Challenges for MapReduce in Big Data

Katarina Grolinger[1], Michael Hayes[1], Wilson A. Higashino[1,2], Alexandra L'Heureux[1]
David S. Allison[1,3,4,5], Miriam A.M. Capretz[1]

[1]Department of Electrical and Computer Engineering
Western University, London, ON, Canada N6A 5B9
{kgroling, mhayes34, alheure2, whigashi, dallison, mcapretz}@uwo.ca
[2]Instituto de Computação
Universidade Estadual de Campinas, Campinas, Brazil

[3]CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[4]Univ de Toulouse, LAAS, F-31400 Toulouse, France
[5]Univ de Toulouse, UT1-Capitole, LAAS, F-31000 Toulouse, France
dallison@laas.fr

***Abstract*— In the Big Data community, MapReduce has been seen as one of the key enabling approaches for meeting continuously increasing demands on computing resources imposed by massive data sets. The reason for this is the high scalability of the MapReduce paradigm which allows for massively parallel and distributed execution over a large number of computing nodes. This paper identifies MapReduce issues and challenges in handling Big Data with the objective of providing an overview of the field, facilitating better planning and management of Big Data projects, and identifying opportunities for future research in this field. The identified challenges are grouped into four main categories corresponding to Big Data tasks types: data storage (relational databases and NoSQL stores), Big Data analytics (machine learning and interactive analytics), online processing, and security and privacy. Moreover, current efforts aimed at improving and extending MapReduce to address identified challenges are presented. Consequently, by identifying issues and challenges MapReduce faces when handling Big Data, this study encourages future Big Data research.**

Keywords- Big Data, Big Data Analytics, MapReduce, NoSQL, Machine Learning, Interactive Analytics, Online Processing, Privacy, Security

I. INTRODUCTION

Recent developments in the Web, social media, sensors and mobile devices have resulted in the explosion of data set sizes. For example, Facebook today has more than one billion users, with over 618 million active users generating more than 500 terabytes of new data each day [1]. Traditional data processing and storage approaches were designed in an era when available hardware, storage and processing requirements were very different than they are today. Thus, those approaches are facing many challenges in addressing Big Data demands.

The term "Big Data" refers to large and complex data sets made up of a variety of structured and unstructured data which are too big, too fast, or too hard to be managed by traditional techniques. Big Data is characterized by the 4Vs [2]: volume, velocity, variety, and veracity. Volume refers to the quantity of data, variety refers to the diversity of data types, velocity refers both to how fast data are generated and how fast they must be processed, and veracity is the ability to trust the data to be accurate and reliable when making crucial decisions.

Enterprises are aware that Big Data has the potential to impact core business processes, provide competitive advantage, and increase revenues [2]. Thus, organizations are exploring ways to make better use of Big Data by analyzing them to find meaningful insights which would lead to better business decisions and add value to their business.

MapReduce is a highly scalable programming paradigm capable of processing massive volumes of data by means of parallel execution on a large number of commodity computing nodes. It was recently popularized by Google [3], but today the MapReduce paradigm has been implemented in many open source projects, the most prominent being the Apache Hadoop [4]. The popularity of MapReduce can be accredited to its high scalability, fault-tolerance, simplicity and independence from the programming language or the data storage system.

In the Big Data community, MapReduce has been seen as one of the key enabling approaches for meeting the continuously increasing demands on computing resources imposed by massive data sets. At the same time, MapReduce faces a number of obstacles when dealing with Big Data including the lack of a high-level language such as SQL, challenges in implementing iterative algorithms, support for iterative ad-hoc data exploration, and stream processing.

This paper aims to identify issues and challenges faced by MapReduce when confronted by Big Data with the objectives of: a) providing an overview and categorization of the MapReduce issues and challenges, b) facilitating better planning and management of Big Data projects and c) identifying opportunities for future research in this field.

Other MapReduce-related surveys have been previously published, but this work has a different focus. Li *et al.* [5] presented a review of approaches focused on the support of distributed data management and processing using MapReduce. They discussed implementations of database operators in MapReduce and DBMS implementations using MapReduce, while this paper is concerned with identifying MapReduce challenges in Big Data.

Doulkeridis and Nørvåg [6] surveyed the state of the art in improving the performance of MapReduce processing and reviewed generic MapReduce weaknesses and

TABLE I. AN OVERVIEW OF MAPREDUCE CHALLENGES

	Main challenges	**Main solution approaches**
Data Storage	Schema-free, index-free	In-database MapReduce
		NoSQL stores – MapReduce with various indexing approaches
	Lack of standardized SQL-like language	Apache Hive – SQL on top of Hadoop
		NoSQL stores: proprietary SQL-like languages (Cassandra, MongoDB) or Hive (HBase)
Analytics	Scaling complex linear algebra	Use computationally less expensive, though less accurate, algebra
	Interactive analysis	Map interactive query processing techniques for handling small data, to MapReduce
	Iterative algorithms	Extensions of MapReduce implementation such as Twister and HaLoop
	Statistical challenges for learning	Data pre-processing using MapReduce
Online processing	Performance / Latency issues	Direct communication between phases and jobs
	Programming model	Alternative models, such as MapUpdate and Twitter's Storm
Privacy and security	Auditing	Trusted third party monitoring, security analytics
	Access control	Optimized access control approach with semantic understanding
	Privacy	Privacy policy enforcement with security to prevent information leakage

challenges. Sakr *et al.* [7] also surveyed approaches to data processing based on the MapReduce paradigm. Additionally, they analyzed systems which provide declarative programming interfaces on top of MapReduce. While the works of Doulkeridis and Nørvåg [6], and Sakr *et al.* [7] focused on systems built on top of MapReduce, this paper aims to identify challenges that MapReduce faces handling Big Data. Moreover, this paper discusses security and privacy issues, while those others do not.

The identified MapReduce challenges are grouped into four main categories corresponding to Big Data tasks types: data storage, analytics, online processing, security and privacy. An overview of the identified challenges is presented in Table I while details of each category are discussed in sections III to VI. Additionally, this paper presents current efforts aimed at improving and extending MapReduce to address the identified challenges.

The rest of this paper is organized as follows: Section II introduces the MapReduce paradigm. Section III identifies storage-related challenges while Section IV discusses Big Data analytics issues. Online processing is addressed in Section V and privacy and security challenges in Section VI. Finally, Section VII concludes the paper.

II. MAPREDUCE OVERVIEW

MapReduce is a programming paradigm for processing large data sets in distributed environments [3]. In the MapReduce paradigm, the *Map* function performs filtering and sorting, while the *Reduce* function carries out grouping and aggregation operations. The 'hello world' of MapReduce is the word counting example: it counts the appearance of each word in a set of documents. The *Map* function splits the document into words and for each word in a document it produces a (key, value) pair.

```
function map(name, document)
  for each word in document
    emit (word, 1)
```

The *Reduce* function is responsible for aggregating information received from *Map* functions. For each key, `word`, the *Reduce* function works on the list of values, `partialCounts`. To calculate the occurrence of each word, the *Reduce* function groups by word and sums the values received in the `partialCounts` list.

```
function reduce (word, List partialCounts)
  sum = 0
  for each pc in partialCounts
    sum += pc
  emit (word, sum)
```

The final output is the list of words with the count of appearance of each word.

Figure 1 illustrates the MapReduce flow. One node is elected to be the master responsible for assigning the work, while the rest are workers. The input data is divided into splits and the master assigns splits to *Map* workers. Each worker processes the corresponding input split, generates key/value pairs and writes them to intermediate files (on disk or in memory). The master notifies the *Reduce* workers about the location of the intermediate files and the *Reduce* workers read data, process it according to the *Reduce* function, and finally, write data to output files.

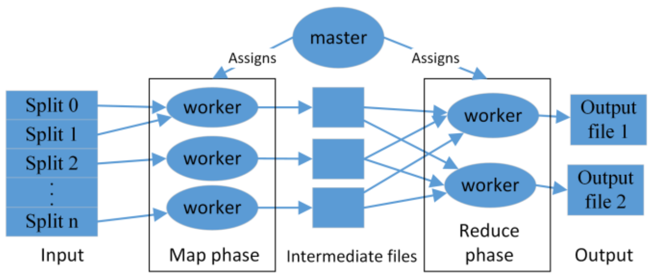

Figure 1. MapReduce flow

The main contribution of the MapReduce paradigm is scalability as it allows for highly parallelized and distributed execution over a large number of nodes. In the MapReduce paradigm, the *Map* or *Reduce* task is divided into a high number of jobs which are assigned to nodes in the network. Reliability is achieved by reassigning any failed node's job to another node. A well known open source MapReduce implementation is Hadoop which implements MapReduce on top of the Hadoop Distributed File System (HDFS).

III. DATA STORAGE

Relational database management systems (RDBMSs) are traditional storage systems designed for structured data and accessed by means of SQL. RDBMSs are facing challenges in handling Big Data and providing horizontal scalability, availability and performance required by Big Data applications. In contrast to relational databases, MapReduce provides computational scalability, but it relies on data storage in a distributed file system such as Google File System (GFS) or Hadoop Distributed File System (HDFS).

NoSQL and NewSQL data stores have emerged as alternatives to Big Data storage. NoSQL refers to “Not Only SQL”, highlighting that SQL is not a crucial objective of those systems. Their main defining characteristics include schema flexibility and effective scaling over a large number of commodity machines. NoSQL horizontal scalability includes data storage scaling as well as scaling of read/write operations. Grolinger *et al.* [8] analyze features driving the NoSQL systems ability to scale such as partitioning, replication, consistency, and concurrency control. NoSQL systems typically adopt the MapReduce paradigm and push processing to the nodes where data is located to efficiently scale read operations. Consequently, data analysis is performed via MapReduce jobs.

MapReduce itself is schema-free and index-free; this provides great flexibility and enables MapReduce to work with semi-structured and unstructured data. Moreover, MapReduce can run as soon as data is loaded. However, the lack of indexes on standard MapReduce may result in poor performance in comparison to relational databases. This may be outweighed by MapReduce scalability and parallelization.

Database vendors, such as Oracle, provide in-database MapReduce [9], taking advantage of database parallelization. Another example of providing analytics capabilities in-database is the MAD Skills project [10] which implements MapReduce within the database using an SQL runtime execution engine. *Map* and *Reduce* functions are written in Python, Perl, or R, and passed to the database for execution.

NoSQL systems from column-family and document categories adopt the MapReduce paradigm while providing support for various indexing methods. In this approach MapReduce jobs can access data using the index, therefore query performance is significantly improved. For example Cassandra supports primary and secondary indexes [11]. In CouchDB [12] the primary way of querying and reporting is through views which use the MapReduce paradigm with JavaScript as a query language. A view consists of a *Map* function and an optional *Reduce* function. Data emitted by *Map* function is used to construct an index and consequently, queries against that view run quickly.

Another challenge related to MapReduce and data storage is the lack of a standardized SQL-like language. Therefore one direction of research is concerned with providing SQL on top of MapReduce. An example of this category is Apache Hive [13] which provides an SQL-like language on top of Hadoop. Another Apache effort, Mahout [14], aims to build scalable machine learning libraries on top of MapReduce. Although those efforts provide powerful data processing capabilities, they lack data management features such as advanced indexing and a sophisticated optimizer.

NoSQL solutions choose different approaches for providing querying abilities [8]: Cassandra and MongoDB provide proprietary SQL-like querying while HBase uses Hive.

It is important to point out the efforts on integration between traditional databases, MapReduce, and Hadoop. For example, the Oracle SQL connector for HDFS [15] provides ability to query data in Hadoop within the database using SQL. The Oracle Data Integrator for Hadoop generates Hive-like queries which are transformed into native MapReduce and executed on Hadoop clusters.

Even though the presented efforts advanced the state of the art for Data Storage and MapReduce, a number of challenges remain, such as:

- the lack of a standardized SQL-like query language,
- limited optimization of MapReduce jobs,
- integration among MapReduce, distributed file system, RDBMSs and NoSQL stores.

IV. BIG DATA ANALYTICS

A. *Machine Learning*

The prevalence and pervasiveness of Big Data offers the promise of building more intelligent decision making systems. This is because the typical premise for many decision making algorithms is that more data can better *teach* the algorithms to produce more accurate outputs. The key to extracting useful information from Big Data lies within the use of Machine Learning (ML) approaches. However, the use of massive datasets themselves for the purpose of analysis and training poses some problems and challenges to the very execution of ML algorithms. The arithmetic and computational complexity brought on by the volume component of Big Data renders traditional ML algorithms almost unusable in conventional development environments. This is due to the fact that ML algorithms were designed to be used on much smaller dataset with the assumption that the entire data could be held in memory [16]. With the arrival of Big Data, this assumption is no longer valid and consequently greatly impedes the performance of those algorithms. In order to remediate to this problem, distributed processing algorithms such as MapReduce were brought forward.

Although some ML algorithms are inherently parallel and can therefore be adapted to the MapReduce paradigm [17], for others the transition is much more complex. The foundation of many ML algorithms relies on strategies directly dependent on in-memory data and therefore once that assumption is severed, entire families of algorithms are rendered inadequate. The parallel and distributive nature of the MapReduce paradigm is a source of such a disconnect. This is what Parker [17] describes as the curse of modularity. The following families of algorithms are amongst those affected [18]:

- Iterative Graph algorithms: Multiple iterations are required in order to reach convergence, each of which corresponds to a job in MapReduce[18] and jobs are expensive in terms of startup time. Furthermore, skews in the data create stragglers in the *Reduce* phase, which causes backup execution to be launched, increasing the computational load [3].
- Gradient Descent algorithms: The sequential nature of these algorithms requires a very large amount of jobs to be chained. It also requires that parameters be updated after each iteration, which will add communication overhead to the process. Both of these steps are therefore expensive in terms of time.

- Expectation Maximization algorithms: Similarly this family of algorithm also depends on iterations that are implemented as jobs, causing the same performance latencies as above.

In order to address the shortcomings of MapReduce, alternatives have been developed to function either independently or in addition to existing MapReduce implementations [18]:

- Pregel [19] and Giraph [20], are alternative models based on the Bulk Synchronous parallel paradigm. They enable all states to be retained in memory, facilitating the iterative process.
- Spark [21] is another alternative based on resilient distributed datasets abstractions, which uses memory to update shared states and facilitate implementations such as gradient descent.
- HaLoop [22] and Twister [23] are both extensions designed for Hadoop [4] in order for this MapReduce implementation to better support iterative algorithms.

Each of these tools possesses its strengths and area of focus but the difficult integration and potential incompatibilities between the tools and frameworks reveal new research opportunities that would fulfill the need for a uniform ML solution.

When considering the volume component of Big Data, additional statistical and computational challenges are revealed. Regardless of the paradigm used to develop the algorithms, an important determinant of the success of supervised ML approaches is the pre-processing of the data. This step is often critical in order to obtaining reliable and meaningful results. Data cleaning, normalization, feature extraction and selection [24] are all essential in order to obtain an appropriate training set. This poses a massive challenge in the light of Big Data as the preprocessing of massive amounts of tuples is often not possible.

The variety component of Big Data, also introduces heterogeneity and high dimensionality, which in turn introduces the following challenges [25]:

- Noise accumulation may be so great that it may over power the significant data.
- Spurious or false correlation may present between different data points although no real relationship exist.
- Incidental endogeneity, meaning that regressors are related to the regression error, which could lead to inconsistencies and false discoveries [26].

In particular, the concept of noise has provided a paradigm shift in the underlying algebra used for ML algorithms. Dalessandro [27] illustrates the usefulness of accepting noise as a given, and then using more efficient, but less accurate, learning models. Dalessandro shows that using computationally less expensive algorithms, which are also less accurate during intermediate steps, will define a model which performs equally well in predicting new outputs when trained on Big Data. These algorithms may take more iterations than their computationally more expensive counterparts; however, the iterations are much faster. Due to this, the less expensive algorithms tend to converge much faster, while giving the same accuracy. An example of such an algorithm is stochastic gradient descent [27].

In addition to the challenges mentioned above, having a variety of dissimilar data sources, each storing dissimilar data types, can also affect the performance of the ML algorithms. Data preprocessing could alleviate some of those challenges and is particularly important in the MapReduce paradigm where outliers can greatly influence the performance of algorithms [28]. In order to remediate to these problems, solutions have been developed to implement data preprocessing algorithms using MapReduce [29]. However, it is still necessary to find ways to integrate the analysis and preprocessing phase, which create new research prospects.

The velocity component of Big Data introduces the idea of *concept drift* within the learning model. In MapReduce, this idea is aggravated by the necessity to pre-process data, which introduces additional delays. The fast arrival of data along with potentially long computing time may cause a concept drift, which Yang and Fong define as "*known problem in data analytics, in which the statistical properties of the attributes and their target classes shift over time, making the trained model less accurate"[30]*. Thus accurate concept drift detection constitutes an important research area to insure accuracy of ML approaches with Big Data.

An important subset of ML algorithms is predictive modeling. That is, given a set of known inputs and outputs, can we predict an unknown output with some probability? Being able to construct an accurate prediction model is hugely important in many disparate domains such as credit card fraud detection, user recommendation systems, malicious URL identification, and many others. For example, to predict movies that clients will enjoy, companies such as Yahoo and Netflix collect a large variety of information on their clients to build accurate recommender systems.

From the authors observation, parallelism techniques for predictive modeling fall into three categories of implementation:

1. Run the predictive algorithm on subsets of the data, and return all the results.
2. Generate intermediate results from subsets of the data, and resolve the intermediate results into a final result.
3. Parallelize the underlying linear algebra.

The two most promising forms of implementation for Big Data are categories 2 and 3. Category 2 is essentially the definition of a MapReduce job; where the algorithm attempts to generate intermediate results using *Map* operations, and combines these outputs using *Reduce* operations. Category 3 can also be seen as a MapReduce job, if the underlying linear algebra separable into *Map* and *Reduce* operations. Finally, Category 1 is essentially not a valid solution for Big Data as the results are only indicative of small subsets of the data and not the prediction over the entire dataset.

MapReduce with predictive modeling has a major constraint which limits its usefulness when predicting highly correlated data. MapReduce works well in contexts where observations can be processed individually. In this case the data can be split up, calculated, and then aggregated together. However, if there are correlated observations that need to be processed together, MapReduce offers little benefit over non-distributed architectures. This is because it will be quite common that the observations that are correlated are found within disparate clusters, leading to large performance overheads for data communication between clusters. Use cases such as this are commonly found in predicting stock market fluctuations. To allow MapReduce to be used in these types of predictive modeling problems, there are a few potential solutions based on solutions from predictive modeling on traditional data sizes: data reduction, data aggregation, and sampling [31].

B. *Interactive Analytics*

Interactive analytics can be defined as a set of approaches to allow data scientists to explore data in an interactive way, supporting exploration at the rate of human thought [32]. Interactive analytics on Big Data provides some exciting research areas and unique problems. Most notably, and similar to other data analytic approaches, is the question *how can we build scalable systems that query and visualize data at interactive rates?* The important difference to other data analytic paradigms is the notion of *interactive rates.* By definition, interactive analysis requires the user to continually tweak or modify their approach to generate interesting analytics [33].

MapReduce for interactive analytics poses a drastic shift from the classic MapReduce use case of processing batch computations. Interactive analytics involves performing several small, short, and interactive jobs. As interactive analytics begins to move from RDBMSs to Big Data storage systems some prior assumptions regarding MapReduce are broken, such as uniform data access and prevalence of large batch jobs. This type of analysis requires a new *class* of MapReduce workloads to deal with the interactive, almost real-time data models. Chen *et al.* [34] discuss these considerations in their survey of industry solutions where the authors find that extending MapReduce with querying frameworks such as Pig and Hive are prevalent. Chen *et al.* note that interactive analysis for Big Data can be seen as an extension of the already well-researched area of interactive query processing. Making this assumption, there exist potential solutions to optimize interactive analytics with MapReduce by mirroring the already existing work in interactive query processing. One open area of future work is finding the best method to bring these solutions to the MapReduce programming paradigm.

MapReduce is one parallelism model for interactive analytics. Another approach tuned for interactivity is Google's Dremel system [35], which acts in complement to MapReduce. Dremel builds on a novel column-family storage format, as well as algorithms that constructs the columns and reassemble the original data. Some highlights of the Dremel system are:

- Real-time interactivity for scan-based queries.
- Near linear scalability in the number of clusters.
- Early termination, similar to progressive analytics, to provide speed tradeoffs for accuracy.

Other interactive analytics research have been based on the column-family NoSQL data storage approach [36, 37]. The main benefit of column-based approaches versus row-based, traditional, approaches is that only a fraction of the data needs to be accessed when processing typical queries [8]. However, most of these approaches are specialized for certain types of datasets and certain queries and thus provide an open research area for a generalized solution.

C. *Data Visualization*

A large category of interactive analytics is data visualization. There are two primary problems associated with Big Data visualization. First, many instances of Big Data involve datasets with large amount of features, *wide datasets*, and building a highly multi-dimensional visualization is a difficult task. Second, as data grows larger vertically, *tall datasets*, uninformative visualizations are generally produced. For these *tall* datasets, the resolution of the data must be limited, i.e. through a process to aggregate outputs to ensure that highly dense data can still be deciphered [32]. For highly *wide* datasets, a preprocessing step to reduce the dimensionality is needed. Unfortunately this tends to be useful on tens to hundreds of dimensions, for even higher dimensions a mixed-initiative method, including human intervention, to determine subsets of related dimensions is required [32]. This approach generally requires human input to determine an initial subset of "interesting" features, which is also a difficult task and open research area.

MapReduce for data visualization currently performs well in two cases: memory-insensitive visualization algorithms, and inherently parallel visualization algorithms. Vo *et al.* [38] have provided a study on moving existing visualization algorithms to the MapReduce paradigm. One major contribution is empirically proving that MapReduce provides a good solution to large-scale exploratory visualization. The authors present that this is because scalability is achieved through data reduction tasks which can be highly parallel; these types of tasks are common in data visualization algorithms. Further, visualization algorithms that tend to increase the total amount of data for intermediate steps will perform poorly when mapping to the MapReduce paradigm. Another drawback to MapReduce with visualization is that a typical MapReduce job uses one pass over the data. Therefore, algorithms that require multiple iterations, such as mesh simplification, will suffer from a large overhead in trying to naively map the algorithm to the MapReduce paradigm. This is similar to the problems created for iterative machine learning algorithms discussed in Section IV-A. Therefore, there is the potential for research aimed at providing optimized multiple iteration solutions for MapReduce.

V. Online Processing

The Velocity dimension, as one of the Vs used to define Big Data, brings many new challenges to traditional data processing approaches and especially to MapReduce. Handling Big Data velocity often requires applications with *online processing* capabilities, which can be broadly defined as real-time or *quasi* real-time processing of fast and continuously generated data (also known as data *streams*). From the business perspective, the goal is normally to obtain insights from these data streams, and to enable prompt reaction to them. This instantaneous reaction can bring business value and competitive advantage to organizations, and therefore has been generating research and commercial interest. Areas such as financial fraud detection and algorithmic trading have been highly interested in this type of solutions.

The MapReduce paradigm is not an appropriate solution for this kind of low-latency processing because:

- MapReduce computations are batch processes that start and finish, while computations over streams are continuous tasks that only finish upon user request.
- The inputs of MapReduce computations are snapshots of data stored on files, and the content of these files do not change during processing. Conversely, data streams are continuously generated and unbounded inputs [39].
- In order to provide fault tolerance, most of MapReduce implementations, such as Google's [3] and Hadoop [4], write the results of the *Map* phase to local files before sending them to the reducers. In addition, these implementations store the output files in distributed and high-overhead file systems (Google File System [40] or HDFS [4], respectively). This extensive file manipulation adds significant latency to the processing pipelines.
- Not every computation can be efficiently expressed using the MapReduce programming paradigm, and the model does not natively support the composition of jobs.

Despite these limitations, the prevalence and success of MapReduce has motivated many researchers to work on systems that leverage some of its advantages, and at the same time try to overcome its limitations when applied to low-latency processing.

One of the first projects in this direction was developed by Condie *et al.* [41]. In this work, the authors proposed an online MapReduce implementation with the goal of supporting online aggregation and continuous queries. In order to reduce the processing latency, the *Map* and *Reduce* phases are pipelined by having the *Map* tasks sending intermediate results to the *Reduce* tasks. The authors also introduced the idea of executing reducers on *snapshots* of the data received from the mappers. This mechanism enables the generation of partial / approximate results, which is particularly useful for interactive analytics scenarios as described in Section IV-C. All these changes were implemented on Hadoop, and demonstrated in a monitoring system prototype.

Nevertheless, it is important to note that Condie *et al.*'s [41] work still has limitations that may hinder its use in online processing scenarios. For instance, if the reducers are not simultaneously scheduled with the mappers, the mappers cannot push the intermediate results to them. In addition, the platform does not support elasticity (dynamic scale in and out of provisioned resources), which is a very important requirement for scenarios where the data input rate is subject to high fluctuations and burst behavior.

To overcome the inherent limitations of the traditional MapReduce platforms, other authors have been leveraging the familiar MapReduce programming paradigm but additionally providing a different runtime environment. For instance, Logothetis and Yocum [42] proposed a continuous MapReduce in the context of a data processing platform that runs over a wide-area network. In this work, the execution of the *Map* and *Reduce* functions is managed by a data stream-processing platform. In order to improve the processing latency, the mappers are continuously fed with batches of tuples (instead of input files), and they push their results to reducers as soon as they are available. This approach is similar to the one adopted by the StreamMapReduce [43] project, which uses these ideas to implement a fully-fledged event stream processing (ESP) implementation.

Alternatively, the difficulty of expressing online computations using MapReduce has also been motivating the creation of other programming models inspired by it. For instance, the Muppet project [39] conceived a new programming paradigm called MapUpdate. The paradigm mimics MapReduce by specifying computations through the definition of two functions (*Map* and *Update*). The main difference, however, is the fact that the update phase has access to *slates*, data structures that contain persistent state related to each update key. In theory, these slates can enable easier implementations of iterative algorithms.

Other frameworks, such as Twitter's Storm [44] and Yahoo's S4 [45] propose a more radical departure from MapReduce programming paradigm, but maintain runtime platforms inspired by the MapReduce implementations. For instance, in Twitter's Storm [44], a computation is defined by a topology, which specifies the sequence of processing elements (*bolts*) containing user-defined logic, the number of threads (*tasks*) for each bolt, and how to partition the input streams among the many bolt tasks. Similarly, in Yahoo's S4 [45] case, a computation is expressed by a graph of processing elements (PE), which are equivalent to Storm's bolts. In both projects the runtime platform manages many low-level aspects of distributed computing, such as parallelization, messages delivery, and fault tolerance.

Finally, it is also worth mentioning the Spark Stream project [46] as another MapReduce alternative. The goal of this project is to provide a data stream processing framework based on the Spark platform [21]. Similar to Logothetis and Yocum [42], events are grouped into small batches and all processing is performed on these batches, which contrast with the event-by-event processing in Storm and S4.

Despite all the advancements described in this section, there still are many challenges related to online processing of Big Data, such as:

- Most platforms are designed to run on clusters of servers only, and cannot leverage the elasticity and automatic provisioning capabilities of modern cloud environments.
- Some use cases require a response time that is very difficult to achieve in networked environments.
- There is no high-level standardized language that can be used to express online computations.

- Current platforms require a considerable effort to deploy and manage, and cannot be easily integrated with other data processing platform.

VI. Security and Privacy

In this section security and privacy concerns for MapReduce and Big Data are discussed. Also, current efforts to address these problems for MapReduce are presented.

Accountability and auditing are security issues that present a problem for both MapReduce and Big Data. Accountability is the ability to know when someone performs an action and to hold them responsible for that action and is often tracked through auditing. In MapReduce accountability is only provided when the mappers and reducers are held responsible for the tasks they have completed [47]. One solution to this issue that has been proposed is the creation of an Accountable MapReduce [47]. This solution utilizes a set of auditors to inconspicuously perform accountability tests on the mappers and reducers in real-time [47]. Through the monitoring of the results of these tests, malicious mappers or reducers can be detected and accountability can be provided.

An additional security challenge presented to MapReduce and Big Data is that of providing access control, which can be shown through three of Big Data's defining V properties: volume, variety and velocity [48]. When dealing with a large volume of information, work performed on that information is likely to require access to multiple storage locations and devices. Therefore, multiple access requirements will be required for any one task. When dealing with data that has a large variety, semantic understanding of the data should play a role in the access control decision process [48]. Finally, the velocity requirement of MapReduce and Big Data requires that whatever access control approach is used must be optimized to determine access control rights in a reasonable amount of time.

Privacy is a major topic of concern whenever large amounts of information are used. Processes such as data mining and predictive analytics can discover or deduce information linkages. Information linkages are advantageous to organizations, allowing them to better understand, target and provide for their clients or users. However, on an individual basis this discovery of information can cause the identities of data providers to be exposed.

Privacy protection requires an individual to maintain a level of control over their personal information. This control can be provided through transparency and allowing input from the data provider. User input allows an individual to state their private information usage wishes. Transparency is provided to an individual by the knowledge of how private information is collected, what private information is collected, how the private information is being used, and who has access to it. This can be very difficult when dealing with a large number of mappers and reducers that MapReduce often requires. It is possible that the ability to provide transparency and control is stated in legislation that must be followed or penalties can be incurred. The following are examples of issues that could lead to penalties using the example of the Personal Information Protection and Electronic Documents Act (PIPEDA) in Canada [49], and the Data Protection Directive of the European Union [50]:

- Both require in their respective jurisdictions that individuals who have data collected on them are able to understand how it is being used, by whom, and for what purposes. Abiding by such legislation is difficult for any large data environment.
- Both state that in some circumstances consent must be given before information can be used. Due to the size of the data and the complexity of the analytics performed during a MapReduce, informing an individual about what is happening to their information is a challenge.
- Both state that consent can be withdrawn and if so the information should be deleted by the data repository. However, in Big Data once information has been put into the system it is difficult if not impossible to remove.

Some work has been done in order to provide privacy protection for MapReduce. Airavat [51] is a system that has been designed to enable the execution of trusted and untrusted MapReduce computations on sensitive data, while also providing enforcement of privacy policies belonging to the data providers [51]. Airavat splits the MapReduce process into two parts, the untrusted mapped code, and the trusted reducer code. Drawbacks of the Airavat solution include the mandatory use of an Airavat provided Reducer, which reduces its ability to operate in any domain. While this initial approach has shown some promise, there is still room for improvement.

VII. Conclusions

Traditional data processing and storage approaches are facing many challenges in meeting the continuously increasing computing demands of Big Data. This work focused on MapReduce, one of the key enabling approaches for meeting Big Data demands by means of highly parallel processing on a large number of commodity nodes.

Issues and challenges MapReduce faces when dealing with Big Data are identified and categorized according to four main Big Data task types: data storage, analytics, online processing, and security and privacy. Moreover, efforts aimed at improving and extending MapReduce to address identified challenges are presented. By identifying MapReduce challenges in Big Data, this paper provides an overview of the field, facilitates better planning of Big Data projects and identifies opportunities for future research.

References

[1] P. Zadrozny and R. Kodali, Big Data Analytics using Splunk, Berkeley, CA, USA: Apress, 2013.

[2] F. Ohlhorst, Big Data Analytics: Turning Big Data into Big Money, Hoboken, N.J, USA: Wiley, 2013.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun ACM, 51(1), pp. 107-113, 2008.

[4] Apache Hadoop, http://hadoop.apache.org.

[5] F. Li, B. C. Ooi, M. T. Özsu and S. Wu, "Distributed data management using MapReduce," ACM Computing Surveys, 46(3), pp. 1-42, 2014.

[6] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," The VLDB Journal, pp. 1-26, 2013.

[7] S. Sakr, A. Liu and A. Fayoumi, "The family of mapreduce and large-scale data processing systems," ACM Computing Surveys, 46(1), pp. 1-44, 2013.

[8] K. Grolinger, W. A. Higashino, A. Tiwari and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," Journal of Cloud Computing: Advances, Systems and Application, 2, 2013.

[9] X. Su and G. Swart, "Oracle in-database hadoop: When MapReduce meets RDBMS," Proc. of the 2012 ACM SIGMOD International Conference on Management of Data, 2012.

[10] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein and C. Welton, "MAD skills: New analysis practices for Big Data," VLDB Endowment, 2(2), pp. 1481-1492, 2009.

[11] Apache Cassandra, http://www.datastax.com/docs.

[12] J. C. Anderson, J. Lehnardt and N. Slater, CouchDB: The Definitive Guide, Sebastopol, CA, USA: O'Reilly Media, 2010.

[13] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy, "Hive: A warehousing solution over a map-reduce framework," Proc. of the VLDB Endowment, 2(2), pp. 1626-1629, 2009.

[14] Apache Mahout, https://mahout.apache.org/.

[15] Oracle Big Data connectors, http://www.oracle.com/us/products/database/big-data-connectors/overview/index.html.

[16] K. A. Kumar, J. Gluck, A. Deshpande and J. Lin, "Hone: Scaling down hadoop on shared-memory systems," Proc. of the VLDB Endowment, 6(12), pp. 1354-1357, 2013.

[17] C. Parker, "Unexpected challenges in large scale machine learning," Proc. of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, 2012.

[18] J. Lin, "Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!" Big Data, 1(1), pp. 28-37, 2013.

[19] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: A system for large-scale graph processing," Proc. of the 2010 ACM SIGMOD International Conference on Management of Data, 2010.

[20] Apache Giraph, https://giraph.apache.org/.

[21] Apache Spark, https://spark.incubator.apache.org/.

[22] Y. Bu, B. Howe, M. Balazinska and M. D. Ernst, "HaLoop: Efficient iterative data processing on large clusters," Proc.VLDB Endow., 3(1-2), pp. 285-296, 2010.

[23] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu and G. Fox, "Twister: A runtime for iterative MapReduce," Proc. of the 19th ACM International Symposium on High Performance Distributed Computing, 2010.

[24] S. B. Kotsiantis, D. Kanellopoulos and P. Pintelas, "Data preprocessing for supervised learning," International Journal of Computer Science, 1(2), pp. 111, 2006.

[25] J. Fan, F. Han and H. Liu, "Challenges of Big Data analysis," National Science Review, in press, 2014.

[26] J. Fan and Y. Liao, "Endogeneity in high dimensions," The Annals of Statistics, in press, 2014.

[27] B. Dalessandro, "Bring the noise: Embracing randomness is the key to scaling up machine learning algorithms," Big Data, 1(2), pp. 110-112, 2013.

[28] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha and E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation, 2010.

[29] Q. He, Q. Tan, X. Ma and Z. Shi, "The high-activity parallel implementation of data preprocessing based on MapReduce," Proc. of the 5th International Conference on Rough Set and Knowledge Technology, 2010.

[30] H. Yang and S. Fong, "Countering the concept-drift problem in Big Data using iOVFDT," IEEE International Congress on Big Data, 2013.

[31] T. Hill and P. Lewicki, STATISTICS: Methods and Applications, Tulsa, OK: StatSoft, 2007.

[32] J. Heer and S. Kandel, "Interactive analysis of Big Data," XRDS: Crossroads, the ACM Magazine for Students, 19(1), pp. 50-54, 2012.

[33] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar and R. Pasquin, "Incoop: MapReduce for incremental computations," Proc. of the 2nd ACM Symposium on Cloud Computing, 2011.

[34] Y. Chen, S. Alspaugh and R. Katz, "Interactive analytical processing in Big Data systems: A cross-industry study of MapReduce workloads," Proc. of the VLDB Endowment, 5(12), pp. 1802-1813, 2012.

[35] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton and T. Vassilakis, "Dremel: Interactive analysis of Web-scale datasets," Proc. of the VLDB Endowment, 3(1-2), pp. 330-339, 2010.

[36] A. Hall, O. Bachmann, R. Büssow, S. Gănceanu and M. Nunkesser, "Processing a trillion cells per mouse click," Proc. of the VLDB Endowment, 5(11), pp. 1436-1446, 2012.

[37] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," Proc. of the Eight ACM European Conference on Computer Systems, 2013.

[38] H. T. Vo, J. Bronson, B. Summa, J. L. Comba, J. Freire, B. Howe, V. Pascucci and C. T. Silva, "Parallel visualization on large clusters using MapReduce," IEEE Symposium on Large Data Analysis and Visualization, 2011.

[39] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri and A. Doan, "Muppet: MapReduce-style processing of fast data," Proc.VLDB Endow., 5(12), pp. 1814-1825, 2012.

[40] S. Ghemawat, H. Gobioff and S. Leung, "The Google file system," ACM SIGOPS Operating Systems Review, 2003.

[41] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, "MapReduce online," Proc. of the 7th USENIX Conference on Networked Systems Design and Implementation, 2010.

[42] D. Logothetis and K. Yocum, "Ad-hoc data processing in the cloud," Proc. of the VLDB Endowment, 1(2), pp. 1472-1475, 2008.

[43] A. Brito, A. Martin, T. Knauth, S. Creutz, D. Becker, S. Weigert and C. Fetzer, "Scalable and low-latency data processing with stream MapReduce," IEEE Third International Conference on Cloud Computing Technology and Science, pp. 48-58, 2011.

[44] Storm, distributed and fault-tolerant realtime computation, http://storm-project.net/.

[45] L. Neumeyer, B. Robbins, A. Nair and A. Kesari, "S4: Distributed stream computing platform," Proc. of IEEE International Conference on Data Mining Workshops, 2010.

[46] M. Zaharia, T. Das, H. Li, S. Shenker and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters," Proc. of the 4th USENIX Conference on Hot Topics in Cloud Computing, 2012.

[47] Z. Xiao and Y. Xiao, "Achieving accountable MapReduce in cloud computing," Future Generation Computer Systems, 30,pp. 1-13, 2014.

[48] W. Zeng, Y. Yang and B. Luo, "Access control for Big Data using data content," IEEE International Conference on Big Data, 2013.

[49] The Personal Information Protection and Electronic Documents Act (PIPEDA), http://www.priv.gc.ca/leg_c/r_o_p_e.asp.

[50] Protection of Personal Data, http://ec.europa.eu/justice/data-protection.

[51] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov and E. Witchel, "Airavat: Security and privacy for MapReduce." Proc. of the 7th Usenix Symposium on Networked Systems Design and Implementation, 2010.