The Six Pillars for Building Big Data Analytics Ecosystems

SHADI KHALIFA, YEHIA ELSHATER, KIRAN SUNDARAVARATHAN, APARNA BHAT, PATRICK MARTIN, and FAHIM IMAM, Queen's University
DAN ROPE and MIKE MCROBERTS, IBM, Washington D.C.
CRAIG STATCHUK, IBM, Canada

With almost everything now online, organizations look at the Big Data collected to gain insights for improving their services. In the analytics process, derivation of such insights requires experimenting-with and integrating different analytics techniques, while handling the Big Data high arrival velocity and large volumes. Existing solutions cover bits-and-pieces of the analytics process, leaving it to organizations to assemble their own ecosystem or buy an off-the-shelf ecosystem that can have unnecessary components to them. We build on this point by dividing the Big Data Analytics problem into six main pillars. We characterize and show examples of solutions designed for each of these pillars. We then integrate these six pillars into a taxonomy to provide an overview of the possible state-of-the-art analytics ecosystems. In the process, we highlight a number of ecosystems to meet organizations different needs. Finally, we identify possible areas of research for building future Big Data Analytics Ecosystems.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; H.2.8 [**Database Management**]: Database Applications; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; D.3.4 [**Programming Languages**]: Processors; D.4.7 [**Operating Systems**]: Organization and Design

General Terms: Big Data, Analytics, Workflow, Taxonomy, Scheduling, Storage, Intelligent Assistance

Additional Key Words and Phrases: Orchestration, analytics talent gap, consumable analytics

ACM Reference Format:
Shadi Khalifa, Yehia ElShater, Kiran Sundaravarathan, Aparna Bhat, Patrick Martin, Fahim Imam, Dan Rope, Mike McRoberts, and Craig Statchuk. 2016. The six pillars for building big data analytics ecosystems. ACM Comput. Surv. 49, 2, Article 33 (August 2016), 36 pages.
DOI: http://dx.doi.org/10.1145/2963143

1. INTRODUCTION

In the Big Data era, we all, in one form or another, participate in generating data. Big Data can be *structured*, which is generated by applications like Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems and typically stored in rows and columns with well-defined schemas. It can be *semi-structured,* which is generated by sensors, web feeds, event monitors, stock market feeds, and network and security systems. Semi-structured data usually have meta-data that describes

This work was supported by IBM CAS research and NSERC.
Authors' addresses: S. Khalifa, Y. ElShater, K. Sundaravarathan, A. Bhat, P. Martin, and F. Imam, School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada; emails: {khalifa, elshater, sundarav, bhat, martin, imam}@cs.queensu.ca; D. Rope and M. McRoberts, IBM, 600 14th St NW #300, Washington, DC 20005, United States; emails: {drope, mtm}@us.ibm.com; C. Statchuk, IBM, 340 Albert St #1610, Ottawa, ON K1R 7Y6, Canada; email: Craig.Statchuk@ca.ibm.com.

DOI: http://dx.doi.org/10.1145/2963143

their structure; however, this structure does not always fit in rows and columns. Big Data can also be *unstructured*, which is typically generated by people in forms such as emails, social media, text documents, videos, audio, and images.

Along with having a variety of data formats, Big Data is generated in huge *volumes* at a rapid *velocity* with no obvious way of telling the *veracity* of the collected data. With such properties, data has outgrown the ability to be stored and processed by many traditional systems [Manyika et al. 2011].

The value of data is realized through insights, taking into consideration that the utility of some data points declines very quickly. Increasingly, organizations' success has become dependent on how quickly and efficiently they can turn the petabytes of data they collect into actionable information [Turner et al. 2012].

Analytics is a multi-disciplinary art for extracting useful insights from data by discovering hidden patterns and using those patterns to predict the likelihood of future events. Analytics is the application of computer science, data storage, data mining and machine learning, statistical analysis, artificial intelligence and pattern recognition, visualization, operations research, and Business Intelligence (BI) on real-world data to better understand data-oriented problems [Turner et al. 2012].

A *software ecosystem*, in general, consists of the set of software solutions that enable, support, and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions [Bosch 2009]. A *Big Data Analytics Ecosystem*, in particular, is a set of software solutions to support the activities associated with transforming raw Big Data into meaningful insights. These activities typically include the following [Chapman et al. 2000]:

—*Data Exploration*: Analysts go through the data, using *ad-hoc queries* and *visualizations,* to better understand the data;
—*Data Preparation*: Analysts clean, prepare, and transform the data for modeling using *batch processing* to run computational and IO intensive operations;
—*Modeling/Scoring*: Data models are trained, using *iterative processing,* on the prepared data and trained models are used to score the unlabeled data.

For a Big Data Analytics Ecosystem, all these phases need to run in a distributed parallel way and to handle the variety of data formats. The efficiency of the ecosystem, however, is dependent on the efficiency of its components. Thus, one weak component and the whole ecosystem will not function efficiently. This brings us to the main problem with research in this field that motivated us to write this article; there is a need to study the whole ecosystem to consider how each component affects the whole.

Given the already significant number of research papers related to Big Data Analytics, this work aims to provide a clear overview of the work on the different components of the Big Data Analytics Ecosystems. This article targets data engineers interested in the "How" aspect of Big Data Analytics; for example, how to run analytics on Big Data, how to store Big Data, and how to manipulate Big Data. Discussing the data science of defining "Which" data mining algorithm to use for the different use cases is out of the scope of this survey.

Our survey differs from previously published surveys in several ways. First, our survey examines all main components that must be present in an ecosystem to support Big Data Analytics, unlike other surveys [Doulkeridis and NØrvåg 2014; Liu et al. 2014; Bhatt and Kankanhalli 2011; da Silva et al. 2005; Serban et al. 2013; Mittelstadt et al. 2012] that only focus on single components. The first four of the other surveys only cover processing, while the fifth only covers assistance and the sixth only covers user interfaces. Second, we focus on the capabilities of Big Data Analytics Ecosystems where data is problematic to store and analyze where other surveys [Deelman et al. 2009; Barker and Van Hemert 2007; Chen et al. 2007; Yu and Buyya 2005] study

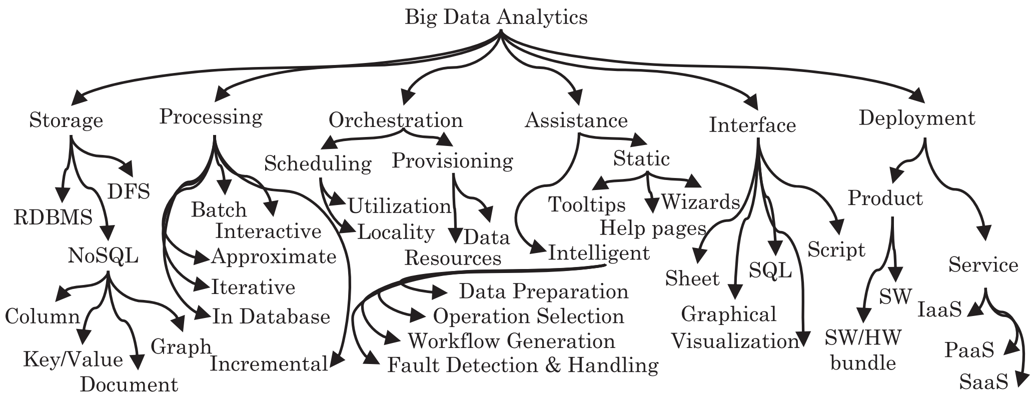

Fig. 1. The Big Data Analytics Ecosystem taxonomy.

analytics engines without regard for the additional challenges imposed by Big Data. The survey most similar to ours is the work by Babu and Herodotou [2013], which is over 100 pages. Our survey is more condensed and more focused on analytics for Big Data, which is not fully covered in the other survey.

The remainder of this article is structured as follows: Section 2 identifies six main capabilities, or pillars, required in a Big Data Analytic Ecosystem and presents a taxonomy based on the six pillars that is used to classify existing work in the area. Section 3 presents a set of ecosystems based on the six pillars to fit different business use cases. Section 4 identifies the open research problems that need to be addressed in future Big Data Analytics Ecosystems.

2. BIG DATA ANALYTICS ECOSYSTEM TAXONOMY

We survey current work in the area of Big Data Analytics Ecosystems from a practical perspective, namely the components necessary to deal with the challenges of volume, velocity, variety, and veracity inherent in Big Data. To present and compare the work, we organize the components into six capabilities or pillars of a Big Data Analytics Ecosystem. These pillars are the following:

—*Storage* that handles the data's huge volume, fast arrival, and multiple formats;
—*Processing* that meets the Big Data Analytics processing needs;
—*Orchestration* that manages available resources to reduce processing time and cost;
—*Assistance* that goes beyond the interface and provides suggestions to help users with decisions when selecting operations and building their analytics process;
—*User Interface* that provides users with a familiar environment to build and run their analytics;
—*Deployment Method* that provides scalability, security, and reliability.

We propose the taxonomy shown in Figure 1 based on the six pillars. The taxonomy is used to organize the different approaches used in each pillar. For each pillar we discuss the advantages and disadvantages of the approaches, the way in which the approaches address the Big Data challenges and highlight the design and implementation similarities and differences among the approaches.

2.1. Big Data Storage

Over the past decade, the amount of data organizations have to deal with has become phenomenal. Over time, the requirements for data storage changed to meet the exponential increase in data size, arrival speed, and number of data formats. In this section, we discuss *Relational Database Management Systems (RDBMS),* followed by *Distributed File Systems (DFS),* and ending with *Not-only Structured Query Language Systems (NoSQL)*.

2.1.1. Relational Database Management Systems (RDBMS). RDBMSs are designed to ensure the ACID (Atomicity, Consistency, Isolation, and Durability) properties for storing structured data. But, in this era of Big Data, these systems have to process large amounts of data with low latency while achieving high scalability. Recent RDBMSs developments promise enhanced performance and scalability with an advantage over NoSQL of providing the higher-level SQL language and ACID properties [Cattell 2011]. They also allow operations (e.g., joins) and transactions to span many nodes. Grolinger et al. [2013] provide a comprehensive comparison between NoSQL and NewSQL stores for interested readers.

MySQL Cluster[1] uses a shared-nothing architecture to shard data over multiple database servers, with replication to support recovery. *ScaleDB*[2] is similar to *MySQL Cluster*, except it implements a shared-data architecture, giving access to every disk from every server. While this approach limits its scalability, it allows using techniques like multi-table indexing that speeds up the processing. *VoltDB*[3] is an open-source shared-nothing distributed RDBMS that partitions tables to fit in the distributed memory of multiple servers, eliminating the need to wait for the disk. *ScaleBase*[4] uses unmodified single-node MySQL databases and implements a control layer to shard tables over them while providing partial SQL support for querying [Cattell 2011].

2.1.2. Distributed File System (DFS). A DFS [Silberschatz et al. 2008] is a file system, where files are stored in a distributed manner across several machines and are accessed using a client server architecture via a network protocol. This allows storing all kinds of data, structured, semi-structured, and unstructured. The main goal of DFSs is to provide transparency by hiding the underlying mechanisms from users, which comes in many forms. *Location Transparency*, wherein, the name of a file is not related to its physical location. *Concurrency Transparency*, where each user sees the same state of the file. *Failure Transparency*, wherein, all users see the same state after a server failure. *Scalability Transparency*, where the DFS scales over heterogeneous hardware. *Replication Transparency*, where data is replicated for fault tolerance, without user intervention, in a way that minimizes the write cost and achieves reliability and availability.

Google File System (GFS) [Ghemawat et al. 2003] is a proprietary scalable DFS, designed to meet Google's rapidly growing data processing needs. The GFS divides files into 64MB replicated chunks distributed on a cluster of one master and several workers. The GFS periodically balances the data by replacing replicas to under-utilized servers. The master maintains the metadata, while the workers store the data chunks. The single master presents a single point of failure, overcome by periodically taking snapshots of it. In GFS, users must lease and write on the data primary copy, which is then propagated to the other replicas. The GFS works well with data intensive applications, where data is appended and not overwritten.

Hadoop Distributed File System (HDFS) [Shvachko et al. 2010] is an open source DFS inspired by GFS and designed to run Hadoop's batch jobs [White 2009] in massive parallelism (Figure 2). Unlike GFS, HDFS supports variable block size (64MB, 128MB, 256MB, so on) that are replicated on the slaves (DataNodes). HDFS does not implement leases and users can choose which data replica is to be written. HDFS

[1]http://dev.mysql.com/downloads/cluster/.
[2]http://www.scaledb.com/.
[3]http://voltdb.com/.
[4]https://www.scalebase.com.

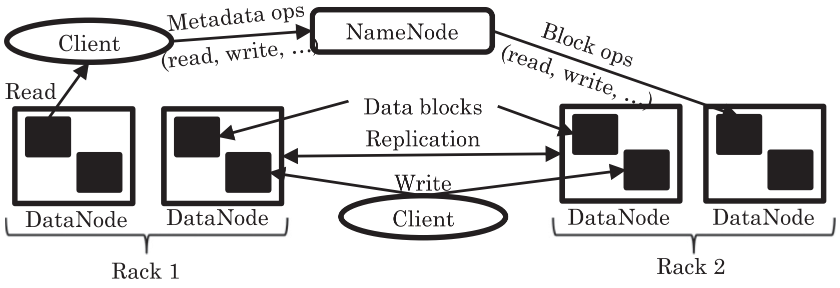

Fig. 2. HDFS architecture.

does data-balancing during writes and not periodically like GFS. Same as GFS, HDFS relies on accessing the metadata stored on the master node (NameNode). Hence, the availability of the entire HDFS is inhibited by the availability of this NameNode. To prevent single point failure, a secondary NameNode is introduced, which periodically checkpoints the primary NameNode and replaces it in case of failure.

Cassandra File System (CFS) [Lakshman and Malik 2010] is a Hadoop-compatible File System, designed to overcome some of the processing overheads of HDFS. Unlike GFS and HDFS, it uses decentralized deployment with multiple masters, avoiding a single point of failure. It also provides cross-data center replication for better failure recovery and availability.

On any of these DFSs, users can have their data stored in various formats. *JSON*[5] is a lightweight data-interchange format that is easy to read and write by both human and machine. *SequenceFile*[6] is a flat file consisting of binary key/value pairs, which can be compressed at the value or the key/value level. *CFile* [Lin et al. 2011] is a columnar storage, where data are partitioned in sorted vertical groups. *RCFile* [He et al. 2011] partitions the data horizontally, then vertically, where columns belonging to the same row are located on the same node. *CIF* [Floratou et al. 2011] is a binary columnar storage that partitions the data into horizontal splits, then columns of each split are stored in individual files. *CIF* uses a lazy approach to read the needed columns, leading to it outperforming *RCFile* [Floratou et al. 2011]. *ORCFile*[7] does horizontal followed by vertical splitting, then applies columnar compression and indexing within the row groups. *Parquet*[8] is a columnar storage that supports nested structures, per-column encoding, and have a high write performance by storing metadata at the end of the file.

2.1.3. Not-only Structured Query Language Systems (NoSQL). According to the CAP theorem [Brewer 2012], distributed systems cannot have all three: Consistency, Availability, and Partitioning tolerance; there will be always a tradeoff between them. NoSQL databases [Pokorny 2011] sacrifice the consistency, to have high availability and scalability. Instead of supporting the ACID model, NoSQL databases support the BASE model with Basically Available, Soft state and Eventual consistent. Along with supporting structured data, NoSQL databases also support semi-structured and unstructured data. NoSQL databases can be classified into *Key / Value*, *Column*, *Document* and *Graph* stores according to their data model.

—*Key / Value Database* is the most popular and simplest form of storage in NoSQL databases, where data is stored as key/value pairs. Most key/value databases support

[5]http://json.org/.
[6]http://wiki.apache.org/hadoop/SequenceFile.
[7]http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.0.2/ds_Hive/orcfile.html.
[8]http://parquet.incubator.apache.org/.

insert, delete, and update operations with a customizable key format. The value is opaque to these datastores, thus they only support querying and indexing through the keys and not the values (data). A Key/Value datastore is useful for storing multiple versions of data and is highly scalable owing to key distribution. Examples of this category are, *Amazon Dynamo DB* [DeCandia et al. 2007], *Voldemort* [Sumbaly et al. 2012], *Redis* [Carlson 2013], *Riak,*[9] and *MemcacheDB.*[10]

Voldemort and *Riak* use Multi Version Concurrency Control (MVCC), allowing multiple users to access the same data concurrently, while others use a locking mechanism. All key/value databases provide asynchronous updates and guarantee reading the latest version, except *Amazon Dynamo DB* which uses synchronous updates across multiple datacenters for high availability. Key/Value databases store data either in RAM or disk, except *Redis*, which stores data in RAM and provides disk as a backup. *Riak* and *Redis* implement MapReduce in their architecture while *Amazon Dynamo DB* supports MapReduce with the help of the Amazon EMR Service[11] and *Voldemort* uses Hadoop to run MapReduce jobs.

—*Column-Oriented Database* [Abadi et al. 2009] is a schema-oriented database, designed to store data as columns rather than rows. In these datastores, each row has a primary key and composed of a variable number of column families, which in turn are composed of a variable number of key/value pairs (columns). It is widely adopted by data warehouses, and ad hoc OLAP (Online Analytical Processing) query systems, where data is aggregated. When compared to other NoSQL databases, column-oriented databases have a high locality reference which minimizes disk access, improves the overall performance and reduces the storage requirements using compression techniques like LZW. Some of the popular column-oriented databases are: *HBase* [George 2011], *BigTable* [Chang et al. 2008], *Cassandra* [Lakshman and Malik 2010] and *Platform for Nimble Universal Table Storage (PNUTS)* [Cooper et al. 2008].

BigTable is a proprietary data storage built on *GFS,* whereas *HBase* works on *HDFS*, *Cassandra* works on *CFS* and *PNUTS* works on any DFS. *HBase* and *Cassandra* can create a column family, which groups multiple columns together, and stores them continuously on the disk. *BigTable* represents the data using three fields (Key, value, and timestamp), also known as *three-dimensional data storage*. On the other hand, *HBase*, *PNUTS,* and *Cassandra* support simple key/value storage of data.

All four databases partition data across the cluster. In *Google BigTable* and *PNUTS*, records are partitioned using hashing into units called Tablets. *Cassandra* and *PNUTS* support automatic partitioning using hashing/sorting mechanisms. All four databases provide asynchronous data replication on updates. Other than *Cassandra*, they provide locking mechanism over data. *Cassandra* and *PNUTS* support variable column length, hence they have a more flexible data model than *HBase* and *Google BigTable*. Only *PNUTS* provides both eventual and timeline consistency models, the others only provide eventual consistency.

—*Document-Oriented Database* is a schema less, more flexible Key/Value store, where the value can be a document with complex data structures like JSON. The documents' contents are not opaque to the system and can be indexed and queried. *MongoDB* [Chodorow 2013], *CouchDB* [Anderson et al. 2010] and *SimpleDB* [Habeeb 2010] are some of the popular document-oriented databases.

MongoDB and *CouchDB* are open-source solutions under the Apache license whereas, *SimpleDB* is an Amazon proprietary cloud service. *SimpleDB* is a simple document store with no support to nested documents. *CouchDB* and *MongoDB*

[9]http://basho.com/riak/.
[10]http://memcachedb.org.
[11]http://aws.amazon.com/elasticmapreduce/.

are high-performance document-oriented databases that support richer data models. *CouchDB* and *MongoDB* provide automatic sharding across the cluster, while *SimpleDB* needs manual intervention. *CouchDB* uses MVCC while *MongoDB* supports document-level atomic operations [Cattell 2011]. All these databases support asynchronous writing to the replicas and eventual consistency.

—*Graph Database* originated from graph theory and stores data in graph structures presenting the relationships between the data items. No indices are needed in graph databases since every data point is directly connected by an edge to the related data points. Each data point (graph node) contains details of the data. For associative datasets, a graph database is often faster and more efficient than a relational database. Like document-oriented databases, graph databases do not require join operations and can scale up for large datasets. Graph databases are powerful for graph-oriented queries such as finding the shortest path between two nodes. They are highly scalable and provide high availability through date replication. *Neo4j* [Partner 2013], *OrientDB* [Tesoriero 2013], and *Infinite Graph* [Objectivity, Inc. 2012] are some popular examples.

These three graph databases are labeled, directed, multi-property, and provide ACID consistency. Both vertices and edges can have multiple key/value properties associated. Unlike column-oriented databases, these three databases use horizontal partitioning, where rows are held separately, rather than being split into columns. Each row-group partition forms a shard that is located on a separate machine. *Infinite Graph* uses *ObjectivityDB*, which was the first DBMS to store a petabyte of objects. *Neo4j* uses native graph storage, which is optimized and designed for storing and managing graph data. *OrientDB* can use any filesystem. *Neo4j* uses a master-slave architecture with cache sharding, where the same master serves all requests of a particular user, to make use of the cached data. *OrientDB* uses a multi-master replication and sharding, where any node can serve arriving requests, to have better utilization over the cluster. *Infinite Graph* supports parallel and loosely synchronized batch loader (aka eventual consistency), while *OrientDB* supports MVCC.

2.2. Big Data Processing

Big Data Analytics inherited a number of centralized data mining solutions from the pre-Big Data era, where data could fit in a single node memory. These centralized solutions like *R*[12] and *Weka*[13] provide a huge set of algorithms but they do not scale to meet the Big Data requirements. To solve the scalability problem, *Google* introduced *MapReduce* [Dean and Ghemawat 2008] to manage and process large multi-structured datasets. Following that, *Yahoo!* developed *Hadoop* [White 2009], an open-source implementation of MapReduce and later incubated by *Apache,*[14] allowing everyone to benefit from it. *Microsoft* developed similar solutions, namely *Cosmos* [Zhou et al. 2012] and *Dryad* [Isard et al. 2007] for their internal usage.

As explained later in this section, not all Big Data Analytics processes can be efficiently executed on MapReduce. Beside the *Batch processing* approach that MapReduce was designed for, other processing approaches have surfaced to deal with the different Big Data Analytics processing needs. These approaches can be categorized based on their intended application type. These categories are: *Batch* processing, *Interactive* processing, *Iterative* processing, *Incremental / Stream* processing, *Approximate* processing, and *In-Database* processing.

[12]http://www.r-project.org/.
[13]http://www.cs.waikato.ac.nz/ml/weka/.
[14]http://hadoop.apache.org/.

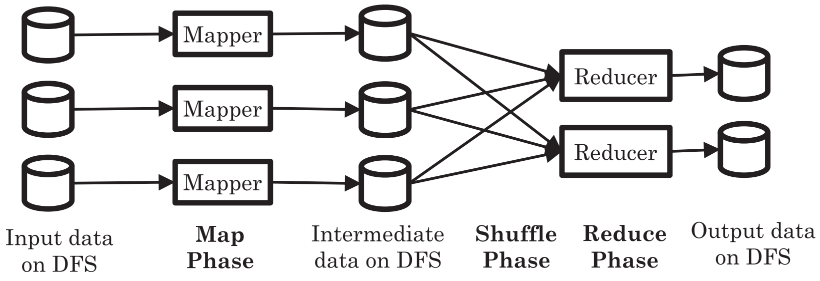

Fig. 3. MapReduce dataflow.

2.2.1. Batch Processing. Batch processing is designed to execute a series of jobs without manual intervention. This makes it perfect for scenarios where a program needs to have a single run on a huge dataset. MapReduce, and its open source implementation *Hadoop*, are popular batch-processing frameworks because of their scalability, fault-tolerance, ease-to-program, and flexibility.

MapReduce, as presented in Figure 3, consists of *Map*, *Shuffle*, and *Reduce* phases, which are executed sequentially, utilizing all nodes in the cluster. In the *Map* phase, the programmer-provided *Map* function (*Mapper*) processes the input data and outputs intermediate data in the form of <*key, value*> tuples which get stored on disk. The S*huffle* phase then groups values to the same key together and sends them to the reduce nodes over the network. Finally, the programmer-provided *Reduce* function (*Reducer*) reads the intermediate data from disk, processes it, and generates the final output. The *Map/Reduce* functions are executed in parallel over a set of distributed data files in a Single Program Multiple Data (SPMD) paradigm.

With the growing popularity of MapReduce, centralized data mining solutions started adding support for MapReduce to enhance their scalability. *DistributedWekaBase,*[15] *DistributedWekaHadoop,*[16] and *DistributedWekaSpark*[17] [Koliopoulos et al. 2015] packages extend *Weka* to access *HDFS, Hadoop,* and *Spark* [Zaharia et al. 2010b], respectively. *RHadoop*[18] allows running *R* code on *Hadoop* and access to *HDFS*. This extension, however, leaves it to the users to write the MapReduce programs themselves, which requires extra expertise. *Radoop* [Prekopcsak et al. 2011] extends *RapidMiner*[19] to run on *Hive* [Thusoo et al. 2009] and *Mahout.*[20] *IBM Analytic Server*[21] extends *SPSS Modeler*[22] to run scalable distributed analytics on *Hadoop*.

One of the main MapReduce issues is the high IO overhead caused by writing intermediate and inter-job data to disk and having to read them in again. Solutions like *Hadoop Online Prototype (HOP)* [Condie et al. 2010] and *Walmart Labs Muppet* [Lam et al. 2012] aim at improving MapReduce efficiency, while maintaining its desirable properties. The *HOP* runs unmodified *Hadoop* jobs and uses memory instead of disk to pipeline tasks' output. *Muppet*, on the other hand, modifies MapReduce by replacing *reducers* with *updaters. Updaters* work in the same way as *reducers*. Except

[15]http://weka.sourceforge.net/packageMetaData/distributedWekaBase/index.html.
[16]http://weka.sourceforge.net/packageMetaData/distributedWekaHadoop/index.html.
[17]http://weka.sourceforge.net/packageMetaData/distributedWekaSpark/index.html.
[18]https://github.com/RevolutionAnalytics/RHadoop/wiki.
[19]https://rapidminer.com/.
[20]https://mahout.apache.org/.
[21]http://www-03.ibm.com/software/products/en/spss-analytic-server/.
[22]http://www-01.ibm.com/software/analytics/spss/products/modeler/.

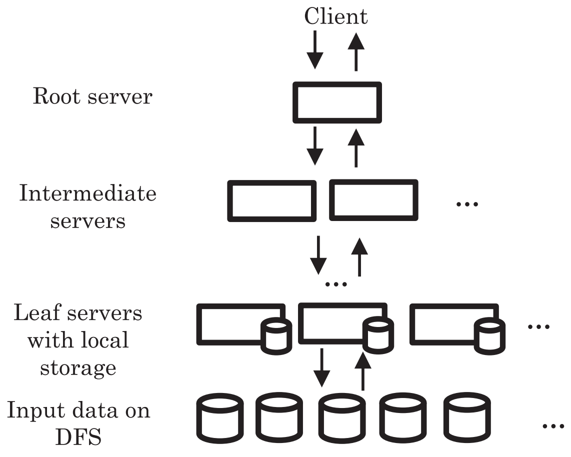

Fig. 4. *Dremel* query execution flow.

they use a live in-memory data structure (a *slate*) to summarize all data seen so far. A *slate* is defined for each key and is continuously updated as more data arrives.

2.2.2. Interactive Processing. The frequent writing to disk and the extensive communication between nodes in the MapReduce shuffle phase to support fault-tolerance, hinders efficient support for interactive applications. Some solutions are proposed to support interactive analytics.

Google Dremel [Melnik et al. 2010] (aka *Google BigQuery*[23]) leverages massive parallel processing, nested data modelling and columnar storage to improve retrieval efficiency for interactive analytics scenarios. *Dremel* is a *Google* proprietary solution that executes its query in parallel without being translated to a sequence of MapReduce jobs. *Dremel* uses multi-level tree processing (Figure 4). Leaf servers only scan the needed columns in parallel. Intermediate servers carry out parallel aggregation on the scanned data. Finally, the root server aggregates the intermediate results. However, with columnar storage, the number of columns accessed affects performance. *Google* has introduced *PowerDrill* [Hall et al. 2012] to have data in memory for faster computations. However, this makes it constrained by the available memory.

Apache Drill[24] and *Cloudera Impala*[25] offer open source implementations of *Google Dremel*. Moreover, they support querying and joining data from different sources, which is not supported in *Dremel*. Architecture-wise, *Drill* does not implement a dedicated root server like the others. Instead, any *Drill* node (aka *Drillbit*) can accept queries and become the root server (aka *driving Drillbit*), which eliminates the issue of having a single point of failure. *Drill* only has leaf servers, which exchange data among themselves to carry out the aggregations, reducing data movement. *Drill* supports dynamic schema discovery, where users can define the schema (column name, data type, length) in the query, or let *Drill* discover the schema from the metadata for self-describing data formats. On the downside, *Drill* adopts an optimistic execution model and does not persist in intermediate data, making it fault-intolerant.

Apache Tez,[26] introduced in *Hadoop 2.0,* aims at generalizing the MapReduce paradigm to support interactive queries. *Tez* groups all MapReduce jobs of

[23]https://cloud.google.com/bigquery/.
[24]http://drill.apache.org/.
[25]http://impala.io/.
[26]http://tez.apache.org/.

an analytics query into a single *Tez* job, eliminating the overhead of launching multiple jobs. The *Tez* job still consists of *mappers* and *reducers*; however, *Tez* merges some of the *mappers* and *reducers* together to minimize the overhead for materializing intermediate outputs to the DFS and to provide better data locality.

2.2.3. Iterative Processing. Iterative computation arises naturally in Data Analytics. Machine learning operations (e.g., in stochastic gradient descent, K-means clustering, etc.), require several passes over the training data for the algorithm to converge. Designed for long-running processes, MapReduce does not natively support iterative computation. For that, users must write a sequence of MapReduce jobs and coordinate their execution. Even with that, MapReduce still lacks a mechanism for reusing output results without rereading them from disk, which causes a big hit to performance.

HaLoop [Bu et al. 2010] and *iMapReduce* [Zhang et al. 2011a] modify MapReduce and its API to add iteration support. *HaLoop* modifies the MapReduce task scheduler to schedule the same job tasks on the same nodes in each iteration, thus enabling the reuse of data across iterations. *iMapReduce* creates the *mappers* and *reducers* only once at the beginning of the job, and uses them in subsequent iterations. This reduces the overhead of creating new MapReduce jobs for each iteration. *iMapReduce* allows asynchronous execution, where output is streamed to the next task so that it can start without waiting for previous tasks to finish. *iMapReduce* checkpoints the *reducers'* output every few iterations for fault tolerance and only persists the final output. *PrIter* [Zhang et al. 2011b] adds prioritized iteration to *iMapReduce* for a faster convergence.

The *Main Memory MapReduce (M3R)* [Shinnar et al. 2012] and *Twister* [Ekanayake et al. 2010] work on speeding up the execution of unmodified *Hadoop* iterative jobs. *M3R* uses in-memory pipelining between *mappers* and *reducers* to achieve the iterative job performance of *HaLoop*, without the burden of using new APIs. *Twister* relies on a message broker to which nodes can publish/subscribe to communicate and transfer their data, where inter-iteration data is cached in memory. Both *M3R* and *Twister* are constrained by the available memory and do not support fault tolerance within an iteration, making them useful only for short jobs running on highly reliable clusters with large memory.

Apache Mahout[27] is a scalable distributed machine-learning library, where the machine-learning algorithms are implemented as a sequence of MapReduce jobs with an outside driver program to control loop execution. *Mahout* adds analytics capabilities to *Hadoop* but still suffers from the MapReduce high IO overhead for re-reading inter-job data from disk. *Hivemall* [Yui and Kojima 2013] adds machine-learning capabilities to *Hive* [Thusoo et al. 2009]. However, *Hivemall* queries are still translated to MapReduce jobs with inter-job data written to disk. To overcome the MapReduce high IO overhead, *Hivemall* amplifies the training data (replicates each row several times) and then randomly shuffles this amplified dataset to emulate the iteration effects without having several MapReduce steps.

Apache Spark[28] [Zaharia et al. 2010b] is now the main rival to *Hadoop*. *Spark* does not run *Hadoop* jobs, however, they can co-exist on the same cluster using *Apache Mesos* [Hindman et al. 2011]. A *Spark* job runs in parallel with one *reducer,* which can be a bottleneck. *Spark* uses read-only Resilient Distributed Datasets (RDDs) to provide in-memory support for iterative jobs. RDDs can also reconstruct themselves in case of failure to provide fault-tolerance without disk checkpointing. *Spark*, however, is restricted to the size of available memory and has to reconstruct the RDDs from disk

[27]https://mahout.apache.org/.
[28]https://spark.apache.org/.

if it runs out of memory. *MLBase*[29] [Talwalkara et al. 2012] uses the *MLlib*[30] [Sparks et al. 2013] library to provide distributed machine learning at scale on *Spark*. *Cloudera Oryx*[31] forks from *Apache Mahout* to run on *Spark*. *Oxdata H2O*[32] provides in-memory machine learning and predictive analytics on Big Data using *Spark*. *Deeplearning4j*[33] provides deep learning algorithms implementation on *Hadoop* and *Spark*.

2.2.4. Incremental Processing. Unlike previous categories which analyze data-at-rest, incremental solutions analyze data-in-motion. Data that is usually outdated quickly, thus fast reactions are required. Online algorithms are used to process real-time data streams, without having the entire input available, and before data is saved to disk. This makes it ideal for continuous and incremental analytics, as with analyzing monitoring logs, sensor network feeds, and high-volume news feeds like twitter. Incremental processing can be in the form of *stream processing* or *micro-batch processing*.

—*Stream processing* provides low-latency as data is analyzed as soon as it arrives. However, it is technically challenging as it requires devising online algorithms to analyze partial data and not to wait for the complete dataset.

Apache Storm[34] and *Apache S4*[35] [Neumeyer et al. 2010] are distributed streaming solutions that can be used with any programming language and can scale to massive numbers of messages per node per second. *S4* asks users to develop their programs for a single key and not for the whole stream like *Storm*, which simplifies the programs' logic and makes their development easier. However, *S4* uses a push model, which can cause data to drop if the receiver's buffer is full. *Storm*, on the contrary, uses a pull model, where the receivers pull the data when they can process it. *Apache Samza,*[36] coming as part of *Hadoop2.0,* uses *Apache Kafka*[37] publish/subscribe messaging system to guarantee that data is processed in the order it arrives, and that no data is ever lost.

Microsoft Trill [Chandramouli et al. 2014] is an in-house stream processing solution for temporal data, which consists of a payload and a validity interval defining the duration this payload is contributing to the output. Microsoft also offers *Stat!* [Barnett et al. 2013] designed for progressive computations using the unmodified *Microsoft StreamInsight* [Chandramouli et al. 2012] temporal streaming engine. Another Microsoft solution is *Naiad*[38] [Murray et al. 2013] that allows iterations in streams to handle scenarios with changing input. Google has the *MillWheel* [Akidau et al. 2013] in-house system that allows users to create stream graphs and define the application code for each graph node, while it handles the continuous data flow, data persistence, and failure recovery.

IBM InfoSphere Streams[39] is a component of the *IBM* analytics solution. It supports analyzing data continuously at high rates using real-time machine learning. *Streams* supports run-time modifications, where input streams, output streams and operations can be added or removed dynamically without restarting the system. It also allows embedding user defined Java and C++ analytics routines.

[29]http://www.mlbase.org/.
[30]https://spark.apache.org/mllib/.
[31]https://github.com/cloudera/oryx.
[32]http://0xdata.com/h2o-2/.
[33]http://deeplearning4j.org/.
[34]https://storm.incubator.apache.org/.
[35]http://incubator.apache.org/s4/.
[36]http://samza.apache.org/.
[37]http://kafka.apache.org/.
[38]http://research.microsoft.com/en-us/projects/naiad/.
[39]http://www-03.ibm.com/software/products/en/infosphere-streams/.

—*Micro-batch processing* presents a middle-ground between streams and batch processing. It has higher latency than streams as it buffers the input, and only processes it when the buffer is full. However, micro-batching is less technically challenging as it allows the use of existing batch algorithms.

Using this approach, *Yahoo! Nova* [Olston et al. 2011] allows users to create workflows of *Pig* programs to process continually arriving data. *Spark Streaming*[40] extends *Spark* to allow joining stream data with historical data using the same *Spark* code written for batch processing. The *streams library* plugin [Bockermann and Blom 2012] adds online processing support to *RapidMiner*. It provides generic streaming wrappers of the *RapidMiner* operations to make them run on partially available data and feed them the rest as it arrives.

2.2.5. Approximate Processing. Designing the analytics process involves many trial and error attempts till the best operations are found. Using traditional MapReduce means analyzing the whole dataset, which is very time consuming and impractical. Quick retrieval of approximate results from a small representative sample should be enough to draw a conclusion.

The *Early Accurate Result Library (EARL)* [Laptev et al. 2012] extends *Hadoop* to allow early termination and incremental computation, along with providing an on-line indicator to estimate the achieved accuracy so far. *EARL* provides early approximate results by iterating over data samples and aggregating the results until an acceptable accuracy is reached. *EARL* modifies *Hadoop* by (i) keeping the *mappers* alive even after they are done, to be used in the next iteration, (ii) pipelining the *mappers* output directly to the *reducers*, and (iii) allowing *reducers* to start before the *mappers* are finished to support incremental computation.

BlinkDB [Agrawal et al. 2012] extends *Hive* to provide fast approximate results with statistical error guarantees. *BlinkDB* uses a dynamic sample selection strategy to select an appropriately sized sample based on the desired query accuracy or response time. *BlinkDB* maintains a set of pre-computed and carefully chosen data samples, so that when a query arrives, it can be directly executed on the appropriate sample. Samples in *BlinkDB* are chosen using an optimization formula that considers the data distribution, past queries, storage constraints, and several other system-related factors.

2.2.6. In-Database Processing. MapReduce-based solutions are considered by some to be better suited for Big Data Analytics because of their scalability and flexibility. Others support parallel databases, which have been extensively studied for decades, and enhanced with a lot of optimization techniques that were refined over time. However, parallel databases require a well-defined schema, which is unsuitable for multi-structured data. They typically run on high-end servers, which makes them an expensive option.

In-Database analytics allows users to run machine learning on data without moving it out of the database. This solves the data movement issue and allows using database techniques like clustered indices to outperform *Hadoop*. *Microsoft SQL Server Analysis Services (SSAS)*[41] allow users to run data mining operations on their data within the MS-SQL database. Users can train models, save and use them later for predicting unlabeled data. There is also *MADLib*[42] [Hellerstein et al. 2012], an open source library for scalable distributed analytics that runs within the database engine.

Hybrid solutions attempt to have the high performance of parallel databases by harvesting all the benefits of database query optimization, while yielding the same fault tolerance and scalability of MapReduce. *HadoopDB* [Abouzied et al. 2010] installs

[40]https://spark.apache.org/streaming/.
[41]http://www.microsoft.com/en-us/server-cloud/solutions/business-intelligence/analysis.aspx.
[42]http://madlib.net.

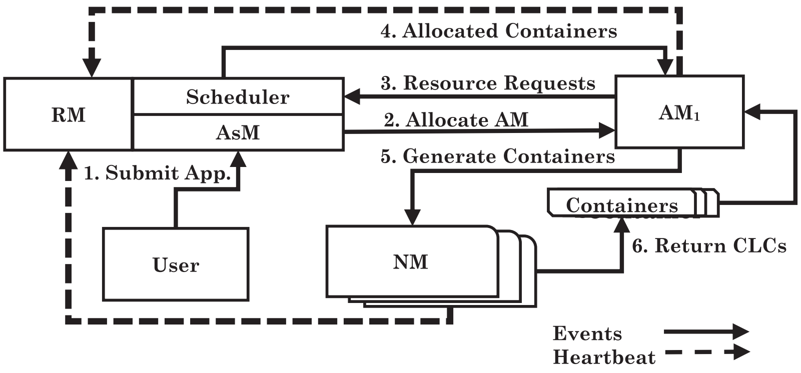

Fig. 5. YARN operation.

a database system (e.g., PostgreSQL) on each *Hadoop* node, where *Hadoop* takes care of the task coordination and communication. *HadoopDB* accepts *HiveQL* queries as input, uses *Hive* to transform them to MapReduce jobs, then assigns as much work as possible (like joins, conditional scans, etc.) to the local database for query processing on each node. Results retuned from local database instances are then aggregated and further processed using *Hive*.

2.3. Analytics Orchestration

Big Data analytic solutions require the orchestration of complex analytic jobs and workflows to achieve the user's goals. This orchestration should intelligently automate and coordinate job scheduling and resource provisioning to satisfy user requirements.

2.3.1. Scheduling. Scheduling is the process of allocating jobs to the available resources, while maximizing resource utilization and data locality. The Scheduler receives the job's computational requirements and assigns resources to it from the available resources. Resources include memory, CPU, network and disk, bounded to one physical node.

—*Resource Utilization.* Hadoop 1.0 has a number of shortcomings when it comes to resource management. One of the main shortcomings is having a fixed number of map and reduce slots per node. This causes the cluster to be under-utilized as it forces new map tasks to wait when all map slots are taken, even if the node has idle resources. The same applies for the reduce tasks and slots. A second shortcoming is the restriction of having a single job tracker to handle only up to 4000 nodes limiting scalability.

Apache Hadoop YARN (aka MapReduce2 or Hadoop 2.0) [Vavilapalli et al. 2013] solves the above issues. YARN does not use the slot configuration paradigm. Each node's resources (e.g., CPU cores and memory) are allocated to the applications when requested. It separates the cluster resource management from the application management, which increases the efficiency and resource utilization. YARN also supports MapReduce and non-MapReduce applications on the same cluster.

YARN involves three main components, shown in Figure 5: (i) *Resource Manager (RM)* is installed per cluster to manage the available resources. It has an *Applications Manager (AsM)* that accepts the submitted applications and negotiates with the Scheduler to obtain the necessary resources for executing an application. The RM also has the *Scheduler,* which is responsible for allocating *resource containers* to applications based on their resource requests. A resource container encapsulates computational resource elements like memory, CPU, network and disk into one entity. A container is bound to one node but a node can hold multiple containers. The scheduler ensures fairness and data locality, but not fault tolerance. (ii) There is an *Application Master (AM)* per job that manages its lifecycle, computes its required resources, and communicates these requirements to the RM. (iii) A *Node Manager (NM)* runs on every worker machine to launch containers for applications once allocated by the RM. Each container is wrapped by a Container Launch Context (CLC), defining environment variables, dependencies, security tokens, etc.

Another example is *IReS* [Doka et al. 2015], a meta-scheduler for running workflows over multi-engine environments. *IReS* automatically matches distinct workflow parts to the right engine(s) according to multiple criteria (like cost and performance), deploy and run them without manual intervention. This schedule is ideal for scenarios where no single engine/store is suitable for all required computations/data.

Adoop [Hamdaqa et al. 2015], on the other hand, is a Hadoop history-based scheduler for volatile, non-dedicated, ad-hoc environments where underutilized computing resources in the existing IT infrastructure are used. *Adoop* uses the nodes' availability history and current utilization to make scheduling decisions and dynamically re-adapts task assignments according to the nodes' availability. It also replicates tasks to provide a guaranteed minimum availability level for each task.

—*Data Locality.* A design goal of many systems [Vavilapalli et al. 2013, Deelman et al. 2005, Ranganathan and Foster 2002]. It refers to the degree to which data and processing are co-located on the same physical node. For data-intensive jobs, the network has been shown to be a potential bottleneck [Zaharia et al. 2010a; Dean and Ghemawat 2008]. Thus, data locality has a significant impact on the job performance since a higher data locality means less data transfers over the network. However, data locality comes with its own problems. For example, nodes storing input data tend to become hotspots while other nodes are under-utilized [Elshater et al. 2015].

Pegasus [Deelman et al. 2005] is a workflow management system. It uses a Replica Location Service (RLS) to achieve data locality. The Pegasus scheduling algorithms queries the RLS to retrieve data replica locations for the incoming tasks. RLS stores the mappings of the files logical names to the physical locations of their replicas.

Ranganathan and Foster [2002] find it inefficient for scheduling algorithms to only target processor utilization with no regard to data-retrieval cost. Targeting data-intensive applications, they proposed decoupled scheduling that separates the job-scheduling policy from the replication policy. The solution consists of three schedulers: an External Scheduler (ES) that assigns nodes to jobs, a Local Scheduler (LS) that prioritizes jobs on the local node, and a Dataset Scheduler (DS) that calculates data popularity and handles data replication.

Zaharia et al. [2010a] noted that fair scheduling is conflicting with data locality. Fair scheduling compromises data locality as jobs might be scheduled away from their input data. They propose the delay scheduler which relaxes the fairness constraints for a better data locality opportunity. The delay scheduler can postpone a job allocation for a certain small duration in the hope that a container becomes available on the node holding the required data. The delay scheduler is considered the only data-locality-aware scheduler among the current Hadoop and YARN schedulers [Vavilapalli et al. 2013]. The delay scheduler goes through three phases: (i) Node Locality, where the scheduler tries to schedule the incoming task on a node that stores the input data; (ii) Rack Locality, used if node locality is not possible, where the scheduler tries to schedule the incoming task to a node on the same rack where the input data exists; and (iii) Off-Switch Locality, which is the worst case, where the scheduler assigns the incoming task to an off-switch node, located on a different rack to avoid task starvation.

Guo et al. [2012] propose a mathematical model for the MapReduce data locality problem to find the optimal schedule for maximizing data locality. It shows that scheduling multiple MapReduce tasks together gives a better performance than the delay scheduling approach, where the scheduling is done on a task-by-task basis. One of the reasons behind this better performance is that it considers the impact of each task assignment on the other tasks, while the delay scheduler does not.

Pixida [Kloudas et al. 2015] is a *Spark* scheduler that works on minimizing the traffic while processing datasets that span multiple data centers. It models the scheduling goals as a graph partitioning problem. It then searches for opportunities to avoid data movement between the data centers by allocating tasks to where data exist.

2.3.2. Provisioning. Provisioning aims at allocating resources and data to jobs while minimizing the job execution time and monetary cost. The provisioner receives the user's Service Level Objectives (SLOs) and budget. Then it tries to find the best set of resources and data distribution to meet the user's SLOs while still being within the user's budget. Provisioning can involve resource provisioning, to allocate resources to jobs, or data provisioning, to allocate data to jobs.

—*Resource Provisioning.* Given the large set of the different resource types provided by cloud providers, it becomes challenging to find the best combination of resources to perform the requested job under the user's SLOs and budget. Moreover, the provisioning systems need to manage the tradeoffs between different user objectives (e.g., minimizing monetary cost versus minimizing running time), which is not always a straightforward task.

The *Resource Set (RS) Maximizer* [Kambatla et al. 2009] is designed to provision MapReduce jobs to minimize the monetary cost, while achieving the best possible performance. Given that the default Hadoop configuration is not fit for all jobs, this algorithm works on choosing the best Hadoop configuration (e.g., the number of mappers and reducers) based on the job and the provisioning conditions. RS Maximizer stores a set of optimum configurations for the different job types and chooses one of them for new jobs based on the job type. Building this configurations database to cover a wide variety of jobs and configurations (more than 150 configuration values) is not always feasible. Also, RS Maximizer assumes that there is a fixed number of cloud machines in the resource pool, preventing it from automatically scaling up as new machines are added.

Conductor [Wieder et al. 2012] is a system that orchestrates execution of MapReduce jobs on the cloud by choosing the most suitable cloud services according to the user-defined objectives (e.g., reducing the execution time or minimizing the costs). Conductor provides an abstraction layer to allow combining services (e.g., storage, computation) from different providers to meet the user goals in the best way possible. It supports integrating the user's local infrastructure with public clouds to provide hybrid deployments. Conductor also considers the dynamic pricing of the cloud services (e.g., Amazon EC2 spot instances[43] prices change every 10 minutes).

Purlieus [Palanisamy et al. 2011] works on minimizing the network distance between compute and storage nodes. First, Purlieus defines the proper set of physical machines that should store the input datasets based on the job type. For example, it distributes the data blocks across the network to utilize all the physical machines for input-heavy jobs, where the mappers generate small intermediate data. Afterwards, it attempts to deploy mappers close to the nodes storing input data blocks and reducers close to the mappers that generate the intermediate data.

Mian et al. [2013] formulate the provisioning problem and design a framework to predict the cost of executing data-intensive workloads given a set of configurations. A configuration defines the number of cloud machines and their specifications. This approach begins with exploring all possible configurations based on the predicted

[43]http://aws.amazon.com/ec2/purchasing-options/spot-instances/.

monetary cost of each configuration. The search space is described as a Directed Acyclic Graph (DAG) where each node in this graph represents a unique configuration and each edge represents a possible movement (e.g., add the cheapest VM, add the same VM, upgrade, and so on) from one configuration to another. Greedy search algorithm is then used to traverse this graph to find the optimal configuration.

Kllapi et al. [2011] study the tradeoff between the completion time and monetary cost for executing workflows. They propose a greedy provisioning algorithm to find the optimum assignment of cloud resources for executing a workflow while satisfying the user's budget.

Mao and Humphrey [2013] propose two auto-scaling algorithms to minimize the job turnaround time (the time elapsed from job submission to the job completion) within budget constraints for executing workflows using cloud resources. The *scheduling-first* algorithm assigns more budget to high-priority jobs in the workflow. Then, it determines the fastest execution plan and accordingly acquires the required cloud resources. On the other hand, the *scaling-first* algorithm first finds the type of cloud resources needed based on the budget constraint and then schedules the workflow jobs to them.

Mesos [Hindman et al. 2011] is a platform for sharing cluster nodes between multiple frameworks, such as *Hadoop* and *Spark*. *Mesos* introduces a distributed two-level scheduling mechanism where it decides how many resources to offer each framework and the frameworks can then use their own scheduling to decide which resources to accept and which computations to run on them.

Li et al. [2015] propose a latency-aware algorithm for running high-speed real-time data streams on *Hadoop*. The algorithm searches for the minimum number of nodes that maximizes throughput without violating the latency requirements.

—*Data Provisioning*. The Data Locality Scheduling techniques, discussed in the previous section, assume that the number of replicas per data block is fixed during execution. However, this can hurt performance and data locality, especially if some of the nodes in the cluster have more data than the others. The different data provisioning techniques below allow a variable number of replicas per data block and automatically replicating these blocks as the demand on them increases.

Scarlett [Ananthanarayanan et al. 2011] uses the Hadoop job history logs in a previous unit of time (e.g., days or weeks) to replicate offline the data blocks based on their observed access statistics. Scarlett computes the data replication factor constrained by a certain budget and uses a replica aging to delete old replicas.

Adaptive Data Replication for Efficient Cluster (DARE) [Abad et al. 2011] assigns an adaptive replication probability to data, based on the non-local data access requests it receives, ignoring network cost. DARE helps achieve better data locality by replicating data blocks to remote machines under a disk budget. DARE is scheduler independent and can work with any Hadoop scheduler (e.g., Delay or FIFO scheduler) to increase data locality. DARE is still an off-line system, but it works with smaller time units compared to Scarlett.

Cost-Effective Dynamic Replication Management (CDRM) [Wei et al. 2010] is a replication placement scheme for Hadoop that calculates the ideal number of replicas per data block to satisfy the availability requirements. However, CDRM does not support increasing the number of replicas dynamically during run-time.

2.4. Big Data Analytics Assistance

With the increasing sophistication of analytics processes and the ever-growing number of data mining algorithms and techniques, organizations find it hard to hire employees with the required experience in Big Data Analytics. According to the McKinsey report [Manyika et al. 2011], there is a widening talent gap in the workforce, wherein by 2018, the demand for skilled data scientists could be 50 to 60 percent more than the expected supply, and that is only in the United States.

Consumable Analytics provides a solution for narrowing the analytics talent gap. The idea is to increase the impact of the skills already existing in organizations by providing in-tool assistance to make analytics easier to build, manage, and execute [IBM 2012]. Assistance can be *Static*, where it always shows the same content no matter what dataset is being analyzed. Static assistance is useful in aiding users with configuring operations but not with selecting them. Novice analysts are typically overwhelmed with the large number of analytics techniques making them unable to confidently select the right technique and often have to resort to time-consuming trial and error [Serban et al. 2013].

Experts have years of hands-on knowledge on selecting techniques for different contexts. But even they are now finding it hard to keep up with the ever-increasing number of operations and they usually tend to use the methods that have proven to be successful in the past [Serban et al. 2013]. To support both novice and expert analysts in selecting the best-suited technique from this plethora of different techniques, several researchers have proposed *Intelligent Assistance*. Intelligent Assistance aids users in choosing and configuring the analytics operations based on the input dataset and the analysis goals. It covers *data preparation*, *selecting operations*, *analytics workflow generation,* and *fault detection and handling*.

2.4.1. Static Assistance. Static assistance provides the means for users to learn to use an analytics solution and its operations. Once users become familiar with the solution, however, static assistance is no longer needed and may even become an obstacle that slows down expert users. The types of static assistance include the following:

—*Tooltips.* A single sentence describing an operation's functionality. Sometimes, a tooltip also includes a description of the inputs and outputs. Tooltips usually appear as users hover over the operation's icon. They represent the simplest way of providing assistance. Their usefulness, however, is limited to new users. As users gain more experience, tooltips can become more annoying than useful.
—*Help Pages.* A reference manual that users can use to learn more about the operations. They are usually provided as web pages with text and images describing how to use the operations. Sometimes, they also include the algorithms, making them very useful when there is a need to extend or modify some operations.
—*Wizards.* A sequence of dialog boxes that guide users through a series of well-defined steps. Wizards are well suited for complex, repetitive, or unfamiliar operations. For example, *RapidMiner*[44] offers wizards for churn reduction and sentiment analysis. Wizards usually do not allow users to configure all of an operation's parameters in order to keep the interface simple for novice users. In some wizards, an "Advanced" option is provided to allow access to a larger set of parameters. Some expert users find wizards slowing them down by forcing them to go through unnecessary steps or posing limitations by not giving access to all parameters they need to configure.

2.4.2. Intelligent Assistance. Over the last several decades, the fields of Statistics and Machine Learning have contributed numerous algorithms for data mining. Users need to effectively utilize this available "arsenal" of algorithms to produce useful and meaningful results. Most existing analytics solutions only offer static assistance [Serban et al. 2013; Charest 2007]. Static assistance, being context and data independent, cannot help users to effectively select the appropriate analytics operations.

[44]http://rapidminer.com/.

Intelligent Assistance can help in this regard by providing context-aware data-aware assistance.

—*Data Preparation.* It is one of the most important phases in the analytics process, it affects all the steps that come after. It includes determining promising and irrelevant attributes, attributes that need to be split and those that need to be combined, along with handling any malicious data-like outliers, missing values, etc. Data preparation is challenging as it needs to deal with different data formats, granularities, and degree of completeness based on the method of acquiring the data. Data preparation is also domain specific, which makes it tricky, especially for analysts new to a business domain.

Ontologies can provide intelligent decision support mechanisms in data preparation, relieving some of the burden of users and resulting in a faster development of the analytics process. An ontology represents the concepts within a domain and specifies how the concepts are related using logical axioms [Gruber 1993].

Ontologies are typically expressed in the Web Ontology Language (OWL) or the newer version OWL2 [Bechhofer et al. 2004] and edited in editors such as Protégé[45]. Using the explicit knowledge represented in an ontology, new implicit knowledge can be inferred using logical reasoners or inference engines [Wang et al. 2004] such as Fact++[46] [Tsarkov and Horrocks 2006] and Pallet[47] [Sirin et al. 2007]. Readers interested in reasoners can check Abburu [2012] and Dentler et al. [2011] for their extensive surveys on the different ontology reasoners.

Ontologies are being used in BI solutions to facilitate data integration [Cui et al. 2007; Martin et al. 2011] and for the Extract-Transform-Load (ETL) operations in data warehousing [Sciarrone et al. 2009]. For data preparation, ontologies can be used to represent the different domain concepts. These concepts can be common concepts such as temporal concepts, geographic concepts, etc. or more domain-specific concepts for a targeted domain like a credit card ontology [Kotsiantis et al. 2009].

Concepts can represent *attributes* describing entities such as “identifier” and “caption” [Rais-Ghasem et al. 2013]. Concepts can represent *metrics,* which are quantifiable indicators to measure the performance along these entities, for example, “cost” or “revenue” [Rais-Ghasem et al. 2013]. Metrics can have a default favorable trend such as down for cost or up for revenue and they can have common value ranges, for example, setting age value ranges to [0-7], [12], [13-19] [Rais-Ghasem et al. 2013], which can be used to identify malicious data. New concepts like “kid”, “Teen”, “Adult” can be then instantiated using an associated rule language like *Semantic Web Rule Language (SWRL*[48]*)* and a rule engine.

Data labels (column names) and values are created using different naming and formatting conventions making ontologies by themselves insufficient for making sense of a dataset. Natural Language Processing (NLP) engines are thus needed to help map the different naming and formatting conventions to the ontology concepts [Rais-Ghasem et al. 2013]. *IBM Watson Analytics* [Rais-Ghasem et al. 2013] is one of the systems that use both ontologies and NLP to semantically annotate the data automatically for a better understanding.

The *MiningMart* project [Morik and Scholz 2004] uses a different approach in helping users with data preparation. *MiningMart* stores the best cases of data preparation workflows that were designed by experts. Users can then choose one of these cases as a starting point and adapt it to their problem.

[45]http://protege.stanford.edu/.
[46]https://code.google.com/p/factplusplus/.
[47]http://clarkparsia.com/pellet.
[48]http://www.w3.org/2001/sw.

—*Selecting Operations*. Selection is implemented by having a set of preconditions for each operator and using simple matching with the input data to recommend an operator. However, this approach does not consider the quality of the results achieved using the recommended operator. Given the large set of operators available today, a trial-and-error approach of all valid operators can be very time consuming.

Expert Systems (ES) apply a set of rules, hand crafted by experts, to recommend operators. *IBM Watson Analytics* [Rais-Ghasem et al. 2013], for example, proposes an ES that defines a set of preconditions for using each operator and a set of scoring rules to measure the usefulness of using an operator on a given dataset. The ESs present the simplest way of providing intelligent assistance. However, it is cumbersome to generate rules to cover all possible cases and all available operators [Serban et al. 2013].

Meta-Learning Systems (MLSs) differ from ESs in that rules are learned automatically from prior runs [Serban et al. 2013]. An MLS models the relationship between input data properties and an operator's performance to recommend operators for new datasets. For example, the KNN learning algorithm is used in the *Data Mining Advisor (DMA)* [Giraud-Carrier 2005] to learn the operators' performance on old data. Using KNN allows adding new operators to the learning model without retraining. The downside of MLSs is that to give good recommendations, they need to be trained on a large set of input datasets representing most of the cases that the system will see. All operators also need to be scored on all training datasets.

Ontology Reasoners (OR) are similar to ESs in the sense that both are based on a set of rules. However, with OR, some rules can be inferred, so that experts do not need to explicitly define every possible rule [Serban et al. 2013]. The *Ontology of Core Data Mining Entities (OntoDM-core)*[49] [Panov et al. 2014] presents a step toward the development of a standard data mining ontology. It provides representations for dataset properties, data mining algorithms, and constraints. The *WINGS (Workflow INstance Generation and Specialization)* system [Gil et al. 2011] implements reasoning to automatically fill in the gaps in an analytics workflow with the best-suited operators.

—*Automatic workflow generation*. It is one of the most advanced forms of intelligent assistance. It combines both data preparation and operation selection. Based on the input data and existing problem, users receive a set of workflows to solve the problem.

One approach is *Case-Based Reasoning (CBR),* which stores a set of cases (workflows) designed by experts and use them as templates for new problems. The *CBR* eliminates the need to train a recommendation model as with *MLS*, but it requires more human involvement to build and clean the cases database. *CBR* is known to be good for domains that are not completely understood, where knowledge is insufficient at the time of implementation but evolves over time [Soltani 2013].

Charest [2007] proposes a data mining intelligent assistance framework based on *CBR* and ontologies to assist non-experts throughout the analytics process. First, it allows users to select the problem domain and then shows them a list of cases that successfully worked with this problem. Users can then select any of these cases and adapt it to the new problem. The framework only supports simple analytics, is not extendable and only supports the *Weka* operations. That being said, Charest's work presents a big step in providing automatic workflow generation to non-experts.

Acquiring a large case base for CBR is not always applicable or easy to do. *Artificial Intelligence (AI) planning* techniques like the *Hierarchical Task Network Planning (HTN)* [Nau et al. 1998] provides a more powerful way to plan analytics workflows for undecidable and unseen problems [Nau et al. 2004]. The *HTN* implemented in *eProPlan*[50] [Kietz et al. 2010] uses hierarchical abstraction planning, which consists

[49]http://www.ontodm.com/doku.php.
[50]http://www.e-lico.eu/eproplan.html.

of starting with an abstract workflow, and then recursively decomposing each of the workflow abstract components until a sequence of applicable operations that satisfies the user's goal is obtained. The *Knowledge Discovery in Databases Virtual Mart (KD-DVM)* system [Diamantini et al. 2009a] utilizes the *KDDONTO* ontology [Diamantini et al. 2009b] to build the workflow using a bottom up approach, which adds operators till the top operator accepts the input dataset.

—*Fault Detection and Handling.* With Big Data, huge amounts of data must be processed, thus having the analytics process fail before completion is expensive and unacceptable. Intelligent assistance can help with minimizing the occurrences and impact of such failures. It can provide *validation* capabilities that include checking input compatibility to operations, ensuring an operation's configuration is valid and that computational and storage resources are available prior to execution. Ontologies can be used to represent the metadata for each operation describing all its characteristics, requirements, and constraints. For example, the *WINGS* system [Gil et al. 2011] uses ontologies and rules to validate the created workflows before execution.

Even with validation, failures still can happen due to situations like unexpected hardware failure, a bug in an operation or a faulty value in the input dataset. For these situations, intelligent assistance can provide failure handling and compensation, which can be divided into *operation-level* and *workflow-level* [Yu and Buyya 2005]. *Operation-level* techniques mask operation failure by trying to re-execute it, execute it on an alternate resource, and using checkpoints to save processed data. *Workflow-level* techniques manipulate the workflow structure to deal with erroneous conditions. They include executing alternate implementations of faulty operations, executing multiple redundant copies of the same operation, and executing user-defined exception handling methods. For interested readers, Russell et al. [2006] present a list of workflow exception patterns and their handling and recovery mechanisms.

2.5. Big Data Analytics User Interfaces

Since Big Data Analytics is a multi-disciplinary science, we can find users from a variety of backgrounds. Analytics solutions usually assume a certain user background and design the solution's interface accordingly. This allows users to work in a familiar environment. However, the consequence is that the full power of an analytics solution is limited to those users with the presumed background knowledge.

In this section, the five main user interface approaches are presented. Their pros and cons are discussed and the intended users subset is defined. *Scripts* present the most flexible environment for experienced users who can use their programming and data mining skills to design the analytics process. *SQL-based interfaces (SQLs)* allow users with a database background to do analytics using the familiar SQL language. *Graph-based interfaces (Graphicals)* allow less experienced users to drag-and-drop different analytic operations to create complex analytics without programming. *Sheets* are for business users, where it provides a familiar Spreadsheet environment. Most recently, *Visualizations* for representing the data in an easy and interactive way.

2.5.1. Scripts.

Instead of having users implement the algorithms from scratch, Scripts provide analytics at the *programming level,* where users create the analytics process by developing programs that interface with these analytic tools. This interface can be the *Command Line Interface (CLI)* or the provided *Application Program Interfaces (APIs)*. Scripts present the most flexible and powerful environment for users. However, Scripts are low-level languages that require writing a lot of obfuscated code for even simple tasks. This makes most novice users prefer other interfaces.

The most widely used solutions in this category are *R*[51] made by and for statisticians, *Matlab*[52] for engineers and *Weka*[53] for data miners. Recently, *Python*[54] has become one of the most widely used languages for analytics, especially by application developers developing proprietary techniques or modifying existing techniques. A large number of analytics libraries have been developed for *Python,* creating a single familiar environment, where users can do both, general-purpose programming and analytics.

Following the same trend, Microsoft offers *F#*[55], a cross-platform functional language. *F#* provides libraries for fetching data from different sources and allows users to use the *.NET* machine learning libraries to do analytics with simple code.

IBM uses *Google Jaql*[56] in its *BigInsights* solution to process Big Data. *Jaql* is a functional language for analyzing large-scale semi-structured JSON data. One of the main *Jaql* features is using lazy evaluation to only fetch data when needed.

Apache Pig[57] uses the *Pig Latin* language [Olston et al. 2008] to abstract coding MapReduce analytics jobs without using the Java MapReduce idioms.

Most of the Scripts solutions are centralized. They provide a large set of algorithms, but can only run on a single node, which means that they are not scalable to meet the Big Data analytic processing needs. Recently, a number of libraries have been developed to provide a scalable distributed implementation of the machine-learning algorithms over *Hadoop* and *Spark,* so that users do not have to write the MapReduce procedures themselves. For example, the *Vowpal Wabbit,*[58] *Apache Mahout,*[59] *Cloudera Oryx*[60] *Oxdata H2O,*[61] and *MLBase*[62] [Kraska et al. 2013] solutions implement distributed machine learning techniques at scale. They are easier to use, require no previous MapReduce knowledge, and are optimized for parallel processing.

2.5.2. SQL-Based interfaces (SQLs). Working with data is synonymous with using SQL to database users. However, with Big Data, data does not only reside in relational databases. SQL solutions provide a unified SQL interface over the different Big Data stores. They provide a familiar environment for database users to work with Big Data. SQL-based solutions provide analytics at the *data level* where users create the analytics process using analytics queries. Using SQL and the standard *JDBC/ODBC interfaces,* the familiar BI tools that auto-generate SQL code can still be used to interact with Big Data.

In SQLs, the standard SQL syntax can be extended to add new functionalities by writing User-Defined Functions (UDFs). The UDFs' development is not done in SQL, which means it is not an option for all users. Without UDFs, users are limited to what the analytics solution provide, making SQLs not as extendible as Scripts. However, SQLs can reduce the learning time for users familiar with the SQL syntax.

SQL solutions can be divided into *SQL-on-Hadoop* and *Machine-Learning SQL*. *SQL-on-Hadoop* solutions only offer basic querying functionalities, like filtering, aggregation, and selection, but have no machine-learning capabilities. The most widely

[51]http://www.r-project.org/.
[52]http://www.mathworks.com/products/matlab/.
[53]http://www.cs.waikato.ac.nz/ml/weka/.
[54]https://www.python.org/.
[55]http://fsharp.org/.
[56]https://code.google.com/p/jaql/.
[57]http://pig.apache.org/.
[58]http://hunch.net/∼vw/.
[59]https://mahout.apache.org/.
[60]https://github.com/cloudera/oryx.
[61]http://0xdata.com/h2o-2/.
[62]http://www.mlbase.org/.

```
CREATE TABLE <model_name> AS
SELECT  <label>,  cast(<<feature>> AS <<datatypes>>) AS feature FROM {
 SELECT TRAIN_MODEL(<<feature>>,<label>) AS
(<label>,<<feature>>,<<weight>>,<<covar>>)
  FROM <training_dataset_table> ) t
GROUP BY <label>, <<feature>>;
```

Fig. 6. Training using Hivemall.

```
CREATE VIEW <prediction_table_name>
AS SELECT <rowid>, <score>, <label> FROM (
  SELECT <t.rowid>, <m.label>, <m.score>
  FROM <testing_dataset_table> t LEFT OUTER JOIN
    <model_name> m ON (<<t.feature>> = <<m.feature>>) );
```

Fig. 7. Prediction using Hivemall.

known SQL-on-Hadoop solution is *Apache Hive* [Thusoo et al. 2009], which translates the user's *HiveQL* queries to MapReduce batch jobs. *Google Dremel* [Melnik et al. 2010] (publicly available as *Google BigQuery*[63]service), *Cloudera Impala,*[64] and *Apache Drill*[65] provide interactive querying on Big Data. *Spark SQL*[66] (previously known as *Shark*) [Xin et al. 2013] uses in-memory computations to further accelerate query processing. *Microsoft SCOPE* [Zhou et al. 2012] is a SQL-like language that creates optimized query execution plans inspired by parallel databases optimization techniques for Microsoft's MapReduce solutions *Cosmos* and *Dryad* [Isard et al. 2007].

The *Machine Learning SQL* solutions provide the machine learning capabilities the SQL-on-Hadoop solutions lack. From these solutions, the *QDrill* [Khalifa et al. 2016], an extension to *Apache Drill* that allows running *WEKA* machine learning algorithms in a distributed fashion from within the SQL query. The *Data Mining Query Language (DMQL)* [Han et al. 1996] attempts to establish a standard for data mining query languages. The *DAEDLUS* Framework [Ortale et al. 2008] introduces the *MO-DMQL* that can be expressed using the *3W* algebraic framework [Johnson et al. 2000], which is similar to relational algebra. *Microsoft* introduces the *Data Mining Extension (DMX)* query language [Tang et al. 2005] in its SQL server to run in-database analytics. *Hivemall* [Yui and Kojima 2013] extends *HiveQL* with a scalable machine learning library, where models can be created and used from within *HiveQL* statements as in Figures 6 and 7, respectively. Meo et al. [1996] propose a specialized SQL extension for only association rules mining. *SQL-TS* [Sadri et al. 2001] is another specialized SQL extension that is highly optimized for complex time series queries.

Looking at the existing machine-learning SQL solutions, we find the following shortcomings. Both *DMQL* and *MO-DMQL* support all types of data-modeling (classification, prediction, clustering, and association rules) but not the data preparation and transformation operations. The language from Meo et al. [1996] only supports modeling association rules and *SQL-TS* only supports complex-time series queries. The closest to achieving a fully functioning SQL interface for Big Data Analytics are *QDrill*, *DMX,* and *Hivemall*. They support different data models and use SQL or SQL-like for data exploration and preparation. However, *DMX* and *Hivemall* supported algorithms are still limited compared to other analytics solutions as they rewrite the algorithms to run in a distributed fashion. *QDrill* does not have this shortcoming as it relies on distributing the WEKA algorithms without rewriting them.

[63]https://cloud.google.com/bigquery/.
[64]http://impala.io/.
[65]http://incubator.apache.org/drill/.
[66]https://spark.apache.org/sql/.

2.5.3. Graph-Based Interfaces (Graphicals). With the increasing sophistication of analytics performed by organizations, Big Data Analytics has become a very complex process that can incorporate tens or even hundreds of operations. *Graphicals* support such sophistication without the need to write code. Their interface typically consists of two main areas: a *panel* and a *canvas*. The *panel* holds a list of supported operations, where each operation represents a workflow *task* that is always carried out in full [vanderAalst and vanHee 2004]. Users can drag and drop these operations into the *canvas* and connect them together to create a *workflow*. Some *Graphicals* support *sub-workflows* that consist of a set of *tasks* and possibly further *sub-workflows* to allow reusing frequently occurring *tasks*. However, Graphicals have a number of shortcomings, namely not all of them support distributed execution, conditionals, loops, and extensions.

A large number of Graphical solutions have been developed both by industry and academia. *RapidMiner*[67] is the most widely used Graphical solution. It uses sub-workflows, wizards, and quick fixes extensively, making complex workflows easier to design and interpret. It also has connectors for *Weka* and *R*. *Radoop* [Prekopcsak et al. 2011] extends *RapidMiner* using *Hive* and *Mahout* to support analytics at scale on top of *Hadoop,* while hiding the complexity of distributed Data Analytics.

IBM SPSS Modeler[68] is a commercial solution providing a range of advanced algorithms and techniques for text analytics, decision management, predication and optimization. *Modeler* supports conditionals, iterations, and sub-workflows using *Python* scripts, which requires having good programming skills. *IBM Analytic Server*[69] extends *Modeler* to provide scalable distributed analytics on *Hadoop*.

WINGS (Workflow INstance Generation and Specialization) [Gil et al. 2011] is an intelligent Graphical solution that uses semantic reasoning to help users design complex workflows. *WINGS* uses workflow templates to automatically complete and validate workflows based on the operations' and datasets' requirements. *WINGS* can be extended with new operators written in any language and encapsulated in its Shell script wrapper.[70] It supports parallel execution but does not explicitly support sub-workflows or control constructs; however, this can be done implicitly [Sethi et al. 2013].

Kantere and Filatov [2015] propose a framework for expressing complex workflows in an abstract manner, adaptable to the user role, interest, and expertise. The framework also prepares the workflow tasks for execution on a range of engines based on the execution semantics of the individual tasks.

2.5.4. Sheets. *Sheets* are the closest to providing consumable analytics as they offer the most familiar environment for business users but they are not designed to handle Big Data. They are more focused on data exploration and preparation, and require moving the prepared data to another solution for modeling, which is very costly with Big Data.

Microsoft offers the *Excel Analytics* solution (*Power Query,*[71] *Data Analysis Expressions (DAX) Language,*[72] *Data Mining add-in*[73]), where users can use the *Power Query* Sheet GUI to fetch and merge data from different sources and do data preparation and transformation. The *DAX* language keeps track of all executed steps to support undos,

[67]http://rapidminer.com/.
[68]http://www-01.ibm.com/software/analytics/spss/products/modeler/.
[69]http://www-03.ibm.com/software/products/en/spss-analytic-server/.
[70]http://goo.gl/mhgnDH.
[71]http://social.technet.microsoft.com/wiki/contents/articles/18542.power-query.aspx.
[72]http://social.technet.microsoft.com/wiki/contents/articles/677.powerpivot-data-analysis-expressions-dax-language.aspx.
[73]http://office.microsoft.com/en-ca/excel-help/data-mining-add-ins-HA010342915.aspx.

and the *Data Mining add-in* sends the prepared data to the SQL server for modeling. *Power Query* is limited to a maximum of 1,000,000 records[74] per dataset. For that, Microsoft offers *Microsoft Tabular*, a server-based solution for *in-database* analytics on structured data.[75] *Microsoft Daytona* [Barga et al. 2012] can be used to offload Excel's operations to *MapReduce* on the cloud for distributed processing.

Google OpenRefine[76] is a browser-based, spreadsheet-style tool designed for data exploration and preparation but does not support data modeling. Unlike Excel's *Power Query*, *OpenRefine* can handle any number of records as it only shows a data sample to users. However, *OpenRefine* still runs on a single machine and is thus limited to the machine's memory and computational resources.

IBM Cognos[77] provides a simple GUI for manipulating data in the form of spreadsheets and data cubes. As with previous solutions, *Cognos* runs on a single machine, making it limited to small and medium datasets. IBM offers the *BigInsights BigSheets*[78] which is a browser-based, spreadsheet-style tool in *IBM InfoSphere BigInsights*[79] that enables business users to explore, manipulate, and analyze Big Data using the underlying *Hadoop* layer for distributed processing.

2.5.5. Visualizations. With Big Data, users can drown in the excessive volumes of data. This can lead to analyzing the wrong or incomplete set of attributes or becoming frustrated with the whole analytics process. Visualization solutions are designed for business users to allow them to have an interactive conversation with their data.

IBM Watson Analytics[80] [Rais-Ghasem et al. 2013] allows users to go from data to analysis in seconds without any setup or configuration. It allows users to use visualization and natural language to understand their data. It relies on *IBM SPSS Analytics Server* and *IBM Big Insights* to automatically build and use data models, and on the *Rapidly Adaptive Visualization Engine (RAVE)* for interactive visualizations. On the down side, *Watson Analytics* requires data to be pre-cleaned outside the *Watson* framework. It does not output an analytic model to use for further analysis, and it does not specify how the analysis outputs were achieved making it hard to trust the outputs.

Microsoft offers *Power View*[81] as part of its *Power BI* solution for *Excel 2013* and *SQL Server 2012*. As with *Watson Analytics, Power View* provides interactive data exploration and visualization to support intuitive ad-hoc reporting in a familiar environment. However, *Power View* does not support data modeling, and it requires loading all data first, which limits the analysis to the size of the available memory.

SAS Visual Analytics[82] powered by the *SAS analytics framework*[83]*,* is designed to empower business users with limited technical skills to do analytics using *Hadoop* without any programming. In addition to data exploration, users can do sophisticated analytics like forecasting using a drag-and-drop approach.

MicroStrategy Visual Insight[84] and *tableau*[85] allow users to access data from multiple sources (spreadsheets, databases, or HDFS) and present them in interactive visualiza-

[74]http://technet.microsoft.com/en-us/library/hh212940.aspx.
[75]http://msdn.microsoft.com/en-us/library/gg492165.aspx.
[76]http://openrefine.org/.
[77]http://www-01.ibm.com/software/analytics/cognos/.
[78]http://goo.gl/d4E9PG.
[79]http://www-01.ibm.com/software/data/infosphere/biginsights/.
[80]https://www.analyticszone.com/homepage/web/displayNeoPage.action?CT=ISM0056.
[81]http://goo.gl/OvOVh8.
[82]http://www.sas.com/en_us/software/business-intelligence/visual-analytics.html.
[83]http://www.sas.com/.
[84]http://www.microstrategy.com/us/software/products/visual-insight.
[85]http://www.tableau.com/.

tions for analysis. The analytics capabilities of those solutions are limited as they do not implement machine learning techniques. Their power comes from their availability on mobile devices and allowing users to share visualizations.

2.6. Big Data Analytics Deployment Methods

The ecosystem is composed of many overlaid components that need to be integrated together. Deploying and maintaining such ecosystem can be complex, challenging, and beyond the capabilities of the in-house IT team in many organizations.

Deployment methods differ in terms of access needs, IT cost, security, data privacy, scalability, maintenance complexity and time-to-first-insight. First, the *Product* model, where users buy and setup the solution on their infrastructure. As solutions become more complex, there is a shift towards a *Service* deployment model where the setup and maintenance of the solutions are outsourced.

2.6.1. Product. Organizations use the product deployment model to ensure their data security and privacy, and to handle large volumes of on-site data. However, this model (i) requires a large upfront cost to buy the solution, (ii) needs an IT team to set up and maintain the solution, and (iii) has limited scalability bounded by the organization's resources.

Most of the analytics ecosystem components are available for free (e.g., Hadoop), however, integrating them is technically challenging and time consuming. A number of solutions integrates all needed components into *a Software Bundle (SW)*, which organizations can buy and deploy on their infrastructure. *Hortonworks Data Platform (HDP)*[86] provides an integrated solution using open source solutions like Hadoop, Pig, Hive, Spark, Yarn, etc. Other solutions provide a *Software/Hardware Bundle (SW/HW),* usually using powerful servers. Organizations can buy these solutions and build their own Big Data Analytics cloud. *Oracle Exalytics*[87] is an example, providing powerful cloud nodes (servers) designed for in-memory analytics.

2.6.2. Service. In the *Service* model, a service provider gives organizations on-demand access to the Big Data Analytics Ecosystems and organizations are charged on a pay-per-use basis. This model allows organizations to outsource the software and infrastructure setup and maintenance to the service provider, and gives them better scalability and availability using the larger and more reliable service provider's infrastructure. For the previous reasons, the service model is becoming one of the main factors in the increased adoption of Big Data Analytics [Güemes et al. 2013]. However, organizations are still challenged by the service model's data security and privacy, and the cost of moving their Big Data to the provider's cloud.

One proposed solution to the service model's challenges is Hybrid Clouds, where the architecture is split, with data storage and processing residing on the organization's infrastructure, while the coordination and analytics services are provided by the public cloud [Güemes et al. 2013]. While this solution solves the data movement and privacy problems, it eliminates a lot of the *Services* advantages, where the organization still has to setup and maintain its own analytics ecosystem and infrastructure. Another solution would be streaming the data to the provider's cloud and running the analytics on the data as it arrives. The main problem facing the implementation of this solution is that most analytics solutions are batch based and need all data in order to start.

Analytics Platform as a Service (APaaS) follows the *Platform as a Service (PaaS)* cloud deployment model to deliver highly customizable web-based services covering the end-to-end process of an analytics solution, from acquiring data to reporting results

[86]http://hortonworks.com/hdp/.
[87]https://www.oracle.com/engineered-systems/exalytics/index.html.

to end-users. The *APaaS* model provides a platform for organizations to develop, run, manage and share their analytics without the complexity of building and maintaining the software ecosystem or the infrastructure.

Amazon Web Services (AWS)[88] provide an infrastructure for the different components of the analytics ecosystem: *EBS* and *S3* for scalable storage, *EC2* for scalable on-demand computation, *EMR* for MapReduce as a service, *RDS* for storing structured data, and *DynamoDB* for semi- and unstructured data.

Google BigQuery[89] is an online on-demand *APaaS* solution, where users can upload their datasets to the Google cloud, analyze it using SQL-like queries, and get charged per terabyte processed and stored. IBM offers the *Analytics for Hadoop*[90] *APaaS* solution, which is a cloud version of their *BigInsights* solution deployed on their *Bluemix PaaS* cloud. *RapidMiner Cloud*[91] offers a similar service. HP offers the *HAVEn OnDemand*[92] *APaaS* solution that runs on their version of the *OpenStack*[93] cloud platform, code named *HP Helion*[94]. *EMC*^{2} offers the *Greenplum Unified Analytics Platform (UAP)* [EMC 2013], a high-end *APaaS* solution hosted on the *EMC Greenplum Data Computing Appliance (DCA)* cloud, which provides massive parallel processing power to speed up the analytics performed using *SAS* and other analytics engines. Xu et al. [2015] propose an architecture for providing real-time APaaS. The architecture uses RESTful web services to wrap and integrate the different data storage and data mining services.

IBM Watson Analytics,[95] *Tableau Online,*[96] and *MicroStrategy Analytics Express*[97] use a *Software-as-a-Service (SaaS)* model where users can only analyze their data using interactive visualizations without being able to develop their analytics processes.

2.7. Summary

In this subsection, we summarize in Table I the different approaches used in the six pillars. It should be noted that an ecosystem can implement one or more approach within the same pillar to support more use cases. Approaches within a pillar complement one another and are not mutually exclusive.

3. USE CASES

The Big Data Analytics Ecosystem is a stack of components working together to answer some business questions using the available data. Each organization should be able to assemble its own personalized ecosystem based on its *objectives*, *data*, *size* and *technical capabilities*. The proposed six pillars can assist in building this ecosystem.

In terms of the organization analytics objectives, raw data in its unprocessed state does not offer much value to business. Organizations have to dig deep to pull the insights. Organizations differ on how deep they can/want-to dig. Some organizations are satisfied with *Descriptive Analytics (D)*, which tells them about *what is going on*, either on historical or real-time data. This includes approaches such as profiling, segmentation, or clustering. Other organizations go a step deeper and implement *Predictive Analytics (F)* to know *what is likely to happen*. Predictive analytics utilizes a variety

[88]http://aws.amazon.com/.
[89]https://cloud.google.com/bigquery/.
[90]http://www-01.ibm.com/software/data/infosphere/hadoop/trials.html.
[91]https://rapidminer.com/documentation/cloud/.
[92]http://www.vertica.com/hp-vertica-products/ondemand/.
[93]https://www.openstack.org/.
[94]http://www.hpcloud.com/.
[95]https://www.analyticszone.com/homepage/web/displayNeoPage.action?CT=ISM0056.
[96]http://www.tableau.com/products/online.
[97]http://www.microstrategy.com/us/free/express.

Table I. Big Data Approaches for the Six Pillars

	Approach	*Advantages*	*Disadvantages*	*Examples*
Storage	RDBMS	- High performance - Ensure consistency	- Low scalability - Mostly only store structured data	MySQL Cluster, ScaleDB, VoltDB, ScaleBase
	DFS	- Highly scalable - Store any data format	- No support for querying data	GFS, HDFS, CFS
	NoSQL	- Highly scalable - Store any data format - Support querying data	- Mostly only support eventual consistency	DynamoDB, Voldemort, Redis, Riak, MemcacheDB, HBase, BigTable, Cassandra, PNUTS, MongoDB, CouchDB, SimpleDB, Neo4j, OrientDB, InfiniteGraph
Processing	Batch	- Execute series of jobs without manual intervention. - Fault tolerant	- High IO overhead - No loops support - Long execution time	MapReduce, Hadoop, Spark, HOP, Muppet, DistributedWekaBase, DistributedWekaHadoop, RHadoop, Radoop, IBM Analytic Server
	Interactive	- Shorter execution time - Support ad-hoc queries	- Weaker fault tolerance - No loops support	Dremel, BigQuery, PowerDrill, Drill, Imapala, Tez
	Iterative	- Loop support - Machine learning support	- Weaker fault tolerance	HaLoop, iMapReduce, PrIter, M3R, Twister, Mahout, Hivemall, MLBase, Oryx, H2O, Deeplearning4j
	Incremental	- Handle data-in-motion (realtime)	- Weaker fault tolerance	Storm, S4, Samza, Trill, Stat!, Naiad, MillWheel, InfoShpereStreams, Nova, SparkStreaming
	Approximate	- Fast retrieval of partial results	- Long execution time to get the complete results	EARL,BlinkDB
	In-Database	- Ensure consistency - Support in-database analytics	- Low scalability - Mostly only store structured data	SSAS, MADLib, HadoopDB
Orchestration	Resource Scheduling	- Increase resource utilization and reduce cost	- High IO and network overhead	YARN, Adoop, IReS
	Data Locality	- Reduce IO and network overhead	- Increase waiting time for resources holding data	Pegasus, [Zaharia et al. 2010a; Guo et al. 2012], Pixida
	Resource Provisioning	- Reduce execution time - Reduce execution cost	- Increase cost to reduce execution time - Increase execution time to reduce cost	RSMaximizer, Conductor, Purlieus, [Kllapi et al. 2011], [Mian et al. 2013; Mao and Humphrey 2013], Mesos, [Li et al. 2015]
	Data Provisioning	- Reduce IO and network overhead	- Increased waiting time for data replication	Scarlett, DARE, CDRM
Assistance	Static	- Easy to develop	- Not very helpful	Tooltips, Help Pages, Wizards
	Intelligent	- Provide suggestions on case bases to help users with different experience levels	- Hard to develop	Ontologies, IBM Watson Analytics, MiningMart, DMA, WINGS, [Charest 2007]
Interface	Script CLI, API	- Very flexible and powerful to expert users and programmers	- Low level - Hard to learn and use	R, Matlab, Weka, Python, F#, Jaql, Pig, Vowpal Wabbit, Mahout, Oryx, H2O, MLBase
	SQL	- Provide SQL interface to work on Big Data for database administrators	- Less flexible than Scripts	DMQL, DAEDLUS, QDrill, DMX, Hivemall, SQL-TS, [Meo et al. 1996]
	Graphical	- Provide drag-and-drop interface for novice users - Allow creating sophisticated workflows	- Less flexible than Scripts	RapidMiner, IBM SPSS Modeler, WINGS, [Kantere and Filatov 2015]
	Sheet	- Provide familiar interface for business users	- Only supports simple operations	Excel Analytics, Tabular, OpenRefine, Cognos, BigSheets
	Visualization	- Provide interactive analysis for business users	- Only supports simple operations	IBM Watson Analytics, PowerView, SAS Visual Analytics, Microstrategy Visual Insight, tableau
Deployment	Product	- Better data privacy and security	- More expensive and time consuming - Less scalable	HDP, Oracle Exalytics
	Service	- Cheaper - Faster to setup - More scalable	- Weaker on the data privacy and security - Difficult to work when huge amounts of data need to be uploaded	AWS, BigQuery, IBM Analytics for Hadoop, HP HAVEn OnDemand, EMC^2 Greenplum UAP, IBM Watson Analytics, Tableau Online, MicroStrategy Analytics Express, RapidMiner Cloud, [Xu et al. 2015]

Table II. Big Data Analytics Ecosystem Use Cases

Use Case			*Ecosystem Components*					
Objective	*Data*	*Available skills*	*Storage*	*Processing*	*Orchestration*	*Assistance*	*Interface*	*Deployment*
D	S	b, s, d	RDBMS	Batch, Interactive, Incremental, Approximate, In-Database	calculations	Intelligent	Sheet, Visual, SQL	SM -> Service L -> Product
D	U	p, s	DFS, NoSQL	Batch, Interactive, Incremental, Approximate	calculations	Intelligent	Script, Visual, SQL	SM -> Service L -> Product
F	S	p, s, d	RDBMS	Approximate, Iterative, In-Database	workflows	Intelligent, static	Script, Graphic,	SM -> Service L -> Product
F	U	p, s	DFS, NoSQL	Approximate, Iterative	workflows	Intelligent, static	Script, Graphic,	SM -> Service L -> Product
P	S	p, s, d	RDBMS	Approximate, Iterative, In-Database	workflows	Intelligent, static	Script, Graphic,	SM -> Service L -> Product
P	U	p, s	DFS, NoSQL	Approximate, Iterative	workflows	Intelligent, static	Script, Graphic,	SM -> Service L -> Product

D: Descriptive analytics – F: Predictive analytics – P: Prescriptive analytics; S: Structured data – U: semi and Unstructured data.
b: business user – s: statistician – d: database administrator – p: programmer; SM: Small/Medium organization – L: Large organization.

of statistical, data-mining and machine-learning techniques to make predictions about what might happen in the future given the recent and historical data. Fewer organizations go all way to *Prescriptive Analytics (P)* to produce multiple futures (*What-if scenarios*) for decision makers to choose the best course of action. Prescriptive analytics adds a feedback loop on the predictive model to track the action's outcome.

Different organizations also deal with different data formats. Some only handle *structured data (S)*, others need to process *semi-structured and unstructured data (U)*. Organizations can be *Small-Medium (SM)* or *Large (L)* which affects their ecosystem deployment model. Their analytics technical capabilities can also vary from relying on *business users (b)*, *statisticians (s)*, *programmers (p)*, or *database administrators (d)*.

In Table II, we present a set of recommendations for components to be used in each of the six pillars to form an ecosystem that meets the business needs in different use cases. The first three columns describe the use case (The objective of the analytics process, the type of data and the available users' skills). Unlike buying off-the-self ecosystems, these recommendations can guide an organization in building their own ecosystem that is optimized to their needs.

The reasons for recommending these components for the use cases in Table II are as follows. For use cases that involves Small-Medium organizations, the Service deployment model is recommended to save on infrastructure and IT costs. Large organizations can afford using the Product deployment model, which may better suit their privacy and security policies and better handle their larger data.

Intelligent assistance is recommended for all use cases whenever available for any type of analytics. However, with experienced users carrying out advanced predictive or prescriptive analytics, static assistance can be sufficient.

Storage depends on the kind of data at hand. A combination of datastores can be used for use cases with a mixture of structured and unstructured data. Structured data can be moved to DFS or NoSQL, minimizing the number of data stores to use.

Not all users can perform all types of analytics. For instance, business users do not have the technical skills to do predictive and prescriptive analytics. For use cases requiring those types of analytics, programmers and statisticians need to be hired. The ecosystem interface needs to support the backgrounds of all system users. While business users prefer sheets and visualizations, database administrators prefer SQL, and programmers and statisticians prefer scripts.

For descriptive analytics use cases, minimum orchestration is required since it is mostly summarizations and calculations. A simple caching mechanism for frequent queries can be sufficient. Predictive and prescriptive analytics use cases include data transformations, data modeling, and calculations. This workflow of operations requires advanced orchestration to achieve good performance.

In terms of processing, descriptive analytics use cases are the most demanding, they need to handle different data exploration approaches. Interactive and approximate processing are required to try different operations on a subset of the data before implementing them, if useful, on the whole data set using batch processing. Incremental processing is also required to continuously update the results as more data becomes available. Predictive and prescriptive analytics use cases usually rely on approximate processing for trying different techniques on sample data and on iterative processing to run data models for predictions. If all data is structured in a use case, then in-database processing is highly recommended to eliminate data movement.

4. FUTURE DIRECTIONS

Each of the discussed analytics solutions brings some features not available in the others, but also adds some limitations and overheads. While there has been a continuous improvement in analytics solutions to address different analytics scenarios, there are still some gaps. In this section, we define the specifications for future Big Data Analytics Ecosystems to provide improved and broader support to organizations needs for the different analytics scenarios.

—*Extensibility*. Solutions can become obsolete if they are designed to support only a fixed number of data stores and analytics techniques. It is thus crucial for future Big Data Analytics solutions to have a plug-in architecture to support adding new algorithms and datastores with minimal modifications to the solution.

—*Seamless Data Integration*. With organizations having their data in multi-structured formats and distributed across heterogeneous data stores, future Big Data Analytics solutions need to provide an abstraction layer to hide these details from users. They need to allow users to join this data together while minimizing data movement.

—*Seamless Engine Integration*. Analytics consists of multiple operations to transform raw data to meaningful insights. Usually in the data preparation phase, *SQL* or script engines are used to have interactive ad-hoc queries. Then, in the modeling phase, machine-learning engines like *Mahout*, *Weka,* and the like. are used. This requires future Big Data Analytics solutions to be able to integrate engines in a single pipeline and handle the inter-engine compatibility issues like handling the different file formats.

—*Distributed Processing*. The majority of existing machine-learning engines are local engines like *R*, running on local machines and cannot scale to deal with Big Data. Distributed engines like *Mahout* and local engine extensions like *RHadoop* re-implement the algorithms which makes their development time consuming. To limit the need for reimplementation, future Big Data Analytics solutions can deploy local engines in parallel and seamlessly distribute the processing among them.

—*Approximate and Incremental Processing*. While designing the analytics process, users are not usually sure of which operation to use and how it is going to affect the results. This makes them try different operations, which can be time consuming if they run on the whole dataset. Thus, future Big Data Analytics solutions need to provide approximate and incremental results to give users indications of the results of an operation without running it on the whole dataset and to support real-time analytics.

—*Execution and Storage Optimization.* Having the data distributed among cloud nodes requires optimizing the analytics execution to utilize the shared resources while minimizing data movement. Future Big Data Analytics solutions need to predict what data will be needed by future operations and to make sure that this data is available on the underutilized nodes for the future operations.

—*Fault-Tolerance.* Big Data analytics can run for long periods. Having to restart from the beginning in case of failure is not acceptable. Future Big Data Analytics solutions need to support fault-tolerance while minimizing its impact on performance.

—*Intelligent Assistance.* With the huge set of available analytics techniques, even experts sometimes need help. Intelligent assistance provides customized assistance to meet the user's skill level and problem at hand. Providing intelligent assistance in future Big Data Analytics solutions is important for analytics to be more accessible to organizations, to minimize the time-to-insights and to enhance the quality of analytics.

—*Multiple User Interfaces.* For future Big Data Analytics solutions, a combination of user interfaces should be provided to meet the needs of users of different skillsets. Future analytics frameworks should provide the flexibility and the ability to add user defined operations of the scripts; the easy-to-use drag-and-drop interface for designing complex workflows of the Graphicals; the ad-hoc capabilities of the SQLs; the familiar environment of the sheets, and the easy interpretation of the visualizations.

—*Service-Based.* Having multiple analytics engines and datastores in the analytics process makes the solution hard to set up, maintain, and scale. Future Big Data Analytics solutions can use the Service approach to outsource all these problems to the service provider. However, data privacy and security issues need to be addressed.

5. CONCLUSIONS

With the existence of too many solutions forming the Big Data Analytics Ecosystem, practitioners and researchers can get lost deciding what to use in which context. A weak component in the ecosystem can cause the whole ecosystem to function inefficiently. Thus, this study defines six pillars on which to build the Big Data Analytics Ecosystem, namely Storage, Processing, Orchestration, Assistance, Interfacing, and Deployment. For each of these pillars, different approaches are discussed and popular systems are presented. The pillars form a taxonomy that aims to give an overview on the field, to guide organizations and researchers to build their Big Data Analytics Ecosystem, and help to identify challenges and opportunities in the field. A set of recommendation of personalized ecosystems for different business use cases is presented based on the proposed six pillars. The taxonomy, together with the personalized ecosystems recommendations, can assist practitioners in building more optimized ecosystems to better meet their needs. Finally, this work identifies the challenges in the domain and provides guidelines for building future Big Data Analytics Ecosystems.

REFERENCES

Cristina L. Abad, Yi Lu, and Roy H. Campbell. 2011. DARE: Adaptive Data Replication for Efficient Cluster Scheduling. In *Proc. of the 2011 IEEE Int'l Conf. on Cluster Computing (CLUSTER'11)*. 159–168.

Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. 2009. Column-oriented database systems. *Proc. VLDB Endow.* 2, 2, 1664–1665.

Sunitha Abburu. 2012. A survey on ontology reasoners and comparison. *Int'l Journal of Computer Applications.* 57, 17, 33–39.

Azza Abouzied, Kamil Bajda-Pawlikowski, Jiewen Huang, Daniel J. Abadi, and Avi Silberschatz. 2010. HadoopDB in action: building real world applications. In *Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'10)*. 1111–1114.

Sameer Agarwal, Anand P. Iyer, Aurojit Panda, Samuel Madden, Barzan Mozafari, and Ion Stoica. 2012. Blink and it's done: Interactive queries on very large data. *Proc. VLDB Endow.* 5, 12, 1902–1905.

Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: fault-tolerant stream processing at internet scale. *Proc. VLDB Endow.* 6, 11, 1033–1044.

Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. 2011. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proc. of the 6th Conf. on Computer Systems (EuroSys'11).* 287–300.

J. Chris Anderson, Jan Lehnardt, and Noah Slater. 2010. *CouchDB: the definitive guide.* O'Reilly, Inc.

Shivnath Babu and Herodotos Herodotou. 2013. Massively parallel databases and MapReduce systems. *Found. Trends Databases* 5, 1, 1–104.

Roger S. Barga, Jaliya Ekanayake, and Wei Lu. 2012. Project Daytona: Data analytics as a cloud service. In *Proc. of the 2012 IEEE 28th Int'l. Conf. on Data Engineering (ICDE'12)*. 1317–1320.

Adam Barker and Jano Van Hemert. 2007. Scientific workflow: A survey and research directions. In *Proc. of the 7th Int'l Conf. on Parallel Processing and Applied Mathematics (PPAM'07)*. 746–753.

Mike Barnett, Badrish Chandramouli, Robert DeLine, Steven Drucker, Danyel Fisher, Jonathan Goldstein, Patrick Morrison, and John Platt. 2013. Stat!: An interactive analytics environment for big data. In *Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'13)*. 1013–1016.

Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. 2004. OWL web ontology language reference. *W3C Recommendation.*

Chidansh Amitkumar Bhatt and Mohan S. Kankanhalli. 2011. Multimedia data mining: state of the art and challenges. *Multimedia Tools Appl.* 51, 1, 35–76.

Christian Bockermann and Hendrik Blom. 2012. Processing data streams with the rapidminer streams-plugin. In *Proc. of the RapidMiner Community Meeting and Conference.*

Jan Bosch. 2009. From software product lines to software ecosystems. In *Proc. of the 13th Int'l Software Product Line Conf. (SPLC'09)*. 111–119.

Eric Brewer. 2012. CAP twelve years later: How the rules have changed. *Computer* 45, 2, 23–29.

Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. 2010. HaLoop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.* 3, 1–2, 285–296.

Josiah L. Carlson. 2013. *Redis in Action*. Manning Publications Co., Greenwich, CT.

Rick Cattell. 2011. Scalable SQL and NoSQL data stores. *SIGMOD Rec.* 39, 4, 12–27.

Badrish Chandramouli, Jonathan Goldstein, and Songyun Duan. 2012. Temporal analytics on Big Data for web advertising. In *Proc. of the 28th IEEE Int'l Conf. on Data Engineering (ICDE'12)*. 90–101.

Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, Danyel Fisher, John C. Platt, James F. Terwilliger, and John Wernsing. 2014. *The Trill Incremental Analytics Engine*. MSR-TR-2014-54.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4, 26 pages.

Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. 2000. CRISP-DM 1.0: Step-by-step data mining guide. Retrieved January 20th, 2015 from http://goo.gl/3YjNiX.

Michel Charest. 2007. *Intelligent data mining assistance via case-based reasoning and a formal ontology*. Ph.D dissertation, Universite du Quebec, Quebec, Canada.

Xiaojun Chen, Yunming Ye, Graham Williams, and Xiaofei Xu. 2007. A survey of open source data mining systems. In *Proc. of the 2007 Int'l Conf. on Emerging Technologies in Knowledge Discovery and Data Mining (PAKDD'07)*. 3–14.

Kristina Chodorow. 2013. *MongoDB: The Definitive Guide.* O'Reilly Media, Inc.

Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. 2010. MapReduce online. In *Proc. of the 7th USENIX Conf. on Networked Systems Design and Implementation (NSDI'10)*.

Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. 2008. PNUTS: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.* 1, 2, 1277–1288.

Zhan Cui, Ernesto Damiani, and Marcello Leida. 2007. Benefits of ontologies in Real Time Data Access. In *Proc. of the 2007 IEEE Int'l. Conf. on Digital Ecosystems and Technologies (IEEE DEST 2007)*. 392, 397.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (January 2008), 107–113.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. In *Proc. of 21st ACM Symp. on Op. Sys. Principles (SOSP'07).* 205–220.

Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. 2009. Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.* 25, 5, 528–540.

Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. 2005. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.* 13, 3, 219–237.

Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. 2011. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semant. Web* 2, 2, 71–87.

Claudia Diamantini, Domenico Potena, and Emanuele Storti. 2009a. Ontology-driven KDD process composition. In *Advances in Intelligent Data Analysis VIII*, Lecture Notes in Computer Science, vol. 5772, 285–296.

Claudia Diamantini, Domenico Potena, and Emanuele Storti. 2009b. KDDONTO: An ontology for discovery and composition of KDD algorithms. In *Proc. of the ECML-PKDD Workshop on Service-Oriented Knowledge Discovery.* 13–24.

Katerina Doka, Nikolaos Papailiou, Dimitrios Tsoumakos, Christos Mantas, and Nectarios Koziris. 2015. IReS: Intelligent, multi-engine resource scheduler for big data Analytics Workflows. In *Proc. of the 2015 ACM SIGMOD Int'l Conference on Management of Data (SIGMOD'15).* 1451–1456.

Christos Doulkeridis and Kjetil NØrvåg. 2014. A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal* 23, 3, 355–380.

Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. 2010. Twister: A runtime for iterative MapReduce. In *Proc. of the 19th ACM Int'l Symposium on High Performance Distributed Computing (HPDC'10)*. 810–818.

Yehia Elshater, Patrick Martin, Dan Rope, Mike McRoberts, and Craig Statchuk. 2015. A study of data locality in YARN. In *Proc. of the 2015 IEEE Int'l Congress on Big Data.* 174–181.

EMC. 2013. EMC accelerates journey to big data with business analytics-as-a-service. Retrieved January 20th, 2015 from http://goo.gl/LyAxa7.

Avrilia Floratou, Jignesh M. Patel, Eugene J. Shekita, and Sandeep Tata. 2011. Column-oriented storage techniques for MapReduce. *Proc. VLDB Endow.* 4, 7, 419–429.

Lars George. 2011. *HBase: The Definitive Guide.* O'Reilly Media, Inc.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. 29–43.

Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro Gonzalez-Calero, Paul Groth, Joshua Moody, and Ewa Deelman. 2011. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems* 26, 1, 62–72.

Christophe Giraud-Carrier. 2005. The Data Mining Advisor: Meta-learning at the Service of Practitioners. In *Proc. of the 4th Int'l Conf. on Machine Learning and Applications (ICMLA'05)*. 113–119.

Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam A. M. Capretz. 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications* 2, 1, 22.

Thomas R. Gruber. 1993. A translation approach to portable ontology specifications. *Knowl. Acquis.* 5, 2, 199–220.

Celestino Güemes, Jordan Janeczko, Thierry Caminel, and Matthew Roberts. 2013. Data analytics as a service: Unleashing the power of cloud and Big Data. White Paper. Atos Scientific Community. Retrieved January 20th, 2015 from http://goo.gl/hqIC9n.

Zhenhua Guo, Geoffrey Fox, and Mo Zhou. 2012. Investigation of data locality and fairness in MapReduce. In *Proc. of 3d Int'l Workshop on MapReduce and Its Applications Date (MapReduce'12).* 25–32.

Mocky Habeeb. 2010. *A developer's guide to Amazon SimpleDB*. Addison-Wesley Professional.

Alexander Hall, Olaf Bachmann, Robert Büssow, Silviu Gănceanu, and Marc Nunkesser. 2012. Processing a trillion cells per mouse click. *Proc. VLDB Endow.* 5, 11, 1436–1446.

Mohammad Hamdaqa, Mohamed M. Sabri, Akshay Singh, and Ladan Tahvildari. 2015. Adoop: MapReduce for ad-hoc cloud computing. In *Proc. of the 25th CASCON Conf.* 26–34.

Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski, and Osmar Zaiane. 1996. DMQL: A data mining query language for relational databases. In *Proc. of the 1996 SiGMOD*. 27–34.

Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. 2011. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In *Proc. of the 2011 IEEE 27th Int'l Conf. on Data Engineering (ICDE'11).* 1199–1208.

Joseph M. Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib analytics library: *or MAD skills, the SQL*. *Proc. VLDB Endow.* 5, 12, 1700–1711.

Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. of the 8th USENIX Conf. on Networked Systems Design and Implementation (NSDI'11)*. 295–308.

IBM. 2012. IBM Global Technology Outlook 2012. Retrieved January 20th, 2015 from http://goo.gl/63zjGl.

Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.* 41, 3 (March 2007), 59–72.

Theodore Johnson, Laks V. S. Lakshmanan, and Raymond T. Ng. 2000. The 3W model and algebra for unified data mining. In *Proc. of the 26th Int'l Conf. on Very Large Data Bases (VLDB'00)*. 21–32.

Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. 2009. Towards optimizing Hadoop provisioning in the cloud. In *Proc. of the 2009 Conf. on Hot Topics in Cloud Computing (HotCloud'09).*

Verena Kantere and Maxim Filatov. 2015. A framework for Big Data Analytics. In *Proc. of the 8th International C∗ Conference on Computer Science & Software Engineering (C3S2E'15).* 125–132.

Shadi Khalifa, Patrick Martin, Dan Rope, Mike McRoberts, and Craig Statchuk. 2016. QDrill: Query-based distributed consumable analytics for Big Data. In *Proc. of the 2016 IEEE Int'l Congress on Big Data.*

Jorg-Uwe Kietz, Floarea Serban, and Abraham Bernstein. 2010. eProPlan: A tool to model automatic generation of data mining workflows. In *Proc. of the 3rd Planning to Learn Workshop (WS9) At the European Conf. on Artificial Intelligence (ECAI'10).* 15.

Herald Kllapi, Eva Sitaridi, Manolis Tsangaris, and Yannis Ioannidis. 2011. Schedule optimization for data processing flows on the cloud. In *Proc. of the Int'l Conf. on Management of Data (SIGMOD'11).* 289–300.

Konstantinos Kloudas, Margarida Mamede, Nuno Preguiça, and Rodrigo Rodrigues. 2015. Pixida: Optimizing data parallel jobs in wide-area data analytics. *VLDB Endow.* 9, 2 (October 2015), 72–83.

Aris-Kyriakos Koliopoulos, Paraskevas Yiapanis, Firat Tekiner, Goran Nenadic, and John Keane. 2015. A parallel distributed Weka framework for Big Data mining using Spark. In *Proc. of 2015 IEEE International Congress on Big Data.* 9–16.

S. B. Kotsiantis, D. Kanellopoulos, V. Karioti, and V. Tampakas. 2009. An ontology-based portal for credit risk analysis. In *Proc. of the 2nd IEEE Int'l Conf. on Comp. Sci. and Info. Tech. (ICCSIT'09)*. 165–169.

Tim Kraska, Ameet Talwalkar, John Duchi, Rean Griffith, Michael J. Franklin, and Michael Jordan. 2013. MLbase: A distributed machine learning system. In *Proc. of the 6th Biennial Conf. on Innovative Data Systems Research (CIDR'13)*. Asilomar, California.

Avinash Lakshman and Prashant Malik. 2010. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44, 2, 35–40.

Wang Lam, Lu Liu, Sts Prasad, Anand Rajaraman, Zoheb Vacheri, and AnHai Doan. 2012. Muppet: MapReduce-style processing of fast data. *Proc. VLDB Endow.* 5, 12 (August 2012), 1814–1825.

Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. 2012. Early accurate results for advanced analytics on MapReduce. *Proc. VLDB Endow.* 5, 10, 1028–1039.

Boduo Li, Yanlei Diao, and Prashant Shenoy. 2015. Supporting scalable analytics with latency constraints. *VLDB Endow.* 8, 11 (July 2015), 1166–1177.

Feng Li, Beng Chin Ooi, M. Tamer Özsu, and Sai Wu. 2014. Distributed data management using MapReduce. *ACM Comput. Surv.* 46, 3, Article 31, 42 pages.

Yuting Lin, Divyakant Agrawal, Chun Chen, Beng Chin Ooi, and Sai Wu. 2011. Llama: Leveraging columnar storage for scalable join processing in the MapReduce framework. In *Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of data (SIGMOD'11).* 961–972.

Xiufeng Liu, Nadeem Iftikhar, and Xike Xie. 2014. Survey of real-time processing systems for big data. In *Proc. of the 18th Int'l Database Engineering & Applications Symposium (IDEAS'14).* 356–361.

James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. 2011. Big Data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute. Retrieved January 20th, 2015 from http://bit.ly/McKinseyBigDataReport.

Ming Mao and Marty Humphrey. 2013. Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows. In *Proc. of the 27th IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS'13).* 67–78.

A. Martin, D. Maladhy, and V. P. Venkatesan. 2011. A framework for business intelligence application using ontological classification. *Int'l Journal of Engineering Science and Technology*. 3. 1213–1221.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow.*3, 1–2, 330–339.

Rosa Meo, Giuseppe Psaila, and Stefano Ceri. 1996. A New SQL-like Operator for Mining Association Rules. In *Proc. of the 22th Int'l Conf. on Very Large Data Bases (VLDB'96)*. 122–133.

Rizwan Mian, Patrick Martin, and Jose Luis Vazquez-Poletti. 2013. Provisioning data analytic workloads in a cloud. *Future Gener. Comput. Syst.* 29, 6, 1452–1458.

Sebastian Mittelstadt, Michael Behrisch, Stefan Weber, Tobias Schreck, Andreas Stoffel, Rene Pompl, Daniel Keim, Holger Last, and Leishi Zhang. 2012. Visual analytics for the big data era - A comparative review of state-of-the-art commercial systems. In *Proc. of the 2012 IEEE Conf. on Visual Analytics Science and Technology (VAST) (VAST'12)*. 173–182.

Katharina Morik and Martin Scholz. 2004. The MiningMart approach to knowledge discovery in databases. In *Intelligent Technologies for Information Analysis*, N. Zhong, and J. Liu (Eds.), Springer, 47–65.

Derek Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A timely dataflow system. In *Proc. of the 24th Symp. on Op. Sys. Principles (SOSP'13)*. 439–455.

Dana S. Nau, Stephen J. J. Smith, and Kutluhan Erol. 1998. Control strategies in HTN planning: Theory versus practice. In *Proc. of the Fifteenth National/Tenth Conf. on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI'98/IAAI'98)*. 1127–1133.

Dana Nau, Malik Ghallab, and Paolo Traverso. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. 2010. S4: Distributed Stream Computing Platform. In *Proc. of the IEEE Int'l Conf. on Data Mining Workshops (ICDMW'10)*. 170–177.

Objectivity, Inc. 2012. *InfiniteGraph: The Distributed Graph Database.* White Paper. Objectivity, Inc. Retrieved January 20th, 2015 from http://goo.gl/WPK308.

Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig latin: A not-so-foreign language for data processing. In *Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'08)*. 1099–1110.

Christopher Olston, Greg Chiou, Laukik Chitnis, Francis Liu, Yiping Han, Mattias Larsson, Andreas Neumann, Vellanki B. N. Rao, Vijayanand Sankarasubramanian, Siddharth Seth, Chao Tian, Topher Zi-Cornell, and Xiaodan Wang. 2011. Nova: Continuous Pig/Hadoop workflows. In *Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'11)*. 1081–1090.

Riccardo Ortale, Ettore Ritacco, Nikos Pelekis, Roberto Trasarti, G. Costa, F. Giannotti, G. Manco, C. Renso, and Y. Theodoridis. 2008. The DAEDALUS framework: Progressive querying and mining of movement data. In *Proc. of the 16th ACM SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems (GIS'08)*. Article 52, 4 pages.

Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. 2011. Purlieus: Locality-aware resource allocation for MapReduce in a cloud. In *Proc. of 2011 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC'11).* Article 58, 11 pages.

Panče Panov, Larisa Soldatova, and Sašo Džeroski. 2014. Ontology of core data mining entities. *Data Min. Knowl. Discov.* 28, 5–6, 1222–1265.

Jonas Partner. 2013. *Neo4j in Action*. O'Reilly Media, Inc.

Jaroslav Pokorny. 2011. NoSQL databases: A step to database scalability in web environment. In *Proc. of 13th Int'l Conf. on Info. Integration and Web-based Applications and Services (iiWAS'11).* 278–283.

Zoltan Prekopcsak, Gabor Makrai, Tamas Henk, and Csaba Gaspar-Papanek. 2011. Radoop: Analyzing Big Data with RapidMiner and Hadoop. In *Proc. of RCOMM 2011.*

Mohsen Rais-Ghasem, Robin Grosset, Martin Petitclerc, and Qing Wei. 2013. Towards semantic data analysis. In *Proc. of the 2013 CASCON Conf.*. 192–199.

Kavitha Ranganathan and Ian Foster. 2002. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proc. of the 11th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC'02).* 352–358.

Nick Russell, Wil van der Aalst, and Arthur ter Hofstede. 2006. Workflow exception patterns. In *Proc. of the 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'06)*. 288–302.

Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, and Jafar Adibi. 2001. A sequential pattern query language for supporting instant data mining for e-services. In *Proc. of the 27th VLDB Conf.* 653–656.

Filippo Sciarrone, Paolo Starace, and Tommaso Federici. 2009. A business intelligence process to support information retrieval in an ontology-based environment. In *Proc. of the 2009 9th Int'l Conf. on Intelligent Systems Design and Applications (ISDA'09)*. 896–901.

Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. 2013. A survey of intelligent assistants for data analysis. *ACM Comput. Surv.* 45, 3, Article 31, 35 pages.

Ricky J. Sethi, Yolanda Gil, Hyunjoon Jo, and Andrew Philpot. 2013. Large-scale multimedia content analysis using scientific workflows. In *Proc. of 21st ACM Int'l Conf. on Multimedia (MM'13)*. 813–822.

Avraham Shinnar, David Cunningham, Vijay Saraswat, and Benjamin Herta. 2012. M3R: Increased performance for in-memory Hadoop jobs. *Proc. VLDB Endow.* 5, 12, 1736–1747.

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop distributed file system. In *Proc. of the IEEE 26th Symp. on Mass Storage Sys. and Technologies (MSST'10)*. 1–10.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2008. *Operating System Concepts (8th ed.)*. Wiley Publishing.

Josenildo C. da Silva, Chris Giannella, Ruchita Bhargava, Hillol Kargupta, and Matthias Klusch. 2005. Distributed data mining and agents. *Eng. Appl. Artif. Intell.* 18, 7, 791–807.

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. 2007. Pellet: A practical OWL-DL reasoner. *Web Semant.* 5, 2, 51–53.

Sima Soltani. 2013. Case-based reasoning for diagnosis and solution planning. Queen's University Tech. Rep. No. 2013-611. Queen's University, Ontario, Canada.

Evan Sparks, Ameet Talwalkar, Virginia Smith, Xinghao Pan, Joseph Gonzales, Tim Kraska, Michael Jordan, and Michael J. Franklin. 2013. MLI: An API for Distributed Machine Learning. In *Proc. of the 2013 IEEE 13th Int'l Conf. on Data Mining (ICDM)*. 1187–1192.

Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. 2012. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX Conf. on File and Storage Technologies (FAST'12)*.

Ameet Talwalkara, Tim Kraskaa, Rean Griffith, John Duchi, Joseph Gonzalez, Denny Britz, Xinghao Pan, Virginia Smith, Evan Sparks, Andre Wibisono, Michael J. Franklin, and Michael I. Jordan. 2012. MLbase: A distributed machine learning wrapper. In *Proc. of Big Learning Workshop at NIPS*.

Zhaohui Tang, Jamie Maclennan, and Peter Pyungchul Kim. 2005. Building data mining solutions with OLE DB for DM and XML for analysis. *SIGMOD Rec.* 34, 2, 80–85.

Claudio Tesoriero. 2013. *Getting Started with OrientDB*. Packt Publishing Ltd.

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.* 2, 2, 1626–1629.

Dmitry Tsarkov and Ian Horrocks. 2006. FaCT++ description logic reasoner: System description. In *Proc. of the 3rd Int'l Joint Conf. on Automated Reasoning (IJCAR'06)*. 292–297.

David Turner, Michael Schroeck, and Rebecca Shockley. 2012. *Analytics: The real-world use of big data in financial services.* IBM Institute for Business, and Saïd Business School at the University of Oxford. Retrieved January 20th, 2015 from http://goo.gl/T6hD4k.

Wil van der Aalst and Kees van Hee. 2004. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA.

Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet another resource negotiator. In *Proc. of the 4th Symposium on Cloud Computing (SOCC'13)*. Article 5, 16 pages.

Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. 2004. Ontology based context modeling and reasoning using OWL. In *Proc. of the 2nd IEEE Annual Conf. on Pervasive Computing and Communications Workshops (PERCOMW'04)*. 18–22.

Qingsong Wei, Bharadwaj Veeravalli, Bozhao Gong, Lingfang Zeng, and Dan Feng. 2010. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In *Proc. of the 2010 IEEE Int'l Conf. on Cluster Computing (CLUSTER'10)*. 188–196.

Tom White. 2009. *Hadoop: The Definitive Guide (1st ed.)*. O'Reilly Media, Inc.

Alexander Wieder, Pramod Bhatotia, Ansley Post, and Rodrigo Rodrigues. 2012. Orchestrating the deployment of computations in the cloud with conductor. In *Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation (NSDI'12)*. 367–381.

Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2013. Shark: SQL and rich analytics at scale. In *Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'13)*. 13–24.

Donna Xu, Dongyao Wu, Xiwei Xu, Liming Zhu, and Len Bass. 2015. Making Real Time Data Analytics Available as a Service. In *Proc. of the 11th Int'l ACM SIGSOFT Conf. on Quality of Software Architectures (QoSA'15)*. 73–82.

Jia Yu and Rajkumar Buyya. 2005. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.* 34, 3, 44–49.

Makoto Yui and Isao Kojima. 2013. A database-Hadoop hybrid approach to scalable machine learning. In *Proc. of the 2013 IEEE Int'l Congress on Big Data (BIGDATACONGRESS'13)*. 1–8.

Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010a. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of the 5th European Conf. on Computer Systems (EuroSys'10)*. 265–278.

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010b. Spark: Cluster computing with working sets. In *Proc. of the 2nd USENIX Conf. on Hot Topics in Cloud Computing (HotCloud'10)*.

Yanfeng Zhang, Qinxin Gao, Lixin Gao, and Cuirong Wang. 2011a. iMapreduce: A distributed computing framework for iterative computation. In *Proc. of the 2011 IEEE Int'l Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW'11)*. 1112–1121.

Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. 2011b. PrIter: A distributed framework for prioritized iterative computations. In *Proc. of the 2nd ACM Symp. on Cloud Comp. (SOCC'11)*. Art. 13.

Jingren Zhou, Nicolas Bruno, Ming-Chuan Wu, Per-Ake Larson, Ronnie Chaiken, and Darren Shakib. 2012. SCOPE: Parallel databases meet MapReduce. *The VLDB Journal* 21, 0035, 611–636.

Received July 2015; revised May 2016; accepted May 2016