Big Data Analytic Approaches Classification

Yudith Cardinale[1] and Sonia Guehis[2,3] and Marta Rukoz[2,3]
[1]*Dept. de Computación, Universidad Simón Bolívar, Venezuela*
[2]*Université Paris Nanterre, 92001 Nanterre, France*
[3]*Université Paris Dauphine, PSL Research University, CNRS, UMR[7243], LAMSADE, 75016 Paris, France*

Keywords: Big Data Analytic, Analytic Models for Big Data, Analytical Data Management Applications.

Abstract: Analytical data management applications, affected by the explosion of the amount of generated data in the context of Big Data, are shifting away their analytical databases towards a vast landscape of architectural solutions combining storage techniques, programming models, languages, and tools. To support users in the hard task of deciding which Big Data solution is the most appropriate according to their specific requirements, we propose a generic architecture to classify analytical approaches. We also establish a classification of the existing query languages, based on the facilities provided to access the Big Data architectures. Moreover, to evaluate different solutions, we propose a set of criteria of comparison, such as OLAP support, scalability, and fault tolerance support. We classify different existing Big Data analytics solutions according to our proposed generic architecture and qualitatively evaluate them in terms of the criteria of comparison. We illustrate how our proposed generic architecture can be used to decide which Big Data analytic approach is suitable in the context of several use cases.

1 INTRODUCTION

The term Big Data has been coined for representing the challenge to support a continuous increase on the computational power that produces an overwhelming flow of data (Kune et al., 2016). Big Data databases have become important NoSQL data repositories (being non-relational, distributed, open-source, and horizontally scalable) in enterprises as the center for data analytics, while enterprise data warehouses (EDWs) continue to support critical business analytics. This scenario induced a paradigm shift in the large scale data processing and data mining, computing architecture, and data analysis mechanisms. This new paradigm has spurred the development of novel solutions from both industry (e.g., analysis of web-data, clickstream) and science (e.g., analysis of data produced by massive-scale simulations, sensor deployments, genome sequencers) (Chen et al., 2014; Philip Chen and Zhang, 2014).

In this sense, modern data analysis faces a confluence of growing challenges. First, data volumes are expanding dramatically, creating the need to scale out across clusters of hundreds of commodity machines. Second, different sources of data producers (i.e., relational databases, NoSQL) make the heterogeneity a strong characteristic that has to be overcome. Finally, despite these increases in scale and complexity, users still expect to be able to query data at interactive speeds. In this context, several enterprises such as Internet companies and Social Network associations have proposed their own analytical approaches, considering not only new programming models, but also medium and high-level tools, such as frameworks/systems and parallel database extensions.

This imposes the trend of the moment 'the Big Data boom': many models, frameworks, languages, and distributions are available. A non expert user who has to decide which analytical solution is the most appropriate for particular constraints in a Big Data context, is today lost, faced with a panoply of disparate and diverse solutions. Some of them, are well-known as MapReduce, but they have been overtaken by others more efficient and effective, like Spark system[1] and Nephele/PACT (Warneke and Kao, 2009).

We propose a generic architecture for analytical approaches which focuses on the data storage layer, the parallel programming model, the type of database, and the query language that can be used. These aspects represent the main features that allow to distinguish classical EDWs from analytical Big Data approaches. We aim in this paper to clarify the concepts

[1]http://spark.apache.org/

Cardinale, Y., Guehis, S. and Rukoz, M.
Big Data Analytic Approaches Classification.
DOI: 10.5220/0006437801510162
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 151-162
ISBN: 978-989-758-262-2

dealing with analytical approaches on Big Data and classify them in order to provide a global vision of the different existing flavors.

Based on our proposed architecture, the contribution of this paper is threefold: (i) establish a classification of the existing query languages, based on the facilities provided to access the big data architectures; this classification allows to evaluate the level of programming expertise needed to create new queries; (ii) propose a classification of the Big Data architectures, allowing to consider the underlined framework structure to implement the data warehouse; and (iii) propose a set of criteria of comparison, such as On-Line Analytical Processing (OLAP) support, scalability, and fault tolerance support. We classify different existing Big Data analytics solutions according to our proposed generic architecture and qualitatively evaluate them in terms of the criteria of comparison, as well as in terms of type of query language and parallel programming model that they use. We highlight their differences and underline their fair employment. We also illustrate how to use our proposed architecture in the context of several use cases. This work represents a first step towards the construction of a Decision Support System that will help non-expert users in selecting appropriate components.

Section 2 presents a review of Big Data programming models. Section 3 proposes a classification of Big Data query languages. Our proposed architecture and classification for analytic approaches are presented in Section 4. In Section 5, diverse solutions for analytic Big Data are studied. Section 6 presents the related work. We finally conclude in Section 7.

2 BIG DATA PROGRAMMING MODELS

The large-scale parallel Big Data processing scenario has brought new challenges in the programming models in order to process and analyze such huge amount of data. It is necessary a new model of cluster computing, able to adapt the constant data growing without affecting the performance, in which data-parallel computations can be executed on clusters of unreliable machines by systems that automatically provide locality-aware scheduling, fault tolerance, and load balancing. Classical parallel programming models, such as master-slave with Message Passing Interface (MPI) and multithreading with Open Multi-Processing (OpenMP), are not adequate for Big Data scenarios due to the high network bandwidth demanded to move data to processing nodes and the need to manually deal with fault tolerance and load balancing (Chen et al., 2014). However, inspired on them there have been deployments of cluster computing models, which aggregate computational power, main memory, and I/O bandwidth of shared-nothing commodity machines, combined with new parallel programming models (Leskovec et al., 2014; Pavlo et al., 2009). They improve the performance of NoSQL sources and reduce the performance gap to relational databases. The main difference between classical parallel models and the new ones is that, instead of moving data, the processing functions are taken to the data.

The most popular model of data parallel programming in the context of Big Data is MapReduce (Dean and Ghemawat, 2008). Along with it, Hadoop[2] is its most popular core framework implementation for carrying out analytic on Big Data. With simple distributed functions (based on Lisp primitives), the idea of this parallel programming model is to combine `map` and `reduce` operations with an associated implementation given by users and executed on a set of `n` nodes, each with data. Hadoop framework is in charge of splitting input data into small chunks, designated as key/value pairs, storing them on different compute nodes, and invoking `map` tasks to execute the user-defined functions (UDF); the `map` tasks generate intermediate key/value pairs according to the UDF. Subsequently, the framework initiates a Sort and Shuffle phase to combine all the intermediate values related to the same key and channelizes data to parallel executing `reduce` tasks for aggregation; the `reduce` tasks are applied to the set of data with a common key from the intermediate key/value pairs generated by `map`. MapReduce has inspired other models that extend it to cater different and specific needs of applications, as we explain in the following.

In (Battré et al., 2010), it is proposed another programming model, which extends the MapReduce model with Parallelization Contracts (PACT) and additional second-order functions. Nephele framework (Warneke and Kao, 2009), is a distributed execution engine that supports PACT programs execution, which are represented by Directed Acyclic Graphs (DAGs). Edges in the DAG represent communication channels that transfer data between different subprograms. Vertices of the DAG are sequential executable programs that process the data that they get from input channels and write it to output channels. The main differences between MapReduce and PACT programming models are: (i) besides map and reduce, PACT allows additional functions that fit more complex data processing tasks, which are not naturally and efficiently expressible as map or reduce functions,

[2]http://hadoop.apache.org

as they occur in fields like relational query processing or data mining; (ii) while MapReduce systems tie the programming model and the execution model (conducting sub-optimal solutions), with PACT, it is possible to generate several parallelization strategies, hence offering optimization opportunities; and (iii) MapReduce loses all semantic information from the application, except the information that a function is either a map or a reduce, while the PACT model preserves more semantic information through a larger set of functions and annotations.

The Spark system (Zaharia et al., 2012; Zaharia et al., 2010) proposes a programming model that unlike acyclic data flows problems (such those treated with MapReduce and Nephele/PACT), it focuses on applications that reuse a working set of data across multiple parallel operations. Spark provides two main abstractions for parallel programming: Resilient Distributed Datasets (RDDs) and parallel operations on these datasets (invoked by passing a function to apply on a dataset). The RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Therefore, users can explicitly cache a RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. RDDs do not need to be materialized at all times. Beside programming model, which allow iterative algorithms (i.e., cyclic data flows), Spark overcomes MapReduce and Nephele/PACT because it handles most of its operations "in memory", copying data sets from distributed physical storage into far faster logical RAM memory. By contrast, MapReduce writes and reads from hard drives.

In general, the underlying system that implements the programming model, also manages the automatic scheduling, load balancing, and fault tolerance without user intervention. While programming models are most focused on supporting the implementation of complex parallel algorithms and on the efficient distribution of tasks, there exists another axis related to querying and analyzing that huge amount of data. Several languages have been proposed with the intention of improving the programming efficiency for task description and dataset analysis. Section 3 presents a classification of the different query languages that have been proposed in the context of Big Data.

3 CLASSIFICATION OF QUERY LANGUAGES

The wide diversification of data store interfaces has led the loss of a common programming paradigm for querying multistore and heterogeneous repositories and has created the need for a new generation of special federation between Hadoop-like Big Data platforms, NoSQL data stores, and EDWs. Mostly, the idea of all query languages is to execute different queries to multiple, heterogeneous data stores through a single query engine. We classify these query languages in Procedural Languages, Language Extensions, and SQL-like Languages (Chattopadhyay et al., 2011).

Procedural Languages are built on top of frameworks that do not provide transparent functions (e.g., `map` and `reduce`) and cannot cover all the common operations. Therefore, programmers have to spend time on programming the basic functions, which are typically hard to maintain and reuse. Hence, simpler procedural language have been proposed. The most popular of this kind of languages are Sawzall of Google (Pike et al., 2005), PIG Latin of Yahoo! (Olston et al., 2008), and more recently Jaql (Beyer et al., 2011). They are domain-specific, statically-typed, and script-like programming language used on top of MapReduce. These procedural languages have limited optimizations built in and are more suitable for reasonably experienced analysts, who are comfortable with a procedural programming style, but need the ability to iterate quickly over massive volumes of data.

Language Extensions are proposed to provide simple operations for parallel and pipeline computations, as extensions of classical languages, usually with special purpose optimizers, to be used on top of the core frameworks. The most representatives in this category are FlumeJava proposed by Google (Chambers et al., 2010) and LINQ (Language INtegrated Query) (Meijer et al., 2006). FlumeJava is a Java library for developing and running data-parallel pipelines on top of MapReduce, as a regular single-process Java program. LINQ embeds a statically typed query language in a host programming language, such as C#, providing SQL-like construct and allowing a hybrid of declarative and imperative programming.

In the **Declarative Query Languages** category, we classify those languages also built on top of the core frameworks with intermediate to advanced optimizations, which compile declarative queries into MapReduce-style jobs. The most popular focused on this use-case are HiveQL of Facebook (Thusoo et al., 2010), Tenzing of Google (Chattopadhyay et al., 2011), SCOPE of Microsoft (Zhou et al., 2012; Chaiken et al., 2008), Spark SQL (Xin et al., 2013) on top of Spark framework, and Cheetah (Chen, 2010). These languages are suitable for reporting and analysis functionalities. However, since they are built on, it is hard to achieve interactive query response times,

with them. Besides, they can be considered unnatural and restrictive by programmers who prefer writing imperative scripts or code to perform data analysis.

4 GENERIC ARCHITECTURE FOR ANALYTICAL APPROACHES

Basically, a Big Data analytical approach architecture is composed of four main components: Data Sources, ETL (Extract, Transform, and Load) module, Analytic Processing module, and Analytical Requests module. A generic representation of this architecture is shown in Fig. 1 and described as follows:

Data Sources: They constitute the inputs of an analytical system and often consider heterogeneous (i.e., structured, semi-structured, and unstructured) and sparse data (e.g., measures of sensors, log files streaming, mails and documents, social media content, transactional data, XML files).

ETL Module: It has as main functionality retrieving data from sources, transforming data for integration constraints, and loading data to the receiving end. Other functionalities include data cleaning and integration, through tools like Kettle[3], Talend[4], and Scribe (Lee et al., 2012).

Analytic Processing Module: It constitutes the hard core studied within this paper and can be defined as the chosen approach for the analysis phase implementation in terms of data storage (i.e., disk or memory) and the data model used (e.g., relational, NoSQL). Then, several criteria, such as parallel programming model (e.g., MapReduce, PACT) and scalability allow comparing approaches in the same class.

Analytical Requests: This module considers Visualization, Reporting, and BI (Business Intelligent) Dashboards Generation functionalities. Based on the analytic data and through different query languages (e.g., procedural, SQL-like), this module can generate outputs in several formats with pictorial or graphical representation (dashboards, graphics, reports, etc), which enables decision makers to see analytics presented visually. We focus in the Analytics Processing module to classify the different Big Data analytic approaches.

Analytic Processing Classification

Concerning the Analytic Processing component, several architectures in Big Data context exist. We classify the approaches according to their data storage support: (i) those based on disk data storage, which we divide into two groups according the data model, NoSQL databases (class *A* in Fig. 1) and relational databases (class *B* in Fig. 1); and (ii) those that support in-memory databases (class *C* in Fig. 1), in which the data is kept in RAM reducing the I/O operations. We detail each class as follows:

A- NoSQL based Architecture: It is based on NoSQL databases as storage model, relying or not on DFS (Distributed File System). It relies also on a parallel-processing models such as MapReduce, in which the processing workload is spread across many CPUs on commodity compute nodes. The data is partitioned among the compute nodes at run time and the underlined framework handles inter-machine communication and machine failures. The most famous embodiment of a MapReduce cluster is Hadoop. It was designed to run on many machines that do not share memory or disks (the shared-nothing model). This kind of architecture is designed by *A* in Fig. 1.

B- Relational Parallel Database based Architecture: It is based, as classical databases, on relational tables stored on disk. It implements features like indexing, compression, materialized views, I/O sharing, caching results. Among these architectures there are: shared-nothing (multiple autonomous nodes, each owning its own persistent storage devices and running separate copies of the DBMS), shared-memory or shared anything (a global memory address space is shared and one DBMS is present), and shared-disk (based on multiple loosely coupled processing nodes similar to shared-nothing, but a global disk subsystem is accessible to the DBMS of any processing node). However, most of current analytical DBMS systems deploy a shared-nothing architecture parallel database. In this architecture, the analytical solution is based on the conjunction of the parallel (sharing-nothing) databases with a parallel programming model. We can see this type of architecture designed by *B* in Fig. 1.

C- The in-memory Structures based Architecture: This type of architecture attempts to satisfy commercial demand on real-time and scalable data warehousing criteria, based on a distributed multidimensional in-memory DBMS and the support of OLAP operations over a large volume of data for some of the systems. In-memory analytics offer new possibilities resulting from the huge performance gained by the in-memory placement. Those possibilities exceed the query speed, even more important, in simplifying models for analyz-

[3]http://kettle.pentaho.org
[4]http://www.talend.com

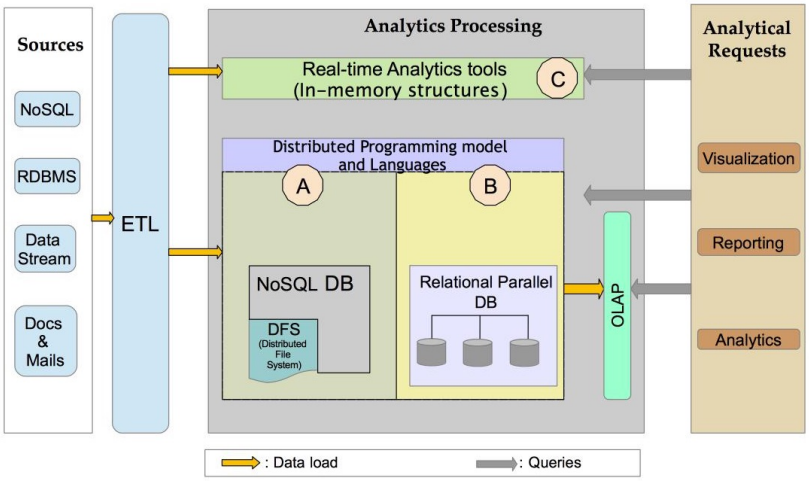

Figure 1: Generic Architecture for a Big Data Analytical Approach.

ing data, making more interactive the interfaces, and lowing overall solution latency. This kind of architecture is described by *C* in Fig. 1.

Criteria of Comparison

To compare the different approaches on each class, we establish a set of criteria related to the interaction facilities (e.g., OLAP-like support, query languages used, Cloud services support) , as well as implementation criteria related to performance aspects (e.g., scalability, programming model, fault tolerance support).

OLAP Support: Some analytical solutions integrate OLAP in their system allowing OLAP operators for mining and analyzing data directly over the data warehouse support (i.e., without extraction from it). Since most solutions classified in our *C* class are characterized by in-memory databases support, they satisfy this criterion. However solutions implementing the architectures *A* or *B* can optionally consider an OLAP phase to support queries from the Analytical Requests module (for Visualization, Reporting, and Analysis). Thus, this property is fixed to `integrated` when it is the case or `not-integrated` otherwise.

Query Languages: This criterion specifies the language on which the user relies on for querying the system. It could be `procedural`, `language extension`, or `declarative` (see Section 3).

Cloud Services Support: The three types of architectures described above can be offered as a product (with `no` cloud support) or as a service deployed in the cloud. In the case of cloud support, it can be done partially, such as Infrastructure as a Service (`IaaS`, which provides the hardware support and basic storage and computing services) and Data Warehouse as a Service (`DWaaS`), or as a whole analytical solution in the cloud, i.e., Platform as a Service (`PaaS`), which provides the whole computing platform to develop Big Data analytical applications.

Scalability: This criterion measures the availability on demand of compute and storage capacity, according to the volume of data and business needs. For this property we fix the values `large-scale` or `medium-scale` (for the case where systems do not scale to thousands of nodes).

Fault Tolerance: This criterion establishes the capability of the system of providing recovery from failures in a transparent way. In the context of analytical workloads, we consider the following fault tolerance techniques: the classical `Log-based` (i.e., write-ahead log), which is used for almost every database management system and `Hadoop-based` techniques, which provides data replication for fault tolerance. In the latter, we consider two types of recovery techniques: `Hadoop-file`, where the data replication recovery control is implemented at the HDFS level (i.e., on the worked nodes allowing re-execute only the part of systems containing the failed data) and `Hadoop-programming`, where the recovery control is implemented at the model program level.

Programming Model: It precises for each item which programming model is adopted. We consider `MapReduce`, `PACT`, `RDD-Spark` models and we mention `ad-hoc` for specific implementations.

5 WHICH APPROACH SHOULD I ADOPT?

In this section we describe some well-known frameworks/systems for analytic processing on the three classes of architectures.

NoSQL based Architectures

Several tools which are NoSQL based architecture exist. We cite as examples:

- **Avatara:** It leverages an offline elastic computing infrastructure, such as Hadoop, to precompute its cubes and perform joins outside of the serving system (Wu et al., 2012). These cubes are bulk loaded into a serving system periodically (e.g., every couple of hours). It uses Hadoop as its batch computing infrastructure and Voldemort (Sumbaly et al., 2012), a key-value storage, as its cube analytical service. The Hadoop batch engine can handle terabytes of input data with a turnaround time of hours. While Voldemort, as the key-value store behind the query engine, responds to client queries in real-time. Avatara works well while cubes are small, for far bigger cubes, there will be significant network overhead in moving the cubes from Voldemort for each query.
- **Apache Kylin**[5]: It is a distributed analytics engine supporting definition of cubes, dimensions, and metrics. It provides SQL interface and OLAP capability based on Hadoop Ecosystem and HDFS environment. It is based on Hive data warehouse to answer mid-latency analytic queries and on materialized OLAP views stored on a HBase cluster to answer low-latency queries.
- **Hive**: It is an open-source project that aims at providing data warehouse solutions and has been built by the Facebook Data Infrastructure Team on top of the Hadoop environment. It supports ad-hoc queries with a SQL-like query language called HiveQL (Thusoo et al., 2009; Thusoo et al., 2010). These queries are compiled into MapReduce jobs that are executed using Hadoop. The HiveQL includes its own system type and Data Definition Language (DDL) which can be used to create, drop, and alter tables in a database. It also contains a system catalog which stores metadata about the underlying table, containing schema information and statistics, much like DBMS engines. Hive currently provides only a simple, naive rule-based optimizer.
- **Cloudera Impala**[6]: It is a parallel query engine that runs on Apache Hadoop. It enables users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without requiring data movement or transformation. Impala uses the same file and data formats, metadata, security, and resource management frameworks used by MapReduce, Apache Hive, Apache Pig, and other Hadoop software. Impala allows to perform analytics on data stored in Hadoop via SQL or business intelligence tools. It gives a good performance while retaining a familiar user experience. By using Impala, a large-scale data processing and interactive queries can be done on the same system using the same data and metadata without the need to migrate data sets into specialized systems or proprietary formats.
- **Mesa:** Google Mesa (Gupta et al., 2014) leverages common Google infrastructure and services, such as Colossus (the successor of Google File System) (Ghemawat et al., 2003), BigTable (Chang et al., 2008), and MapReduce. To achieve storage scalability and availability, data is horizontally partitioned and replicated. Updates may be applied at the granularity of a single table or across many tables. To ensure consistent and repeatable queries during updates, the underlying data is multi-versioned. To achieve update scalability, data updates are batched, assigned a new version number, and periodically (e.g., every few minutes) incorporated into Mesa. To achieve update consistency across multiple data centers, Mesa uses a distributed synchronization protocol based on Paxos (Lamport, 2001).

Discussion and Comparison: NoSQL-based systems have shown to have superior performance than other systems (such as parallel databases) in minimizing the amount of work that is lost when a hardware failure occurs (Pavlo et al., 2009; Stonebraker et al., 2010). In addition, Hadoop (which is the open source implementations of MapReduce) represents a very cheap solution. However, for certain cases, MapReduce is not a suitable choice, specially when intermediate processes need to interact, when lot of data is required in a processing, and in real time scenarios, in which other architectures are more appropriate.

Relational Parallel Databases based Architectures

Several projects aim to provide low-latency engines, whose architectures resemble shared-nothing parallel databases, such as projects which embed MapReduce and related concepts into traditional parallel DBMSs. We cite as examples the following projects:

- **Google's Dremel:** It is a system that supports interactive analysis of very large datasets over shared clusters of commodity machines (Melnik et al., 2011). Dremel is based on a nested

[5]http://kylin.apache.org

[6]https://www.cloudera.com/products/open-source/apache-hadoop/impala.html

column-oriented storage that is designed to complement MapReduce. It is used in conjunction with MapReduce to analyze outputs of MapReduce pipelines or rapidly prototype larger computations. It has the capability of running aggregation queries over trillion-row tables in seconds by combining multi-level execution trees and columnar data layout. The system scales to thousands of CPUs and petabytes of data and has thousands of users at Google.

- **PowerDrill**: It is a system which combines the advantages of columnar data layout with other known techniques (such as using composite range partitions) and extensive algorithmic engineering on key data structures (Hall et al., 2012). Compared to Google's Dremel that uses streaming from DFS, PowerDrill relies on having as much data in memory as possible. Consequently, PowerDrill is faster than Dremel, but it supports only a limited set of selected data sources, while Dremel supports thousands of different data sets. PowerDrill uses two dictionaries as basic data structures for representing a data column. Since it relies on memory storage, several optimizations are proposed to keep small the memory footprint of these structures. PowerDrill is constrained by the available memory for maintaining the necessary data structures.
- **Amazon Redshift:** It is an SQL-compliant, massively-parallel, query processing, and database management system designed to support analytics workload (Gupta et al., 2015). Redshift has a query execution engine based on ParAccel[7], a parallel relational database system using a shared-nothing architecture with a columnar orientation, adaptive compression, memory-centric design. However, it is mostly PostgreSQL-like, which means it has rich connectivity via both JDBC and ODBC and hence Business Intelligence tools. Based on EC2, Amazon Redshift solution competes with traditional data warehouse solutions by offering DWaaS, that it is translated on easy deployment and hardware procurement, the automated patching provisioning, scaling, backup, and security.
- **Teradata:** It is a system which tightly integrates Hadoop and a parallel data warehouse, allowing a query to access data in both stores by moving (or storing) data (i.e., the working set of a query) between each store as needed (Xu et al., 2010). These approaches are based on having corresponding data partitions in each store co-located on the same physical node. The purpose of co-locating data partitions is to reduce network transfers and improve locality for data access and loading between the stores. However, this requires a mechanism whereby each system is aware of the other systems partitioning strategy; the partitioning is fixed and determined up-front. Even though these projects offer solutions by providing a simple SQL query interface and hiding the complexity of the physical cluster, they can be prohibitively expensive at web scale.
- **AsterData System:** It is a nCluster shared-nothing relational database[8], that uses SQL/MapReduce (SQL/MR) UDF framework, which is designed to facilitate parallel computation of procedural functions across hundreds of servers working together as a single relational database (Friedman et al., 2009). The framework leverages ideas from the MapReduce programming paradigm to provide users with a straightforward API through which they can implement a UDF in the language of their choice. Moreover, it allows maximum flexibility, since the output schema of the UDF is specified by the function itself at query plan-time.
- **Microsoft Polybase:** It is a feature of Microsoft SQL Server Parallel Data Warehouse (PDW), which allows directly reading HDFS data into databases using SQL (DeWitt et al., 2013). It allows HDFS data to be referenced through external PDW tables and joined with native PDW tables using SQL queries. Polybase employs a split query processing paradigm in which SQL operators on HDFS-resident data are translated into MapReduce jobs by the PDW query optimizer and then executed on the Hadoop cluster.
- **Vertica:** It is a distributed relational DBMS that commercializes the ideas of the C-Store project (Lamb et al., 2012). The Vertica model uses data as tables of columns (attributes), though the data is not physically arranged in this manner. It supports the full range of standard INSERT, UPDATE, DELETE constructs for logically inserting and modifying data as well as a bulk loader and full SQL support for querying. Vertica supports both dynamic updates and real-time querying of transactional data.
- **Microsoft Azure[9]:** It is a solution provided by Microsoft for developing scalable applications for the cloud. It uses Windows Azure Hypervisor (WAH) as the underlying cloud infrastructure and

[7]http://www.actian.com

[8]http://www.asterdata.com/

[9]http://www.microsoft.com/azure

.NET as the application container. It also offers services including Binary Large OBject (BLOB) storage and SQL service based on SQL Azure relational storage. The analytics related services provide distributed analytics and storage, as well as real-time analytics, big data analytics, data lakes, machine learning, and data warehousing.

Discussion and Comparison: Although these parallel database systems serve some of the Big Data analysis needs and are highly optimized for storing and querying relational data, they are expensive, difficult to administer, and lack fault-tolerance for long-running queries (Pavlo et al., 2009; Stonebraker et al., 2010). None of these systems have been designed to manage replicated data across multiple datacenters (they are mostly centralized and do not scale to thousands of nodes). While more nodes are added into the system, hardware failures become more common and frequent, causing a limited scalability. In contrast, most relational databases assume that hardware failure is a rare event. They employ a coarser-grained recovery model, where an entire query has to be resubmitted if a machine fails. This works well for short queries where a retry is inexpensive, but faces significant challenges in long queries as clusters scale up (Abouzeid et al., 2009). Hence, these systems do not support fine-grained fault tolerance. Also, it is difficult for these systems to process non-relational data.

In-memory Structure based Architectures

Some type of architectures are designed to support complex, multi-dimensional, and multi-level on-line analysis of large volumes of data stored in RAM. We present the following ones:

- **Cubrick:** It is a new architecture that enables real-time data analysis of large dynamic datasets (Pedro et al., 2015). It is an in-memory distributed multidimensional database that can execute OLAP operations such as *slice* and *dice*, *roll up* and *drill down* over terabytes of data. Data in a Cubrick cube is range partitioned in every dimension, composing a set of data containers called *bricks* where data is stored sparsely and in an unordered and append-only fashion, providing high data ingestion ratios and indexed access through every dimension. Unlike traditional data cubes, Cubrick does not rely on any pre-calculations, rather it distributes the data and executes queries on-the-fly leveraging MPP architectures. Cubrick is implemented at Facebook from the ground up. A good result is obtained in a first Cubrick deployment inside Facebook.
- **Druid:** It is a distributed and column-oriented database designed for efficiently support OLAP queries over real-time data (Yang et al., 2014). It supports streaming data ingestion and fault-tolerance. Druid cluster is composed of different types of nodes, each type having a specific role (real-time node, historical nodes, broker nodes, and coordinator nodes) that operate independently. Real-time nodes ingest and query event streams using an in-memory index to buffer events. The in-memory index is regularly persisted to disk. Persisted indexes are then merged together periodically before getting handed off. Both, in-memory and persisted indexes are hit by the queries. Historical nodes are the main workers of a Druid cluster, they load and serve the immutable blocks of data (segments) created by the real-time nodes. Broker nodes route queries to real-time and historical nodes and merge partial results returned from the two types of nodes. Coordinator nodes are essentially responsible for management and distribution of data on historical nodes: loading new data, dropping outdated data, replicating. A query API is provided in Druid, with a new proposed query language based on JSON Objects in input and output.
- **SAP HANA**[10]**:** SAP HANA is an in-memory and column-oriented relational DBMS. It provides both transactional and real-time analytics processing on a single system with one copy of the data. The SAP HANA in-memory DBMS provides interfaces to relational data (through SQL and Multidimensional expressions, or MDX), as well as interfaces to column-based relational DBMS, which allows to support geospatial, graph, streaming, and textual/unstructured data. The approach offers multiple in-memory stores: row-based, column-wise, and also object graph store.
- **Shark**: It is a data analysis system that leverages distributed shared memory to support data analytics at scale and focuses on in-memory processing of analysis queries (Xin et al., 2013). It supports both SQL query processing and machine learning functions. It is built on the distributed shared memory abstraction (RDD) implementation from Spark, which provides efficient mechanisms for fault recovery. If one node fails, Shark generates deterministic operations necessary for building lost data partitions in the other nodes, parallelizing the process across the cluster. Shark is

[10]https://help.sap.com/viewer/product/SAP_HANA_PLATFORM/2.0.00/en-US

compatible with Apache Hive, thus it can be used to query an existing Hive data warehouse and to query data in systems that support the Hadoop storage API, including HDFS and Amazon S3.

- **Nanocubes**[11]**:** It is an in-memory data cube engine offering efficient storage and querying over spatio-temporal multidimensional datasets. It enables real-time exploratory visualization of those datasets. The approach is based on the construction of a data cube (i.e., a nanocube), which fits in a modern laptop's main memory, and then computes queries over that nanocube using OLAP operations like *roll up* and *drill down*.
- **Stratosphere:** It is a data analytics stack allowing *in situ* data analysis by connecting to external data sources going from structured relational data to unstructured text data and semi-structured data (Alexandrov et al., 2014). It executes the analytic process directly from the data source (i.e., data is not stored before the analytical processing), by converting data to binary formats after the initial scans. Stratosphere provides a support for iterative programs that make repeated passes over a data set updating a model until they converge to a solution. It contains an execution engine which includes query processing algorithms in external memory and the PACT programming model together with its associated framework Nephele. Through Nephele/PACT, it is possible to treat low level programming abstractions consisting of a set of parallelization primitives and schema-less data interpreted by the UDFs written in Java.

Discussion and Comparison: Druid and SAP HANA are both in-memory column-oriented relational DBMS. Additionally SAP-HANA simultaneously allows transactional and real-time analytics processing. Cubricks, Druid, Nanocubes, and Stratosphere have in common the support for OLAP operations. Cubricks and Stratosphere are distinguished by the execution on the fly, without pre-calculations, supporting ad-hoc query, which is not provided for most current OLAP DBMSs and multidimensional indexing techniques.

General Discussion

We summarize our classification in Table 1 and compare the revised approaches according to the established criteria: used query language, scalability, OLAP support, fault tolerance support, cloud support, and programming model. Generally, solutions in the class *A* offer materialized views stored on NoSQL databases and updated in short period of times to offer OLAP facilities, meanwhile for in-memory structures based architectures (class *C*), the OLAP facility is embedded in the system. Approaches based on parallel databases do not scale to huge amount of nodes, however they have generally, best performance than the other architectures. Most languages used are declarative SQL-compliant. Regarding fault tolerance, most Big Data approaches leverage on Hadoop facilities, instead of the classical log-based mechanism. Currently, Big Data analytical solutions trend to move to the cloud, either partially or as whole solution (PaaS). Concerning the parallel programming model, most solutions described in this work implement MapReduce programming model native with Hadoop as the underline support or were extended in order to consider this programming model. Other solutions have extended the MapReduce programming model such Cubrick and Shark that also support Spark modeling with RDD.

We are aware of the existence of other data models, such graph databases, that are normally used together with the ones we consider in our architecture. Those complementary data models are currently treated by some analytic approaches. In the future, we will extend our architecture to explicitly integrate other data models and refine our classification.

Three Use Cases for Big Data Analytics

In order to show how our generic architecture can be used to decide which approach and architecture are suitable, in this section we illustrate the analysis of three use cases. They are uses cases that can be scaled into the petabyte range and beyond with appropriate business assumptions, hence they cannot be satisfied with scalar numeric data and cannot be properly analyzed by simple SQL statements (Kimball, 2012).

- **In-flight Aircraft Status.** This use case is representative of many other use cases that are emerging as responses of the increased introduction of sensor technology everywhere. In aircraft systems, in-flight status of hundreds of variables on engines, fuel systems, hydraulics, and electrical systems are measured and transmitted every few milliseconds, as well as information regarding the passengers comfort. All this information can be analyzed in some future time (e.g., for preventive maintenance planning, to improve personalized services for passengers) and also be analyzed in real time to drive real-time adaptive control, fuel usage, part failure prediction, and pilot no-

[11] http://nanocubes.net

tification (e.g., smarter flights)[12]. In this scenario there are several sources of data to consider, there is a need for predictive and on-line queries, scalability is a must, and fault tolerance has to be provided at fine grain. Hence, a solution of class *A* with OLAP support addresses these requirements.

- **Genomics Analysis.** This use case involves the industry that is being formed to address genomics analysis broadly to identify, measure, or compare genomic features such as DNA sequence, structural variation, gene expression, or regulatory and functional element annotations. This allows to conduct various analyses including family-based analysis or case-control analysis. This kind of use case manages mostly structured data and it demands high performance computing[13]. Thus, a solution with large-scale and procedural query language of class *B* addresses its requirements.
- **Location and Proximity Tracking.** GPS location tracking and frequent updates are necessary in operational applications, security analysis, navigation, and social media contexts[14]. This use case represents applications that precise location tracking, which produce a huge amount of data about other locations nearby the GPS measurement. These other locations may represent opportunities for sales or services in real-time. In this case, an in-memory based architecture (class *C*) or a solution of class *A* with OLAP support are suitable.

6 RELATED WORK

Some recent studies present comparative analyses of some aspects in the context of Big Data (Madhuri and Sowjanya, 2016; Gandhi and Kumbharana, 2013; Pkknen and Pakkala, 2015; Khalifa et al., 2016). Authors in (Madhuri and Sowjanya, 2016) and (Gandhi and Kumbharana, 2013) present a comparative study between two cloud architectures: Microsoft Azure and Amazon AWS Cloud services. They consider price, administration, support, and specification criteria to compare them. The work presented in (Pkknen and Pakkala, 2015) proposes a classification of technologies, products, and services for Big Data Systems. Authors first present a reference architecture representing data stores, functionality, and data flows. After that, for each considered solution, they present a mapping between the solution's use case and the reference architecture. The paper studies data analytics infrastructures at Facebook, LinkedIn, Twitter, Netflix, and also consider a high performance streaming analytics platform, BlockMon. However, these works are specific systems-concentrated and do not provide a general vision of existing approaches.

The work in (Khalifa et al., 2016) is most related to our paper. It aims to define the six pillars (namely Storage, Processing, Orchestration, Assistance, Interfacing, and Development) on which Big Data analytics ecosystem is built. For each pillar, different approaches and popular systems implementing them are detailed. Based on that, a set of personalized ecosystem recommendations is presented for different business use cases. Even though authors argue that the paper assists practitioners in building more optimized ecosystems, we think that proposing solutions for each pillar does not necessarily imply a global coherent Big Data analytic system. Even though our proposed classification considers building blocks on which Big Data analytics Ecosystem is built (presented as layers in the generic architecture), our aim is to provide the whole view of the analytical approaches. In that way, practitioners can determine which analytical solution, instead of a set of building blocks, is the most appropriate according to their particular needs.

7 CONCLUSIONS

In this paper, we present a generic architecture for Big Data analytical approaches allowing to classify them according to the data storage layer, the distributed parallel model, and the type of database used. We focus on the three most recently used architectures, as far as we know: MapReduce-based, Parallel databases based, and in-memory based architectures. We compare several implementations based on criteria such as: OLAP support, scalability as the capacity to adapt the volume of data and business needs, type of language supported, and fault tolerance in terms of the need of restarting a query if one of the node involved in the query processing fails. Besides the proposed classification, the main aim of this paper is to clarify the concepts dealing with analytical approaches on Big Data, thus allowing non-expert users to decide which technology is better depending on their requirements and on the type of workload. This work represents a first step towards the implementation of a Decision Support System, which will recommend

[12] https://www.digitaldoughnut.com/articles/2017/march/how-airlines-are-using-big-data

[13] http://www.nature.com/news/genome-researchers-raise-alarm-over-big-data-1.17912

[14] http://www.techrepublic.com/article/gps-serves-a-pivotal-role-in-big-data-stickiness/

Table 1: Comparative table of the studied analytic Big Data approaches.

Architecture	Tool	Query Language	Scalability	OLAP	Fault Tolerance	On the cloud	Programming model
NoSQL based Architectures (Class *A*)	Avatara	Language extensions	Large-scale	integrated	Hadoop-programming	no	MapReduce
	Hive	Declarative (HiveQL)	Large-scale	not-integrated	Hadoop-programming	DWaaS	MapReduce
	Cloudera Impala	Declarative (HiveQL) and Procedural (PIG Latin)	Large-scale	not-integrated	Hadoop-programming	DWaaS	MapReduce
	Apache Kylin	Language extensions	Large-scale	integrated	Hadoop-programming	no	MapReduce
	Mesa	Language extension	Large-scale	not-integrated	Hadoop-file	DWaaS	MapReduce
Relational database based eArchitectures (Class *B*)	PowerDrill	Procedural	Medium-scale	not-integrated	Log-based	no	ad-hoc
	Teradata	Procedural	Medium-scale	not-integrated	Hadoop-file	DWaaS	MapReduce
	Microsoft Azure	Declarative (SQL Azure)	Large-scale	not-integrated	Hadoop-file and Hadoop-programming	IaaS PaaS	MapReduce
	AsterData	Declarative	Medium-scale	not-integrated	Hadoop-file	IaaS	MapReduce
	Google's Dremel	Procedural	Large-scale	not-integrated	Log-based	IaaS (BigQuery)	MapReduce
	Amazon Redshift	Declarative	Large-scale	not integrated	Hadoop-file	PaaS, DWaaS	MapReduce
	Vertica	Declarative	Medium-scale	not-integrated	Hadoop-file	no	MapReduce
in-memory structures based Architectures (Class *C*)	Cubrick	Declarative	Large-scale	integrated	Log-based	no	ad-hoc
	Druid	Procedural (based on JSON)	Large-scale	integrated	Log-based	no	ad-hoc
	SAP HANA	Declarative	Large-scale	not-integrated	Log-based	PaaS	ad-hoc
	Shark	Declarative (HiveQL)	Large-scale	not-integrated	Hadoop-file (RDD properties on Spark)	no	RDD-Spark
	Nanocubes	Declarative	Large-scale	integrated	Log-based	no	ad-hoc
	Stratosphere	Declarative (Meteor)	Large-scale	not-integrated	Log-based	IaaS	Nephele/PACT

the appropriate analytical approach according to the user needs. We will also extend our classification by considering other architectures and data models (e.g., graph databases, document databases).

REFERENCES

Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., and Rasin, A. (2009). HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *VLDB Endow.*, 2(1):922–933.

Alexandrov, A., Bergmann, R., Ewen, S., Freytag, J.-C., Hueske, F., Heise, A., Kao, O., Leich, M., Leser, U., Markl, V., Naumann, F., Peters, M., Rheinländer, A., Sax, M. J., Schelter, S., Höger, M., Tzoumas, K., and Warneke, D. (2014). The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964.

Battré, D., Ewen, S., Hueske, F., Kao, O., Markl, V., and Warneke, D. (2010). Nephele/PACTs: A Programming Model and Execution Framework for Web-scale Analytical Processing. In *Proc. of the 1st ACM Symposium on Cloud Computing*, pages 119–130.

Beyer, K. S., Ercegovac, V., Gemulla, R., Balmin, A., Eltabakh, M. Y., Kanne, C., and et al. (2011). Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283.

Chaiken, R., Jenkins, B., Larson, P.-A., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. (2008). SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. *VLDB Endow.*, 1(2):1265–1276.

Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., and Weizenbaum, N. (2010). Flumejava: Easy, efficient data-parallel pipelines. *SIGPLAN Not.*, 45(6):363–375.

Chang, F., Dean, J., Ghemawat, S., and et. al (2008). Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26.

Chattopadhyay, B., Lin, L., Liu, W., Mittal, S., Aragonda, P., Lychagina, V., Kwon, Y., and Wong, M. (2011). Tenzing: A SQL Implementation On The MapReduce Framework. *PVLDB*, 4(12):1318–1327.

Chen, M., Mao, S., and Liu, Y. (2014). Big data: A survey. *Mob. Netw. Appl.*, 19(2):171–209.

Chen, S. (2010). Cheetah: A high performance, custom data warehouse on top of mapreduce. *VLDB Endow.*, 3(1-2):1459–1468.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

DeWitt, D. J., Halverson, A., Nehme, R., Shankar, S., Aguilar-Saborit, J., Avanes, A., Flasza, M., and Gramling, J. (2013). Split query processing in polybase. In *Proc. of ACM SIGMOD Internat. Conf. on Management of Data*, pages 1255–1266.

Friedman, E., Pawlowski, P., and Cieslewicz, J. (2009). Sql/mapreduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions. *VLDB Endow.*, 2(2):1402–1413.

Gandhi, V. A. and Kumbharana, C. K. (2013). Comparative study of amazon EC2 and microsoft azure cloud architecture. *Journal of Advanced Networking Apps*, pages 117–123.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

Gupta, A., Agarwal, D., Tan, D., Kulesza, J., Pathak, R., Stefani, S., and Srinivasan, V. (2015). Amazon redshift and the case for simpler data warehouses. In

Proc. of the ACM SIGMOD Internat. Conf. on Management of Data, pages 1917–1923.

Gupta, A., Yang, F., Govig, J., Kirsch, A., Chan, K., Lai, K., Wu, S., Dhoot, S. G., Kumar, A. R., Agiwal, A., Bhansali, S., Hong, M., Cameron, J., and et al. (2014). Mesa: Geo-Replicated, Near Real-Time, Scalable Data Warehousing. *PVLDB*, 7(12):1259–1270.

Hall, A., Bachmann, O., Büssow, R., Gănceanu, S., and Nunkesser, M. (2012). Processing a trillion cells per mouse click. *VLDB Endow.*, 5(11):1436–1446.

Khalifa, S., Elshater, Y., Sundaravarathan, K., Bhat, A., Martin, P., Imam, F., Rope, D., and et al. (2016). The six pillars for building big data analytics ecosystems. *ACM Comput. Surv.*, 49(2):33:1–33:36.

Kimball, R. (2012). The evolving role of the enterprise data warehouse in the era of big data analytics. White paper, Kimball Group, pages 1–31.

Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R., and Buyya, R. (2016). The anatomy of big data computing. *Softw. Pract. Exper.*, 46(1):79–105.

Lamb, A., Fuller, M., Varadarajan, R., Tran, N., Vandiver, B., Doshi, L., and Bear, C. (2012). The vertica analytic database: C-store 7 years later. *VLDB Endow.*, 5(12):1790–1801.

Lamport, L. (2001). Paxos made simple. *ACM SIGACT News (Distributed Computing Column)*, 32(4):51–58.

Lee, G., Lin, J., Liu, C., Lorek, A., and Ryaboy, D. (2012). The unified logging infrastructure for data analytics at twitter. *VLDB Endow.*, 5(12):1771–1780.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of Massive Datasets*. Cambridge University Press.

Madhuri, T. and Sowjanya, P. (2016). Microsoft azure v/s amazon aws cloud services: A comparative study. *Journal of Innovative Research in Science, Engineering and Technology*, 5(3):3904–3908.

Meijer, E., Beckman, B., and Bierman, G. (2006). Linq: Reconciling object, relations and xml in the .net framework. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 706–706.

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. (2011). Dremel: Interactive analysis of web-scale datasets. *Communications of the ACM*, 54:114–123.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: A not-so-foreign language for data processing. In *Proc. of SIGMOD Internat. Conf. on Management of Data*, pages 1099–1110.

Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., and Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. In *Proc. of the ACM SIGMOD Intern. Conf. on Management of Data*, pages 165–178.

Pedro, E., Rocha, P., Luis, E. d. B., and Chris, C. (2015). Cubrick: A scalable distributed molap database for fast analytics. In *Proc. of Internat. Conf. on Very Large Databases (Ph.D Workshop)*, pages 1–4.

Philip Chen, C. and Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275(Complete):314–347.

Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. (2005). Interpreting the data: Parallel analysis with sawzall. *Sci. Program.*, 13(4):277–298.

Pkknen, P. and Pakkala, D. (2015). Reference architecture and classification of technologies, products and services for big data systems. *Big Data Research*, 2(4):166 – 186.

Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., and Rasin, A. (2010). Mapreduce and parallel dbmss: Friends or foes? *Commun. ACM*, 53(1):64–71.

Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., and Shah, S. (2012). Serving large-scale batch computed data with project voldemort. In *Proc. of the 10th USENIX Conference on File and Storage Technologies*, pages 18–18.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: A warehousing solution over a map-reduce framework. *VLDB Endow.*, 2(2):1626–1629.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., and Murthy, R. (2010). Hive - a petabyte scale data warehouse using hadoop. In *Proc. of Internat. Conf. on Data Engineering*, pages 996–1005.

Warneke, D. and Kao, O. (2009). Nephele: Efficient parallel data processing in the cloud. In *Proc. of the 2Nd Workshop on Many-Task Computing on Grids and Supercomputers*, pages 8:1–8:10.

Wu, L., Sumbaly, R., Riccomini, C., Koo, G., Kim, H. J., Kreps, J., and Shah, S. (2012). Avatara: Olap for web-scale analytics products. *Proc. VLDB Endow.*, 5(12):1874–1877.

Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., and Stoica, I. (2013). Shark: Sql and rich analytics at scale. In *Proc. of ACM SIGMOD Internat. Conf. on Management of Data*, pages 13–24.

Xu, Y., Kostamaa, P., and Gao, L. (2010). Integrating hadoop and parallel dbms. In *Proc. of SIGMOD Internat. Conf. on Management of Data*, pages 969–974.

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., and Ganguli, D. (2014). Druid: A real-time analytical data store. In *Proc. of ACM SIGMOD Internat. Conf. on Management of Data*, SIGMOD '14, pages 157–168, New York, NY, USA. ACM.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of Conf. on Networked Systems Design and Implementation*, pages 15–28.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proc. of Conf. on Hot Topics in Cloud Computing*, pages 10–10.

Zhou, J., Bruno, N., Wu, M.-C., Larson, P.-A., Chaiken, R., and Shakib, D. (2012). Scope: Parallel databases meet mapreduce. *The VLDB Journal*, 21(5):611–636.