit A 5;
---	---

Sample SAMPLE offers to get a sample of the entire data. It reads through all of the data but returns only a percentage of rows on random basis. Thus, results of a script with sample will vary with every execution. The percentage it will return is expressed as a double value, between 0 and 1. For example, 0.2 indicates 20%.

Outputs only 10% tuples from the relation	A = load 'input' as (name:chararray, city: chararray); B = sample A 0.1;
---	--

Split SPLIT partitions a relation into two or more relations

Outputs A relation A splits into two relations P and Q	A = load 'input' as (name:chararray, rollno:long, marks:float); Split A into P if marks >50.0, Q if marks ≤ 50.0;
--	---

Parallel PARALLEL statement is for parallel data processing.

Any relational operator in Pig Latin can attach PARALLEL. However, it controls only reduce-side parallelism, so it makes sense only for operators that force a reduce phase, such as group, order, distinct, join or limit.

Generating MapReduce job with 10 reducers	A = load 'input' as (name: chararray, marks: float); B = group A by marks parallel 10;
---	--

EVAL Functions Following are the evaluation functions:

Function Name	Description
AVG	Compute the average of the numeric values in a in a single-column bag
SUM	Compute the sum of the numeric values in a single-column bag
MAX	Get the maximum of numeric values or chararrays in a single-column bag
MIN	Get the minimum of numeric values or chararrays in a single-column bag
COUNT and COUNT_STAR	Count the number of tuples in a bag

CONCAT	Concatenate two fields. The data type of the two fields must be the same, either chararray or bytearray.
DIFF	Compare two fields in a tuple
IsEmpty	Check if a bag or map is empty (has no data)
SIZE	Compute the number of elements based on the data type
TOKENIZE	Split a string and output a bag of words

Piggy Bank *Pig* users share their functions from Piggy Bank. *Register* is keyword for using Piggy bank functions.

User-Defined Functions (UDFs) A programmer defines UDFs which perform functionalities not present as built-in Pig function. A programmer can use UDFs for filtering data or performing further analysis. A programmer can write UDF using a programming language, such as Java, Python, Ruby, Jython or JRuby.

A UDF should extend a Filter function or Eval function and must contain a core method called *exec*, which contains a Tuple.

The UDF class extends the *EvalFunc* class which is the base for all Eval functions. All evaluation functions extend the Java class ‘*org.apache.pig.EvalFunc*’. It is parameterized with the return type of the UDF which is a Java String in this case.

Filter functions are Eval functions that return a Boolean value. The UDF class when extends the *FilterFunc* class can be used anywhere a Boolean expression is appropriate, including the FILTER operator or Bincond expression.

The following example gives the codes for developing a user-defined function (UDF) returning Boolean after checking the age.

EXAMPLE 4.17

Write a UDF ‘IsCorrectAge’ which checks if the age given is correct or not. UDF should return a Boolean value: True or False. If the Tuple is null or zero then also it should return False. Use Java. Create a JAR file and then export. Later register the JAR file. The JAR files are in the library files of Apache Pig at the time of loading.

SOLUTION

Followings are the codes for the user-defined function IsCorrectAge.

```
public class IsCorrectAge extends FilterFunc {  
    @Override  
    public Boolean exec (Tuple tuple) throws IOException {  
        if (tuple == null || tuple.size() == 0) {  
            return false;  
        }  
        try {  
            Object object= tuple.get(0);  
            if (object == null) {  
                return false;  
            }  
            Int i = (Integer) object;  
            if (i == 18 || i == 20 || i == 21 || i == 25) {  
                return true;  
            }  
            else {  
                return false;  
            }  
        } catch (ExecException e) {  
            throw new IOException(e);  
        }  
    }  
}
```

Once IsCorrectAge create, the following command registers a JAR file into the library of JAR files:

```
register myudf.jar;  
A = load 'input' as (name chararray, age int);  
X = filter A by IsCorrectAge(age);
```

Self-Assessment Exercise linked to LO 4.5

1. How does Apache Pig execution engine function for faster data processing?

2. List the Grunt shell commands and the use of each command.
3. When is Hive and when is Pig used?
4. How are tuple and map used?
5. How are projections used?
6. Write the functions of GROUP, JOIN, FILTER, LIMIT, ORDER BY, PARALLEL, SORT and SPLIT.
7. How will a UDF return the difference of maximum and minimum sales from sales data values in Pig Latin?



KEY CONCEPTS

aggregation

bag

BLOB

bucketing

collating

collection data type

combining

command line interface

composing

cross correlation

data definition

data manipulation language

deserializer

diagnostic operator

difference

dynamic partition

EVAL

filtering
graph processing
Group By
grouping by keys
having()
Hive
Hive data units
Hive File Format
HiveQL
inner Join
InnerSplit
intersection
iterative message passing
JobTracker
key-value pair
left Join
managed table
map (a Pig data type)
MapReduce metadata
Metastore
natural Join
ORC
outer Join
parallel tasks
parsing
partitioning
Pig
Piggy Bank

Pig Latin
projection
querying table
RCFile
RecordReader
relation
relational operator
right join
sequential file
serializer
shuffle and store
sorting
SQL-like script
static partition
tuple
user-defined function
views



LO 4.1

1. An *application* consists of a number of tasks. A MapReduce program for an application task is termed as a job. Each job consists of several smaller units, called MapReduce tasks. They run in parallel for the application task. MapReduce programming is a software execution framework that defines the parallel tasks, the results combine and application obtains the consolidated result.

2. MapReduce implements a data model, which represents data as key-value pairs.
3. Reduce task implements using Reducer function that takes Mapper output (which is shuffled and sorted), that is grouped key-value data (k_2, v_2) and applies it in parallel to each group. Another set of key-value pairs (k_3, v_3) are the final output file.
4. Coping with node failures is done by the TaskTracker, which when fails to communicate with the JobTracker for a pre-defined period, the JobTracker restarts.

LO 4.2

1. MapReduce functions have a number of applications:
 - (a) Counting, summing, run algorithms for the relational algebra operations, projections, union, intersection, natural Join, grouping and aggregation.
 - (b) Collating, filtering and parsing. Collating is a method to collect all the items which have same value of function.
 - (c) Graph processing using iterative message passing.
 - (d) Web Indexing also uses the method of iterative message-passing. A state of each entity calculates based on characteristics of the other entities in its neighborhood in a given network of entities and relationships between them.
 - (e) Multiplication of matrix with a vector and of matrix with a matrix.
2. When multiplying two matrices, two cascaded MapReduce operations require much less memory than a single step MapReduce.

LO 4.3

1. Apache Hive is an open-source data-warehouse software. Data summarization, analysis and querying are major functions of Hive. Hive facilitates reading, writing and managing large datasets residing in distributed Hadoop files using SQL-like scripts. Hive supports serialization,

deserialization and user-defined functions.

2. Hive includes a system catalog, called Hive Metastore. Hive provides increased flexibility in schema design.
3. Hive supports primitive and collection data types. Hive supports text files, sequence Files (consisting of binary key/value pairs), RCFiles (Record Columnar Files), ORC (optimized row columnar) and HBase file format types. Hive considers database, tables, partitions, bucketed tables and buckets as data units.

LO 4.4

1. HiveQL has SQL-like script statements for (i) data definition, (ii) data manipulation, (iii) creating, dropping, and using the databases and tables, (iv) selection by where, GroupBy and Having clauses.
2. The partitions are must in large dataset tables in the databases of a data warehouse. Hive command creates partitions. HiveQL commands create buckets, views and sub-queries.
3. HiveQL has the command provision for join, sorting and aggregation.
4. HiveQL plug-ins the custom MapReduce scripts into queries.

LO 4.5

1. Pig is an open-source high-level language platform. Pig applications are mainly for analyzing large datasets. Pig executes queries in the HDFS environment. Processes any kind of data: structured, semi-structured or unstructured data from various sources.
2. Pig language used is known as Pig Latin. Pig Latin programming is in Java. Pig is SQL-like query language applied on a larger dataset, and provides additional features. Pig Grunt shell enables development. Pig Grunt shell enable development of Pig Latin scripts.
3. Pig converts all the operations into Map and Reduce tasks that process on

Hadoop efficiently. Programmers write scripts using Pig Latin to analyze data. The scripts internally convert into Map and Reduce tasks.

4. Pig application is ETL operations (Extract, Transform and Load). The language allows a detailed step-by-step procedure by which the data must be transformed. Pig is designed to handle any kind of data. Pig programming language can handle inconsistent schema data as well.
5. Helps programmers write complex data transformations without knowing Java. Possess a rich set of built-in data types, such as Bag (collection of tuples) and Map (set of key-value pairs). Possess a rich set of built-in operators, such as group, join, filter, limit, order by, parallel, sort and split.
6. Allows programmers to write User-Defined Functions (UDF) to write custom functions in other programming languages, such as Java, Python, Ruby, Jython or JRuby. UDF easily embeds in Pig scripts and provides extensibility to Pig.

Objective Type Questions

Select one correct-answer option for each of the following questions:

- 4.1 (i) A user application specifies the input/output data locations, (ii) The application supplies map and reduce functions by the implementation of appropriate interfaces and/or abstract classes, (iii) Application task configures the job and specifies other job parameters, (iv) Map takes output dataset as pieces of data from the Reducer and maps them on various nodes for parallel processing, and (v) The reduce task, which takes the input at Mapper combines those data pieces into a smaller set of data.
- (a) ii and iii
(b) all
(c) i, ii and iii
(d) i, iv and v
- 4.2 (i) MapReduce implies, the reduce task is mostly performed after the map

task, (ii) Map takes input dataset as pieces of data and maps them on various nodes for sequential processing,
(iii) MapReduce framework may not operate entirely on (key-value) pairs, and (iv) The framework views the output to the task as a set of (key, value) pairs and produces a set of (key, value) pairs as the input of the task, possibly of different types.

- (a) none
- (b) only iv
- (c) only ii
- (d) all

4.3 (i) Partitioner does the partitioning, (ii) The partitions are the semi-mappers in MapReduce,

(iii) Combiners are semi-reducers in MapReduce, (iv) Combiners process the input of map tasks before submitting it to Reducer tasks, (v) Reduce task implements using Reduce function (or Reducer) that takes Mapper output (which is shuffled and sorted), that is (grouped key-value data) (k_2, v_2) and applies it in parallel to each group, and (vi) Reduce function iterates over the list of values associated with a key and produces outputs, such as aggregations and statistics.

- (a) all except ii and iii
- (b) all
- (c) i to v
- (d) all except iv

4.4 MapReduce program composes the (i) Count, (ii) find distinct values, (iii) search unique value, (iv) group using attributes, and does aggregating, (v) summing, (vi) relational-algebra operations, (vii) projections, (viii) union, (ix) intersection, (x) difference, (xi) natural Join, (xii) multiplication of two matrices, and (xiii) multiplication of matrix and vector.

- (a) all except ii, iii, iv, xii and xiii

- (b) all
- (c) all except ii to vi
- (d) all except xii and xiii

4.5 (i) Graph processing needs iterative message passing, (ii) Graph processing uses are in web indexing, (iii) A state of each entity calculates based on characteristics of the other entities in its neighborhood in a given network of entities and relationships between them, (iv) Mapper class emit() emits the messages for each node using ID of the non-adjacent node as a key, (v) All messages groups by the incoming node, and (vi) Reducer class method computes the state again and rewrites a node with the new state.

- (a) i, iii, iv and vi
- (b) i to v
- (c) ii to vi
- (d) All except iv

4.6 Hive (i) does not have commands to update or delete using the record level queries, (ii) does not support transactions on the DB, (iii) supports to create, drop and use functions, (iv) latency for query operations is much less than a second for Big Data of petabytes, and (v) provides limited JDBC/ODBC connectivity functions.

- (a) i, iii, iv and vi
- (b) iii to v
- (c) ii to vi
- (d) All except iv

4.7 Hive architecture consists of (i) Hive Server, (ii) CLI, (iii) web interface, (iv) metastore, and
(v) Hive driver. (vi) Usages of Hive metastore are to provide names of tables, databases, columns in a table.

- (a) ii to v
- (b) all

(c) ii to iv

(d) i, ii, iv, v

4.8 HiveQL data manipulation commands are (i) USE, (ii) DROP DATABASE, (iii) DROP SCHEMA, (iv) ALTER TABLE, (v) DROP TABLE, (vi) DELETE TABLE, (vii) DELETE DATABASE, (viii) INSERT TABLE, (ix) INSERT DATABASE, and (x) LOAD DATA.

(a) all except ii, iii and v

(b) all except i, iii

(c) all except vi to ix

(d) i to iv

4.9 HiveQL (i) Join clause combines and retrieves the records from multiple tables, (ii) Join is same as OUTER JOIN in SQL, (iii) JOIN condition uses primary keys and foreign keys of the tables, (iv) JOIN clause combines the columns of two or more tables, based on a related column between them, (v) JOIN is same as SQL JOIN, (vi) A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate, (vii) A RIGHT JOIN returns all the values from the right table, plus not matched values from the left table, or NULL in case of no matching join predicate, (viii) FULL OUTER JOIN combines the records of both the left and the right outer tables those fulfill the JOIN condition, and (ix) the joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

(a) all except v and vii

(b) all

(c) all except iii and x

(d) i to vii

4.10 Pig (i) reads the input data files from HDFS or the data files from other sources such as the local file system, (ii) stores the intermediate data and writes back the output in HDFS, (iii) processes any kind of data: structured,

semi-structured or unstructured data coming from various sources, (iv) allows programmers to write User-Defined Functions (UDFs) to write custom functions,

(v) UDFs written in several other programming languages, (vi) UDF easily embed in Pig scripts written in Linux, (vii) exploits multi-query approach, thereby reducing the length of codes; ten lines is equal to MapReduce code of two hundred lines, which enables spreads processing, and (viii) Pig read data, processing, programming the UDFs in multiple languages and programming multiple queries by fewer code enabling fast processing are guided by four philosophies: live anywhere, take anything, domestic and run like flying.

- (a) i to vi
- (b) all except vi
- (c) all except iii to v
- (d) all

4.11 Pig (i) helps programmers write complex data transformations using scripts (without using Java) that possess a rich set of built-in operators such as (ii) Bag, (iii) BLOB, (iv) Map, (v) Group, (vi) Join, (vii) Filter, (viii) Limit, (ix) Order by, (x) parallel, (xi) sort and (xii) split.

- (a) all except i
- (b) all except i, x and xii
- (c) all except ii, iii and iv
- (d) ii to ix and xi

4.12 Pig (i) is a dataflow language, (ii) low level language, (iii) performs Join, filter, sorting or ordering operations, (iv) uses multi-query approach, thereby increasing the length of the codes, (v) no need for compilation, (vi) on execution, operators convert internally into a MapReduce job, (vii) provides nested data types like tuples, bags, and maps. *MapReduce* on the other hand, (viii) is a data processing paradigm, (ix) relatively difficult to

perform Join, filter, sorting or ordering operations between datasets, (x) Complex Java implementations require exposure to Java language, (xi) Jobs have a long compilation process, and (xii) nested data types, such as tuples, buckets and views.

- (a) all except ii, iv and xii
- (b) all except v, x, xi and xii
- (c) i to x
- (d) all except vi and xi

Review Questions

- 4.1 List and explain the features of the MapReduce programming model? How does MapReduce program enable parallel processing? **(LO 4.1)**
- 4.2 How does a Map task implement using key-value pairs in an input file? What are the uses of Shuffle in processing the aggregates for all the Mapper output by grouping key values of the Mapper output and the value which gets appended in a list of values? **(LO 4.1)**
- 4.3 How does ‘Group By’ operate for creating Mapper output? What are the roles of partitioning and combining? **(LO 4.1)**
- 4.4 How does MapReduce program find the distinct values and count the unique values? **(LO 4.2)**
- 4.5 How does the MapReduce implement the relational algebraic functions, union, projection, difference, intersection, natural join, grouping and aggregation? Explain each with an example. **(LO 4.2)**
- 4.6 How do MapReduce tasks implement a matrix multiplication by a vector? **(LO 4.2)**
- 4.7 Describe the Hive architecture components. Why are HiveQL, SQL-like scripts used in place of RDBMS, such as MySQL for Big Data? **(LO 4.3)**
- 4.8 What are the types of built-in functions available in Hive? What are the uses of each of these? **(LO 4.3)**

- 4.9 Why should partitions be created in databases and tables in Hive data warehouse for very large datasets? **(LO 4.4)**
- 4.10 What are aggregation commands provisioned in HiveQL? What are the partitioning commands? **(LO 4.4)**
- 4.11 An enterprise needs to create and use a Hive data warehouse with very large databases and tables. Why are the usages of RCFile and ORCFile formats, creation of large number of partitions, buckets and views required in the databases and tables? **(LO 4.4)**
- 4.12 What are the differences between Pig programming model with MapReduce, relational database and Hive programming models? **(LO 4.5)**
- 4.13 Describe Pig data types and operators: Group, Join, Filter, Limit, Order by, parallel, sort and split. **(LO 4.5)**
- 4.14 Describe usages of Pig operations: parallel, split and defining a UDF. Give one example of each. **(LO 4.5)**

Practice Exercises

- 4.1 MapReduce program imports the following at the beginning:

```
import org.apache.hadoop.fs.Path  
import org.apache.hadoop.mapreduce.Mapper  
import org.apache.hadoop.mapreduce.Job  
import org.apache.hadoop.mapreduce.Reducer  
import org.apache.hadoop.io.Text  
import  
org.apache.hadoop.mapreduce.lib.input.FileInputFormat  
import  
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat  
import org.apache.hadoop.fs.Path
```

What are the functions that each Class provides in the program? **(LO 4.1)**

4.2 A company manufactures and sells cars through a large number of showrooms. Each car showroom records in main table and transaction tables. Recapitulate Practice Exercise 3.3 Table data. Describe the steps for composing MapReduce program parallel tasks for calculating aggregated annual sales of each model for all showrooms. **(LO 4.2)**

4.3 Recapitulate Section 4.3.4. Consider that two MapReduce cascaded programs multiply 3×4 matrix A with another matrix 4×6 matrix B. Calculate the number of tuples in each matrix, tuples after natural join. List each step for using these tuples, grouping and aggregation of tuples with attributes. Now calculate these numbers again for 8192×4096 matrix multiplication by 4096×32768 matrix. **(LO 4.2)**

4.4 Install Hive and demonstrate usages of each data type and collection types listed in Tables 4.6 and 4.7. **(LO 4.3)**

4.5 Create a HiveQL table for grade-sheet of your five-course semester examination with SGPA (Semester Grade Point average) in a semester. Now, write commands to create partitions in the table in RCFile formats. How will the table serialize? **(LO 4.4)**

4.6 Insert the Hive-table above in all University students' data warehouse. Write commands for joining four tables for four-semester examinations. How will that be used for calculating CGPA (Cumulative Grade Point average)? **(LO 4.4)**

4.7 Recapitulate Examples 4.10 to 4.13. Create a HiveQL data warehouse for toys_company manufacturing 1600 different toys and 2000 puzzle product categories, up to 20 product types for each, and each puzzle product of product types 100, 200, 400, 800, 1600, 2400 and 500 pieces. **(LO 4.4)**

4.8 Recapitulate Example 1.6. Create Pig user-defined functions (UDFs) for selecting the sales of each flavour of chocolate from the multiple ACVMs. **(LO 4.5)**

4.9 Select and list the Pig data types, operations and their usages during

processing the data tables created in Practice Exercise 4.7. (LO 4.5)

Note:

○○● Level 1 & Level 2 category

○●● Level 3 & Level 4 category

●●● Level 5 & Level 6 category

Chapter 5

Spark and Big Data Analytics

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 5.1 Get understanding of the Spark architectural features, software stack components and their functions
- LO 5.2 Get knowledge of analysis steps using Spark, Spark along with Python, advanced features, UDFs, vectorized UDFs, grouped vectorized UDFs and Python analytics libraries
- LO 5.3 Get understanding of the methods of downloading Spark, getting started in programming with Spark, Spark shell, Spark context, developing and testing codes, programming with RDDs and the applications of MLlib
- LO 5.4 Get understanding of the ETL processes using built-in functions, operators and ETL pipelines
- LO 5.5 Get Introduced to analytics, data/information reporting and visualizing methods

RECALL FROM EARLIER CHAPTERS

► CHAPTER 1

Spark, Spark SQL and Apache Drill are advanced processing methods for Big Data. They also enable real-time processing. Berkeley Data Analytics Stack (BDAS) is an open-source data analytics stack. The stack consists of number of software components and frameworks for complex computations using

Big Data.

► CHAPTER 2

The four layers of the Hadoop ecosystem are:

1. Data store layer: Stores Big Data HDFS.
2. Data processing layer: Processes the stored data using programs, such as MapReduce, YARN, HBase and Cassandra.
3. Applications support layer: APIs supporting the processing of applications at the data processing layer, such as Pig, Hive, HiveQL, Sqoop, Ambari, Chukwa.
4. Applications layer: Tools such as Spark, Flink, Flume, Mahout, and Processes ETL, Analytics, BP, BI, Data Visualization, R-Descriptive Statistics, Machine learning, Data mining (Section 2.2.3 and Figure 2.3).

► CHAPTER 3

When the Big Data Store is at clusters HDFS, the applications access the data sequentially. When it is using NoSQL databases, the data read/write access by applications is random-access. The access to a resource is as per the specified resource pointer (address) for the access.

► CHAPTER 4

MapReduce tasks processes in parallel and in a distributed environment. A program composes the MapReduce tasks for the calculations and uses the relational-algebraic operations, ‘grouping by’ and aggregation functions (Section 4.3).

Hive creates databases which load into the enterprise data warehouse. Hive composes the queries and does data aggregation and summarization (Section 4.4). HiveQL functions query the DBs, tables, partitions and buckets, and executes the SQL like operations and UDFs (Section 4.5).

Pig functions executes query on large datasets which are stored in HDFS. Pig programming model enables writing complex data transformations without knowing Java [due to a rich set of built-in functions and operators such as group, join, filter, limit, order by, parallel, sort and split, and possessing of a rich set of built-in data types such as Bag (collection of tuples) and Map (set of key-value pairs)] (Section 4.6).

This chapter focuses on Apache Spark using the data sources at HDFS, any Hadoop compatible data source, such as HBase, Cassandra and Ceph, or Object Store S3. Spark

provides in-memory, distributed and faster cluster-computing, and consists of APIs in Java, Scala, Python and R.

5.1 | INTRODUCTION

Pig or Hive are high-level scripting languages that are used with the Apache Hadoop. They have SQL like commands for queries. The commands before executing, translate to Map and Reduce parallel-tasks. They process the queries, built-in functions, aggregation operations and User Defined Functions (UDFs). They enable ease in programming for these functions. They run ETL processes using Big Data Store.

Pig and Hive programs use complex data types and operations. The scripts and programs use the datasets distributed in the HDFS Data Store.

Applications such as data analytics, stream analytics and graph analytics, and machine learning require the following:

In-memory processing: In-memory processing is fast when compared to processing data most of the times, from the disk or remotely distributed nodes. This is because the processor takes much less time in accessing the memory compared to the disk or remote data node. In-memory processing also facilitates real-time processing and streaming data analysis. DAG-based acyclic data flow further boosts the processing speed.

Application tasks processing Framework: Application tasks require processing in a framework which uses HDFS as well Hadoop compatible data sources, such as HBase, Cassandra, Ceph, cloud-based Objects Store Service or Amazon S3. The tasks require support which facilitates running the Hive, Pig, and other Hadoop ecosystem tools in Java, Python, R and Scala. APIs using Python shell and Scala shell facilitate the interactive running of the applications. Many applications such as statistical, mathematical and graph analytics, and machine learning algorithms require APIs designed in these languages.

These features ease the programming for complex analytics, machine learning and other solutions.

Advent of Apache® Spark™

Berkeley's Algorithms, Machines and Peoples Laboratory (AMP) developed Berkeley Data Analytics Stack (BDAS) which support efficient, large-scale in-memory data processing, and includes applications fulfilling three fundamental processing requirements: accuracy, time and cost. AMP first developed Spark in 2009 and later passed on the project to Apache. A new version is Spark 2.3.1.

Apache® Spark™ uses in-memory data processing. Thus, processing is fast since there is

no delay. The reason is that processor in-memory read and write operations are fast compared to read from disk and write to disk. Apache® Spark™ uses the DAGs and acyclic data-flows, and data from HDFS compatible data sources and cloud-based Data Stores. It provides APIs for programming in R, Python, Java and Scala.

Open Source Analytics Tools

Following are the tools:

1. R and its library provide various statistical analysis functions. R now analyses large data sets also since R integrates with Big Data platforms, such as Spark.
2. Python is a widely used language due to its analysis and statistics libraries, such as numpy, scipy, scikit-learn, pandas, StatsModel.
3. Storm is for real-time continuous data streams.
4. Pig is a data flow language with SQL like operations and uses UDFs. Pig enables easy coding compared to MapReduce for the complex tasks.
5. Hive is for creation of data warehouse, integration of databases and applications, and uses SQL like scripts and UDFs. Coding is easy compared to MapReduce.

Pandas is an open source Python package, and consists of BSD-licensed library functions using the Panda (Panel Data). (5.3.2.1) The Pandas give high performance, easy-to-use data structures and data analysis tools. Pandas enrich the Python programming language.

The most popular open source analytics tools are Apache Spark, Python, R, Apache Pig and Hive, according to a study.

Spark is for high volume unstructured data. Spark seamlessly integrates with Spark SQL which uses the structured data, Spark Streaming is for streaming data, Spark GraphX for graph databases, Spark MLlib for machine-learning library, and Spark Arrow for columnar in-memory analytics. Spark provides easy programmability with inclusion of APIs for programmers to develop applications in Python, R, Java or Scala.

Reader needs to learn the following new select key term, and their meanings besides the ones given in the previous chapters:

User Defined Functions (UDFs) refer to custom functions which are not built-in a programming language but user adds them and they can be written in a language, such as Java, Python, Ruby, Jython, JRuby or Scala. They easily embed into scripts written in that programming language. Examples of languages with provisions of UDFs are Hive, Pig and Spark. The UDFs provide extensibility to the programming language.

Vectorized UDFs (VUDFs) refer to custom functions using series data-structure (meaning

one dimensional array or tuples).

Grouped Vectorized UDFs (GVUDFs) refer to custom functions written using DataFrame as inputs.

DataFrame in Spark refers to a distributed collection of data that organizes into the named columns. The concept of the DataFrame in Spark is similar to database table in a relational database. The data frame concept in R is the basis of the data frame concept in Spark. Scala and Java APIs for DataFrames are just dataset of rows.

SchemaRDD is the name of Spark DataFrame in the earlier versions of Spark.

SerDe refers to Serializer/Deserializer functions (methods). Java syntax is SERDE, ‘serde.class.name’. SerDe use in codes for obtaining records from unstructured data. The serializer function saves the records and the deserializer function loads (extracts) the records.

Data pipeline means data collected from various data sources passes through in-between phases (stages) of processing. The output of each stage is the input to the next in the pipeline. Processing in-between uses a chain of function calls in an application or process, such as ETL.

Graph refers to a non-linear data structure with properties attached to each vertex and edge. Computations perform at each node in a graph structure using path traversals between the vertices (Section 8.2).

Directed Acyclic Graph (DAG) refers to a directed graph with no cyclic traversal. Here, one set of inputs simultaneously applies at a DAG node input, and after the operations (computations) at the node, only one set of outputs is generated. The node represents the statements and operators, which execute at the node in the graph.

Nested tables in databases refer to one column tables. Oracle RDBMS uses PL/SQL. A database stores the rows of a nested table in no particular order. While the SQL assigns the rows in consecutive subscripts starting at 1 so that each row accesses like an array element, PL/SQL accesses a nested table in no order.

Parquet refers to nested hierarchical columnar storage in group of rows, wherein each row has a number of columns, each column has one chunk, and each chunk has a number of pages.

In-memory refers to data read from the memory during computations and data written to the memory at same data node, thus ensuring fasten memory accesses. Disk accesses and remote node accesses make computations slow.

Columnar in-memory analytics refers to usages of optimized layout columnar tables (nested tables, Hive ORC tables). That provides the easier data locality using successive memory addresses. The CPUs and GPUs provide higher performance for native

vectorized optimization during analytics and OLAP. Apache Arrow™ enables usages of in-memory columnar analysis and grouped vectorized UDFs.

ETL (Extract, Transform and Load) refers to operations on a database or Data Store using a tool or program (maximum) to code up to few thousand lines of code. ETL tools pull the data from various data sources and store (load) it in the appropriate Data Store after applying the required transformation operation.

Shell means an environment to write and run programming scripts; for example the scripts for query processing similar to SQL.

Schema refers to a blueprint for organization or structuring of a database or table or dataset. The blueprint tells how the database constructs. A construction may use division of the data into rows. Relational-database construction may use the division into database tables. Schema for a database is defined as set of formulae, called integrity constraints, imposed. (Formulae may be just sentences.)

SQL refers to a language for (i) writing structured queries for processing using a relational database; (ii) schema creation, schema modifications and data access control; (iii) creating client for sending query scripts, creating server databases and managing the databases; and (iv) viewing, querying and changing (update, insert or append or delete) the databases.

Metastore refers to the system objects, files, catalog, schema or tables, databases, columns in a table, their data types, and mapping with HDFS or any other storage formats. Metastore provides access to them during computation.

Cassandra refers to a distributed DBMS designed for handling a high volume of structured data across multiple servers. Cassandra is HDFS compatible. Cassandra DBs distribution model is peer-to-peer distribution in a system across its nodes. Data distributes among all the nodes in a cluster.

Software stack refers to a group of programs. Stack programs work in tandem (together or in conjunction) and produce a result. Software stack also refers to any set of applications that works in a specific and defined order. For example, LAMP is a software stack that consists of a group of open source components, namely Linux, Apache, MySQL, Perl, PHP or Python.

Ad hoc query refers to a “for this purpose” query, “on the fly” query or a “just so” query. It’s the kind of SQL query that is loosely used when required. For example, var newSqlQuery = “SELECT * FROM table WHERE id = ” + toy_puzzleId. It will be different each time this code executes, depending on the value of toy_puzzleId (Example 4.7).

This chapter focusses on Spark and data analysis with Spark. Section 5.2 introduces