Overview

Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks

Alberto Fernández,[1]* Sara del Río,[2] Victoria López,[2] Abdullah Bawakid,[3] María J. del Jesus,[1] José M. Benítez[2] and Francisco Herrera[2,3]

The term 'Big Data' has spread rapidly in the framework of Data Mining and Business Intelligence. This new scenario can be defined by means of those problems that cannot be effectively or efficiently addressed using the standard computing resources that we currently have. We must emphasize that Big Data does not just imply large volumes of data but also the necessity for scalability, i.e., to ensure a response in an acceptable elapsed time. When the scalability term is considered, usually traditional parallel-type solutions are contemplated, such as the Message Passing Interface or high performance and distributed Database Management Systems. Nowadays there is a new paradigm that has gained popularity over the latter due to the number of benefits it offers. This model is Cloud Computing, and among its main features we has to stress its elasticity in the use of computing resources and space, less management effort, and flexible costs. In this article, we provide an overview on the topic of Big Data, and how the current problem can be addressed from the perspective of Cloud Computing and its programming frameworks. In particular, we focus on those systems for large-scale analytics based on the MapReduce scheme and Hadoop, its open-source implementation. We identify several libraries and software projects that have been developed for aiding practitioners to address this new programming model. We also analyze the advantages and disadvantages of MapReduce, in contrast to the classical solutions in this field. Finally, we present a number of programming frameworks that have been proposed as an alternative to MapReduce, developed under the premise of solving the shortcomings of this model in certain scenarios and platforms© 2014 John Wiley & Sons, Ltd.

How to cite this article:
WIREs Data Min Knowl Discov 2014, 4:380–409. doi: 10.1002/widm.1134

INTRODUCTION

We are immersed in the Information Age where vast amounts of data are available. Petabytes of data are recorded everyday resulting in a large *volume* of information; this incoming information arrives at a high rate and its processing involves real-time requirements implying a high *velocity*; we may find a wide *variety* of structured, semi-structured, and unstructured data; and data have to be cleaned before the integration into the system in order to maintain *veracity*.[1] This *4V property* is one of the most widespread definitions of what is known as the *Big Data* problem,[2,3] which has become a hot topic of interest within academia and corporations.

*Correspondence to: alberto.fernandez@ujaen.es

[1]Department of Computer Science, University of Jaen, Jaen, Spain

[2]Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain

[3]Faculty of Computing and Information Technology—North Jeddah, King Abdulaziz University, Jeddah, Saudi Arabia

Conflict of interest: The authors have declared no conflicts of interest for this article.

The current *explosion of data* that is being generated is due to three main reasons[4]: (1) hundreds of applications such as mobile sensors, social media services, and other related devices are collecting information continuously; (2) storage capacity has improved so much that collecting data is cheaper than ever, making preferable to buy more storage space rather than deciding what to delete; (3) Machine Learning and information retrieval approaches have reached a significant improvement in the last years, thus enabling the acquisition of a higher degree of knowledge from data.[5,6]

Corporations are aware of these developments. Gaining critical business insights by querying and analyzing such massive amounts of data is becoming a necessity. This issue is known as Business Intelligence (BI),[7,8] which refers to decision support systems that combine data gathering, data storage, and knowledge management with analysis to provide input to the decision process.[9] Regarding the former issues, a new concept appears as a more general field, integrating data warehousing, Data Mining (DM), and data visualization for Business Analytics. This topic is known as Data Science.[10,11]

The data management and analytics carried out in conventional database systems (and other related solutions) cannot address the Big Data challenges: data size is too large, values are modified rapidly, and/or they do no longer satisfy the constraints of Database Management Systems (DBMS). According to this fact, new systems have emerged to solve the previous issues: (1) 'Not Only SQL' (NoSQL) systems that modify the storage and retrieval of key/value pairs for interactive data serving environments[12] and (2) systems for large-scale analytics based on the MapReduce parallel programming model,[13] Hadoop being the most relevant implementation.[14]

These two approaches are under the umbrella of Cloud Computing.[15–17] Cloud Computing has been designed to reduce computational costs and increase the elasticity and reliability of the systems.[18] It is also intended to allow the user to obtain various services without taking into consideration the underlying architecture, hence offering a transparent scalability. The basis of Cloud Computing is the Service-Oriented Architecture,[19] which is designed to allow developers to overcome many distributed organization computing challenges including application integration, transaction management, and security policies.

The advantages of this new computational paradigm with respect to alternative technologies are clear, especially regarding BI.[20] First, cloud application providers strive to give the same or better service and performance as if the software programs were locally installed on end-user computers, so the users do not need to spend money buying complete hardware equipment for the software to be used. Second, this type of environment for the data storage and the computing schemes allows companies to get their applications up and running faster. They have a lower need of maintenance from the Information Technology department as Cloud Computing automatically manages the business demand by dynamically assigning resources (servers, storage, and/or networking) depending on the computational load in real time.[21]

Being a hot topic, there are currently a high number of works that cover the Big Data problem. But there is also a lack of an unifying view in order to fully understand the nature of this issue, and the ways to manage it. New approaches emerge every day and it is hard to follow this trending topic. In particular, there is not a clear 'guide' for a new user whose aim is to get introduced to the solutions to this problem. According with the above, the main contributions of this article are summarized as follows:

(1) An introduction to the 'Big Data' is given. The significance of addressing properly Big Data in DM and BI applications is stressed.
(2) We will show that Cloud Computing platforms (e.g., Amazon's EC2, Microsoft Azure, and so on) enable researchers to perform Big Data analysis in a very flexible way and without much fixed costs. But we also point out that Big Data technologies can also be deployed in noncloud clusters of computers.
(3) New approaches that have been developed for achieving scalability in Big Data are described in detail. We focus on the MapReduce programming model, and the NoSQL approach for handling data and queries.
(4) A critical evaluation regarding advantages and drawbacks of these new approaches, with respect to classical solutions, is given. Additionally, a compilation of the milestones on the topic are summarized and analyzed. Furthermore, new tools and platforms that have been designed as alternative to MapReduce will be enumerated.
(5) Finally, the discussion developed in this review is aimed for helping researchers to better understand the nature of Big Data. The recommendations given may allow the use of available resources, or the development of new ones for addressing this type of problems.

In order to reach the previous objectives, this article is structured as follows. First we provide a definition of Big Data, linking these concepts to DM and BI, and showing how this problem can be addressed. Next section introduces the main concepts of Cloud Computing, presenting an architecture approach to develop Big Data solutions on such platforms. Afterwards, we focus on the MapReduce programming framework as the most prominent solution for Big Data, describing its features and comparing it with some classical parallel approaches, as well as enumerating several limitations of this model. We then enumerate several alternatives to MapReduce that have been designed in order to overcome its performance under certain work scenarios. The lessons learned from this review are given in the next section. Finally, the main concluding remarks are presented.

ON THE SIGNIFICANCE OF BIG DATA IN BUSINESS INTELLIGENCE

In this section, we will first introduce what it is understood as Big Data. Then, we will establish the relationship between DM and BI for the sake of better understanding the significance of both facets with respect to scalability. Finally, we will present several guidelines that are necessary to address, in a proper way, the Big Data problem.

What is Big Data?

Recently, the term of Big Data has been coined referring to those challenges and advantages derived from collecting and processing vast amounts of data.[22] This topic has appeared as organizations must deal with petabyte-scale collections of data. In fact, in the last 2 years we have produced 90% of the total data generated in history.[23] The sources of such huge quantity of information are those applications that gather data from click streams, transaction histories, sensors, and elsewhere. However, the first problem for the correct definition of 'Big Data' is the name itself,[4] as we might think that it is just related to the data *Volume*.

The heterogeneous structure, diverse dimensionality, and *Variety* of the data representation, also have significance in this issue. Just think about the former applications that carry out the data recording: different software implementations will lead to different schemes and protocols.[24]

Of course it also depends on the computational time, i.e., the efficiency and *Velocity* in both receiving and processing the data. Current users demand a 'tolerable elapsed time' for receiving an answer. We must put this term in relationship with the available computational resources, as we cannot compare the power of a personal computer with respect to a computational server of a big corporation.[3]

Finally, one main concern with applications that deals with this kind of data is to maintain the *Veracity* of the information. This basically refers to the data integrity, i.e., avoiding noise and abnormalities within the data, and also the trust on the information used to make decisions.[25,26]

All these facts are known as the 4V's of Big Data,[1] which lead to the definition given by Steve Todd at Berkeley University[a]:

> Big data is when the normal application of current technology does not enable users to obtain timely, cost-effective, and quality answers to data-driven questions.

We must point out that additional definitions including up to 9V's can be also found, adding terms such as Value, Viability, and Visualization, among others.[27]

The main challenge when addressing Big Data is associated with two main features[28]:

- The storage and management of large volumes of information. This issue is related to DBMS, and the traditional entity-relation model. Commercial systems report to scale well, being able to handle multi-petabyte databases, but in addition to their 'cost' in terms of price and hardware resources, they have the constraint of importing data into a native representation. On the other hand, widely adopted open-source systems, such as MySQL, are much more limited in terms of scalability than their commercial analytics counterparts.
- The process for carrying out the exploration of these large volumes of data, which intends to discover useful information and knowledge for future actions.[23] The standard analytical processing is guided by an entity-relation scheme, from which queries were formulated using the SQL language. The first hitch of these type of systems is the necessity of preloading the data, as stated previously. Additionally, there is not much support for in-database statistics and modeling, and many DM programmers may not be comfortable with the SQL declarative style. Even in the case that engines provide these functionalities, as iterative algorithms are not easily expressible as parallel operations in SQL, they do not work well for massive quantities of data.

In summary, there are several conditions that must be taken into account in order to consider a problem within the Big Data framework. First of all, and referring to the 4Vs' properties, a threshold for the quantity of information that is being processed, and the time constraints for giving an answer, must be established. These two concepts are also closely related. For example, if we address the fingerprint recognition application,[29,30] there is a limit for the number of fingerprints we can manage in the database for providing an accurate answer within a short period of time, i.e., tenths of a second or few seconds.

But, how do we set this *limit*? The answer is unclear as what was 'big' years ago, can now be considered as 'small'. Therefore, for a clear definition of Big Data we must also include which technology is necessary to solve the problem. Suppose a *major* sales enterprise, which aims to adjust the unit pricing for a collection of items based on demand and inventory. Clearly, this firm will need a computational technology beyond a standard cluster of machines with a relational database and a common business analytics product. Now, if we consider a project of similar ambitions within the domain of a retailer company, the application could easily be completed using existing databases and ETL tools. The latter cannot be categorized as Big Data project, according to our definition.

Finally, Big Data is about the insight that we want to extract from information. There are many well-known applications that are based on Cloud Computing such as email servers (Gmail), social media (Twitter), or storage sharing and backup (Dropbox). All this software manage high volumes of data, where fast responses are essential, and with information coming at a high rate in a semi-structured or unstructured way. They must also face the veracity in the information; however, they are not intrinsically considered Big Data.

The key here is the analysis that is made for knowledge and business purposes, what is known as Data Science.[10,11] This speciality include several fields such as statistics, Machine Learning, DM, artificial intelligence, and visualization, among others. Hence, Big Data and Data Science are two terms with a high synergy between them.[31] Some well-known examples include e-Sciences[32] and other related scientific disciplines (particle physics, bioinformatics, and medicine or genomics) Social Computing[33] (social network analysis, online communities, or recommender systems), and large-scale e-commerce,[34,35] all of which are particularly data-intensive.

Regarding the former, many Big Data challenges proliferate nowadays for encouraging researchers to put their efforts in solving these kind of tasks. As examples, we may refer to the 'Data Mining Competition 2014',[b] which belongs to the *Evolutionary Computation for Big Data and Big Learning Workshop* (a part of the well-known GECCO conference), and the three 'DREAM9'[c] challenges opened by Sage Bionetworks and DREAM, linked to the *International Biomedical Commons Congress* and the *RECOMB/ISCB Systems and Regulatory Genomics/DREAM Conference*.

Data Science: Data Mining as a Support for Business Intelligence Applications to Big Data

With the establishment of the Internet, business in the 21st century is carried out within a digital environment, which is known as e-economy or digital economy. Information is a most valuable asset of corporations, but we must take into account that there is a handicap in their operations: each functional area of a company manages their own data. Because of this, it is necessary to integrate all information systems in the corporation, not only for processing this information in an efficient manner, but also to create some 'business intelligence' that can be used for all activities.

The *BI* concept is not a technology by itself, but rather a collection of information systems that works in a coordinate way. It includes several blocks such as data warehouse systems, Data Mining systems, Online Analytical Processing systems, knowledge-based systems, query and report tools, and dashboards, as depicted in Figure 1.

Organizations are looking for new and effective ways of making better decisions that allow them to gain competitive advantage. Therefore, they are in a need of some fundamental principles that guide the extraction of information. We must point out that decisions made by corporations using BI rely on the current activities from which they must capture the information. Among others, we must stress social media, user behavior logs on websites, transactional data, and so on. This implies huge volumes of data to be exploited and analyzed, and the trend is that this vast amount of information will continue growing.

Here is where the concept of Data Science comes up, which encompasses the application of DM techniques in Big Data problems,[36,37] as explained above. DM[38,39,6] consists of identifying valid and novelty patterns, which are potentially useful. It is built up by means of a set of interactive and iterative steps, where we can include preprocessing of data, the search of patterns of interest with a particular representation, and the interpretation of these patterns.

Principles and techniques of Data Science and DM are broadly applied across functional areas in

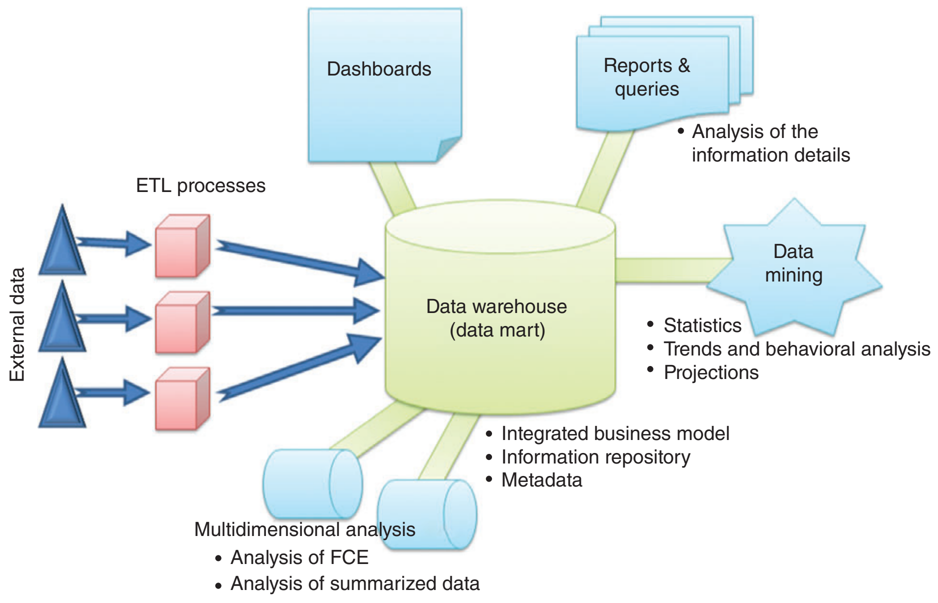
Dashboards
Reports & queries
• Analysis of the information details
ETL processes
External data
Data warehouse (data mart)
Data mining
• Statistics
• Trends and behavioral analysis
• Projections
• Integrated business model
• Information repository
• Metadata
Multidimensional analysis
• Analysis of FCE
• Analysis of summarized data

FIGURE 1 | Business Intelligence structure.

business. Many firms have differentiated themselves strategically thanks to the former, sometimes to the point of evolving into DM companies. We present several examples on how companies would benefit from this:

- The operational risk for setting prices for a retailer grows according to the time that is spent to recalculate markdown prices and tailor prices to individual stores and shopper profiles.
- For large investment banking firms, the speed and agility in the operations can give significant competitive advantages. As they need to recalculate entire risk portfolios, a good efficiency in the data treatment can enable the obtaining of fine-tune responses to changing interest rates, exchange rates, and counterpart risks.
- Insuring companies, and specifically actuaries, rely heavily on using historical data to predict future behavior or creating premium rates to price products. The growing volumes of available data limit the company at using only a subset of this information to generate pricing models.
- Finally, we must mention the case of unplanned growth in the corporate activities. The investment in infrastructure when the company expands its activities can be severe. The contrary case is also possible. In both scenarios, we must stress the benefits that come from a flexible and elastic solution.

The knowledge extraction process from Big Data has become a very difficult task for most of the classical and advanced existing techniques.[40] The main challenges are to deal with the increasing amount of data considering the number of instances and/or features, and the complexity of the problem.[41,42] From the previous examples, we can observe that several key concepts such as scalability, elasticity, and flexible pricing need to be considered in this scenario. Thus, it is straightforward to realize about the necessity of constructing DM solutions within a framework of the former characteristics in order to integrate them into BI applications.

How Can Big Data Problems be Addressed?

From the first part of this section we must recall that there are two main design principles for addressing the scalability in Big Data problems. First, we may refer to very large databases for the storage and management of the data. Furthermore, the processing and analysis must be carried out by parallel programming models.

Regarding the first issue, standard solutions in this field included distributed databases for intensive updating workloads,[43] and parallel database systems[44] for analytical workloads. While the former has never been very successful, the latter approach has been extensively studied for decades. Therefore, we may find several commercial implementations featuring well-defined schemas, declarative query languages, and a runtime environment supporting efficient execution strategies. When large volumes of data are involved, the hitch with these systems is twofold: (1) as they must run under high hardware requirements, it becomes prohibitively expensive when we scale to a cluster of computing elements; (2) the underlying assumptions of relational databases, i.e., fault tolerance, cannot be longer satisfied. Finally, we are aware that current applications are now managing semi-structured or even unstructured data, which imposes another challenge for database solutions.

According to these facts, an alternative to relational databases has arisen. This new data management technology is known as 'NoSQL',[12,45] which basically consists of storing the information as 'Key-Value' pairs, achieving a distributed horizontal scaling. An important difference between traditional databases and NoSQL databases is that they do not support updates and deletes. The major reason for the popularity of the NoSQL scheme is their flexible data model, and the support of various types of data models, most of these not being strict. With these arguments, there is a clear trend in migrating to this recent technology in Big Data applications.

Focusing on the programming models for data analysis, common solutions are those based on parallel computing,[46] such as the Message Passing Interface (MPI) model.[47] Challenges at this point rely on the data access and the ease of developing software with the requirements and constraints of the available programming schemes. For example, typical DM algorithms require all data to be loaded into the main memory. This imposes a technical barrier in Big Data problems as data are often stored in different locations and this supposes an intensive network communication and other input/output costs. Even in the case we could afford this, there is still the need for an extremely large main memory to hold all the preloaded data for the computation. Finally, there is a clear need in a robust fault-tolerant mechanism, as it is crucial for time-consuming jobs.

To deal with the previous issues, a new generation of systems has been established, where MapReduce[13] and its open-source implementation Hadoop[14,48] are the most representative ones in both industry and academia. This new paradigm removes the previous constraints for preloading data, fixed storage schemes, or even the use of SQL. Instead, developers write their programs under this new model that allows the system to automatically parallelize the execution. This can be achieved by the simple definition of two functions, named as Map and Reduce. In short, 'Map' is used for per-record computation, whereas 'Reduce' aggregates the output from the Map functions and applies a given function for obtaining the final results.

The success of these type of systems is also related to additional constraints that have been considered recently. Among them, we must stress low cost, security (considering technical problems), simplicity (in programming, installing, maintaining), and so on. According to the previous features, a new computational paradigm has imposed as the answer to all these issues. This system is Cloud Computing, and it has been settled as the baseline environment for the development of the aforementioned solutions for Big Data.

The concept of Cloud Computing allows several advantages to the perspective of deploying a huge cluster of machines configured such that the load can be distributed among them. The most relevant one is to rent the computational resources when they are strictly necessary. Hence, the cost of processing the data will be only spent when the data are ready to be processed, i.e., paying according to service quantity, type, and duration of the application service. We must be aware that, although the global amount of data in these cases actually exceeds the limits of current physical computers, the frequency of both the data acquisition and the data processing can be variable, thus stressing the goodness of Cloud Computing. Additionally, the inner structure of this computational paradigm in terms of data storage, data access, and data processing make it the most reasonable solution for addressing this problem.

CLOUD COMPUTING ENVIRONMENTS FOR BIG DATA

Cloud Computing is an environment based on using and providing services.[49] There are different categories in which the service-oriented systems can be clustered. One of the most used criteria to group these systems is the abstraction level that is offered to the system user. In this way, three different levels are often distinguished[50]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) as we can observe in Figure 2.

Cloud Computing offers scalability with respect to the use of resources, low administration effort, flexibility in the pricing model and mobility for the software user. Under these assumptions, it is obvious that the Cloud Computing paradigm benefits large projects, such as the ones related with Big Data and BI.[51]

In particular, a common Big Data analytics framework[52] is depicted in Figure 3. Focusing on the structure of the data management sector we may define, as the most suitable management organization architecture, one based on a four-layer architecture, which includes the following components:

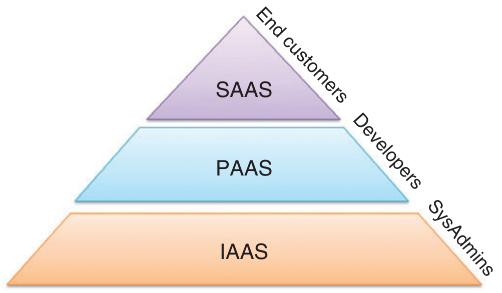

FIGURE 2 | Illustration of the layers for the Service-Oriented Architecture

- A file system for the storage of Big Data, i.e., a wide amount of archives of large size. This layer is implemented within the IaaS level as it defines the basic architecture organization for the remaining tiers.
- A DBMS for organizing the data and access them in an efficient way. It can be viewed in between the IaaS and PaaS as it shares common characteristics from both schemes. Developers used it to access the data, but its implementation lies on a hardware level. Indeed, a PaaS acts as an interface where, at the upper side offers its functionality, and at the bottom side, it has the implementation for a particular IaaS. This feature allows applications to be deployed on different IaaS without rewriting them.
- An execution tool to distribute the computational load among the computers of the cloud. This layer is clearly related with PaaS, as it is kind of a 'software API' for the codification of the Big Data and BI applications.
- A query system for the knowledge and information extraction required by the system's users, which is in between the PaaS and SaaS layers.

Throughout the following subsections, we will address the aforementioned levels in detail.

File System and Storage

The first level is the basis of the architecture of the Cloud system. It includes the network architecture with many loosely coupled computer nodes for providing a good scalability and a fault-tolerance scheme. As suggested, the system must consider a dynamic/elastic scheme where the performance, cost, and energy consumption of the node machines is managed in runtime.

Following this scheme, we need to define a whole system to manage parallel data storage of the large files from which the software will operate on. This is not trivial, and new file system approaches must be considered. Among all possible solutions, the one that has achieved a higher popularity is the Hadoop-distributed file system (HDFS).[53]

HDFS is the open-source project of the Apache Foundation that implements the Google file system, the initial solution conceived to deal with this problem.[54] An HDFS installation is composed of multiples nodes, which are divided into two classes: a master node (namenode) and a large number of fragments storages or datanodes. Files are divided into fixed size chunks of 64 megabytes, in a similar way than the clusters or sectors of traditional hard disk drives. Datanodes store these fragments, which are assigned with a unique ID label of 64 bits in the namenode when it is generated. When an application aims to read a file, it contacts with the namenode in order to determine where the actual data is stored. Then, the namenode returns the corresponding block ID and the datanode, such that the client can directly contact the data node for retrieving the data. An important feature of the design is that data are never moved through the

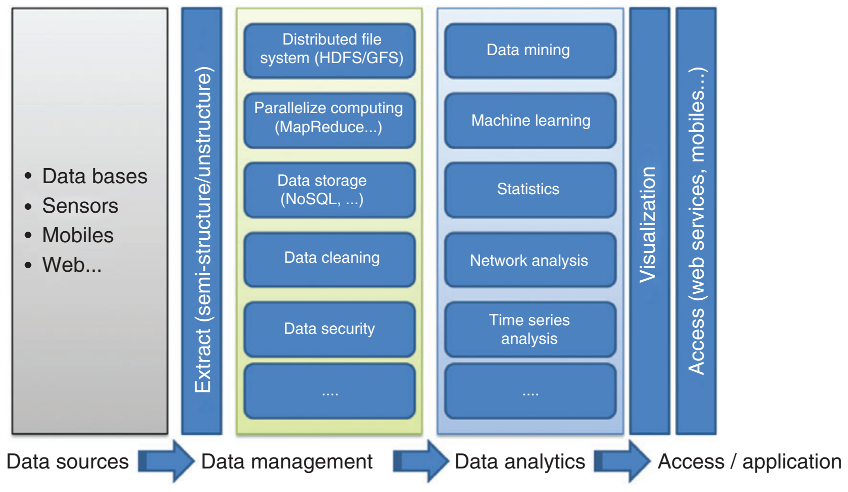

FIGURE 3 | Big Data framework.

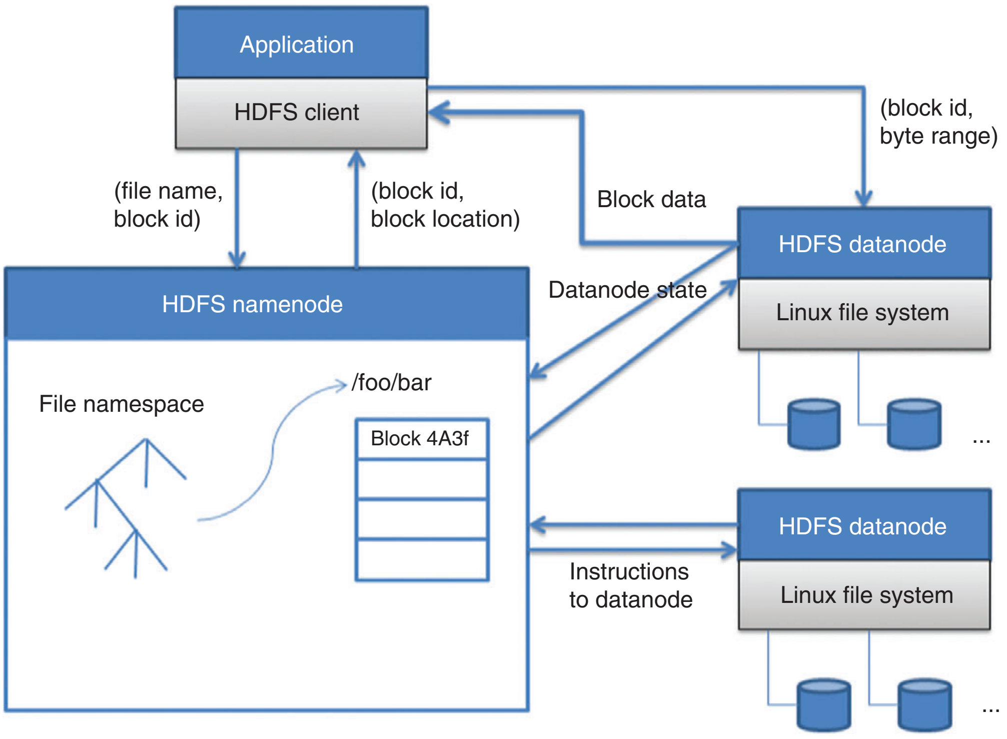

FIGURE 4 | The architecture of Hadoop-distributed file system (HDFS). The namenode (master) is responsible for maintaining the file namespace and directing clients to datanodes (slaves) that actually hold data blocks containing user data.

namenode. Instead, all data transfer occurs directly between clients and datanodes; communications with the namenode only involves transfer of metadata. This architecture is depicted in Figure 4.

In order to ensure reliability, availability, and performance according to an expected high demand of service, redundancy is maintained by default replicating each chunk in at least three servers of the Cloud, this being a parameter of the system. Usually two of them are set in datanodes that are located at the same rack and another on a different one. HDFS is resilient toward two common failure scenarios: individual datanode crashes and failures in networking equipment that bring an entire rack offline. The namenode communicates with datanodes to re-equilibrate data, move copies, and preserve the high replication of the data: if there are not enough replicas (e.g., due to disk or machine failures or to connectivity losses due to networking equipment failures), the namenode directs the creation of additional copies; if there are too many replicas (e.g., a repaired node rejoins the cluster), extra copies are discarded.

In summary, the HDFS namenode has the following responsibilities[14]:

- Namespace management: For a quick access, the namenode holds in memory all information regarding the directory structure, the location of the blocks and so on.
- Coordinating file operations: As pointed out previously, communication is made directly from clients to datanodes by the coordination of the namenode. Files are deleted by a lazy garbage collector.
- Maintaining overall health of the file system: The integrity of the system is kept by the creation of new replicas of the data blocks. When some datanodes have more blocks than others, a rebalancing process is applied.

The main difference with respect to other file systems is that HDFS is not implemented in the kernel space of the operating system, but it rather works as a process in the user space. As blocks are stored on standard file systems, HDFS lies on top of the operating system stack. Additionally, it is not frequent that the stored data are overwritten or removed; in general the files are read only or just new information is added to them.

Finally, there are other platforms that follows different implementations such as Amazon Simple Storage Service (S3),[55] Cosmos,[56] and Sector.[57,58] They aim at managing the information in a local way in order to avoid transactions through the net that can detriment the performance when dealing with executions on Big Data.

Database Management: Not Only SQL

The second layer is devoted to the DBMS. Relational databases may have difficulties when processing Big Data along a big number of servers, and keeping the efficiency required by BI applications with traditional database systems is complicated. Specifically, the storage format and the access to the data must be completely modified, implying the necessity of using different DBMS.

In this framework, NoSQL databases[12] emerge as a new kind of system. The main difference between this new system and the relational model is that they allow a horizontal scalability of the data, by means of variations on the storage and a retrieval of key/value pairs, as opposed to the relational model based on primary-key/foreign-key relationships.

Below we enumerate the basic features of this model:

- As its name indicates, they use a query language similar to SQL, but it is not fully conformant. As it is designed over a partitioned file system, JOIN-style operations cannot be used.
- The 'A.C.I.D.' guarantees (atomicity, consistency, isolation, and durability)[59] cannot be satisfied anymore. Usually, only an eventual consistency is given, or the transactions are limited to unique data items. This means that given a sufficiently long period of time in which no changes are submitted, all the updates will be propagated over the system.
- It has a distributed and fault-tolerant architecture. Data reside in several redundant servers, in a way that the system can be easily scalable adding more servers, and the failure of one single server can be managed without difficulties. This is the issue we stated at the beginning of this section regarding the horizontal scalability, the performance and the real time nature of the system being more important than the consistency.

In this way, NoSQL database systems are usually highly optimized for the retrieval and appending operations, and they offer little functionality beyond the register storing (key-value type), implying the necessity of *ad hoc* efficient join algorithms.[60,61] However, the reduced flexibility compared with relational systems is compensated by means of a significant gain in scalability and performance for certain data models.

In brief, NoSQL DBMS are useful when they work with a large quantity of data, and the nature of these data do not require from a relational model for their structure. No scheme is needed, the data can be inserted in the database without defining at first a rigid format of the 'tables'. Furthermore, the data format may change at any time, without stopping the application, providing a great flexibility. Finally, to reduce the latency and substantially enhance the data throughput, a transparent integrated caching is implemented in the memory system.

One of the first conceptual models of this type of DBMS is possibly BigTable.[45] This database engine was created by Google in 2004 with the aim of being distributed, high efficiency, and proprietary. It is built over the Google file system and works over 'commodity hardware'. In order to manage the information, tables are organized by different groups of columns with variable dimensions, one of them always being the timestamp to maintain the control version and the 'garbage collector'. They are stored as 'subtables', i.e., fragments of a unique table, from 100 to 200 megabytes each, that can also be stored in a compressed format. This disposition enables the use of a load balancing system, i.e., if a subtable is receiving too many petitions, the machine where it is stored can get rid of the other subtables by migrating them to another machine.

The Dynamo system[62] is also a milestone for NoSQL DBMSs. It was originally built to support internal Amazon's applications, and to manage the state of services that have very high reliability requirements. Specifically, a tight control over the tradeoffs between availability, consistency, cost-effectiveness, and performance is needed. The state is stored as binary objects (blobs) identified by unique keys, and no operations span multiple data items. Dynamo's partitioning scheme relies on a variation of a consistent hashing mechanism[63] to distribute the load across multiple storage hosts. Finally, in the Dynamo system, each data item is replicated at N hosts where N is a parameter configured 'per-instance'.

From these systems, several open-source implementations have been developed make these models accessible to worldwide Big Data users. In the following, we enumerate a nonexhaustive list of examples:

- HBase[64] is built over HDFS following the scheme of the BigTable approach, thus featuring compression, in-memory operations, and Bloom filters that can be applied to columns. Tables in HBase can serve as input and output for MapReduce jobs run in Hadoop and may be accessed through the Java API.

TABLE 1 | Design Decisions of NoSQL DBMS. CP stands for Consistency and Partition tolerance, and AP stands for Availability and Partition tolerance, regarding the CAP theorem.

System	Data Model	Consistency	CAP Options	License
BigTable	Column families	Eventually Consistent	CP	Google Proprietary Lic.
Dynamo	Key-value storage	Eventually Consistent	AP	Amazon Proprietary Lic.
HBase	Column families	Eventually Consistent	CP	Open source —Apache
Cassandra	Column families	Eventually Consistent	AP	Open source —Apache
Hypertable	Multidimensional Table	Eventually Consistent	AP	Open source —GNU
MongoDB	Document-oriented Storage	Optimistically Consistent	AP	Open source —GNU
CouchDB	Document-oriented Storage	Optimistically Consistent	AP	Open source —Apache

- Cassandra[65] brings together the BigTable features and the distributed systems technologies from Dynamo. It has a hierarchical architecture in which the database is based on columns (name, value, and timestamp). Columns are grouped in rows as 'Column families' (a similar relationship to the one between rows and columns in relational databases) with keys that map the columns in each row. A 'keyspace' is the first dimension of the Cassandra hash and is the container for column families. Keyspaces are of roughly the same granularity as a schema or database (i.e., a logical collection of tables) in a relational DBMS. They can be seen as a namespace for ColumnFamilies and are typically allocated as one per application. SuperColumns represent columns that have subcolumns themselves (e.g., Maps).
- In HyperTable,[66] data is represented in the system as a multidimensional table of information. HyperTable systems provide a low-level API and the HyperTable Query Language, which allows the user to create, modify, and query the underlying tables. The data in a table can be transformed and organized at high speed, as computations are performed in parallel and distributing them to where the data is physically stored.
- MongoDB[67] is another NoSQL-type scalable DBMS. It is written in C++ with MapReduce support for a flexible data aggregation and processing. It also includes a document-oriented storage, which offers simplicity and potential for JavaScript Object Notation (JSON)[68] (an alternative to XML) type documents with dynamic schemes.
- CouchDB[69] is a document-oriented DBMS that enables the use of queries and indexes using JavaScript following a MapReduce style, i.e., using JSON to store the data. A CouchDB document is an object that consists of named fields, such as strings, numbers, dates, or even ordered lists and associative maps. Hence, a CouchDB database is a flat collection of documents where each document is identified by a unique ID. Additionally, it offers incremental replicas with detection and resolution of bidirectional conflicts. The CouchDB document update model is lockless and optimistic. It is written in the Erlang language, thus providing a robust functional programming base for concurrent and distributed systems, with a flexible design for its scalability and ease to extend.

Finally, for the sake of comparison we show in Table 1 a summary of the main properties of the former systems.

Execution Environment

The third level is the execution environment. Owing to the large number of nodes, Cloud Computing is especially applicable for distributed computing tasks working on elementary operations. The best known example of Cloud Computing execution environment is probably Google MapReduce[13] (the Google's implementation of the *MapReduce* programming model)

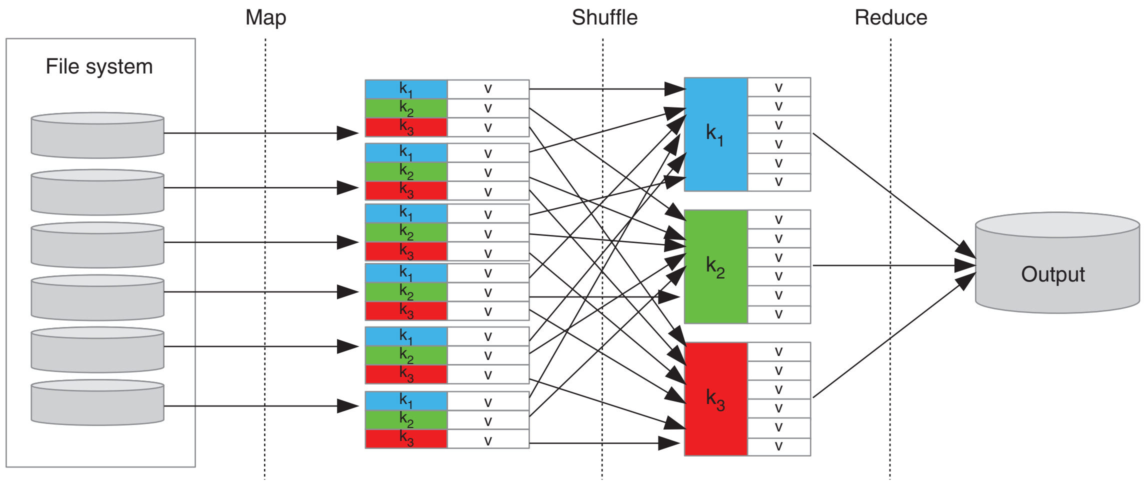

FIGURE 5 | MapReduce simplified flowchart.

and Hadoop, its open-source version.[14] This environment aims at providing elasticity by allowing the adjustment of resources according to the application, handling errors transparently, and ensuring the scalability of the system.

This system has been designed under the following assumptions: first, all the nodes in the cloud should be colocated (within one data center), or a high bandwidth is available between the geographically distributed clusters containing the data. Secondly, individual inputs and outputs to the cloud are relatively small, although the aggregate data managed and processed are very large.

As its name suggests, this programming model is built upon two 'simple' abstract functions named *Map* and *Reduce*, which are inherited from the classical functional programming paradigms. Users specify the computation in terms of a map (that specify the per-record computation) and a reduce (that specify result aggregation) functions, which meet a few simple requirements. For example, in order to support these, MapReduce requires that the operations performed at the reduce task to be both 'associative' and 'commutative'. This two-stage processing structure is illustrated in Figure 5.

We extend the description of this programming scheme in a separate section, where we will establish the properties of this approach for its use in Big Data problems.

Query Systems for Analysis

The last tier is related to the query systems, being the interface to the user and providing the transparency to the other tiers of the architecture. In environments where large databases exist, the necessity of carrying out 'complex' queries with answers in short time implies the use of a parallel model such as MapReduce, as explained in the previous subsection.

In most cases, for obtaining the required information from the data several sophisticated operations such as 'joins' or 'data filtering' need to be performed. Regarding the functional programming model of MapReduce, this task could become quite difficult and time-consuming to implement and it will require highly skilled developers.

According to the previous facts, several approaches have been developed to ease the user to obtain knowledge from the data in NoSQL-type databases. The goal of these systems is being able to provide a trade-off between the declarative style of SQL and the procedural style of MapReduce. This will diminish the efforts of implementing data analysis applications on top of a distributed system, i.e., adapting them toward Online Analytical Processing and Query/Reporting Data-Warehousing tools.

In the remainder of this section, we will present some well-known systems that allow analysts with strong SQL skills (but meager Java programming skills) the use of this type of query languages:

- **Hive** is a platform developed by Facebook that uses the HiveQL language, which is close to a SQL-like scheme.[70] It provides a subset of SQL, with features subqueries in the From clause, various types of joins, group-bys, aggregations, 'create table as select', and so on. Given a query, the compiler translates the HiveQL statement into a directed acyclic graph of MapReduce jobs, which are submitted to Hadoop for execution. Hive compiler may choose to fetch the data only from certain partitions, and hence, partitioning helps in efficiently answering a query. It supports all the major primitive types, as well as collection types such as map, list, and struct. Hive also includes a system catalogue, i.e., a meta-store, which contains schemas and statistics quite useful in data exploration, query optimization, and query compilation.

- **Pig** is a high-level scripting language developed by Yahoo to process data on Hadoop.[71] It is aimed at combining the high-level declarative queries (SQL) with the lower level programming procedure (MapReduce). Pig (via a language called Pig Latin) provides concise primitives for expressing common operations that performs a simple data transformation, i.e., projection, selection, group, join, and so on.[72] This conciseness comes at low cost: Pig scripts approach the performance of programs directly written in Hadoop Java.

Programs written in Pig only need to specify a query execution plan or a dataflow graph. The plan is compiled by a MapReduce compiler, which is then optimized once more by a MapReduce optimizer performing tasks such as early partial aggregation, and then submitted for execution.

Pig has a flexible data model that allows complex types such as set or map. Unlike Hive, stored schemas are optional. Pig also has the capability of incorporating user define functions. Finally, it provides a debugging environment that can generate sample data to help a user in locating any error made in a given script.

- **Jaql** is a functional data query language,[73] designed by IBM and built upon the JSON data model.[68] Jaql is a general-purpose dataflow language that manipulates semi-structured information in the form of abstract JSON values. It provides a framework for reading and writing data in custom formats, and provides support for common input/output formats like CSVs. In the same way as Pig and Hive, it provides significant SQL operators such as filtering, transformations, sort, group-bys, aggregation, and join.

Being constructed over JSON, Jaql is extendable with operations written in several programming languages such as Javascript and Python. Regarding data types, besides to the standard values it supports arrays and records of name-value pairs. It also comes with a rich array of built-in functions for processing unstructured or semi-structured data. Jaql also provides a user with the capability of developing modules, a concept similar to Java packages.

- **Dremel** architecture[74] works in a similar way as distributed search engines do, i.e., the query is managed by a serving tree, and it is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree. In contrast to Hive or Pig, Dremel does not translate these queries into MapReduce jobs.

The benefit of this model is the high efficiency achieved by the way the data is stored. In particular, it follows a column-striped storage representation, which enables it to read less data from secondary storage and reduce CPU cost due to cheaper compression. An extension to a nested-column scheme is developed for a faster access, similarly to JSON. There is an open-source framework for Dremel, which is developed under the Apache Project, known as Drill.[75]

- Scope is a stack of protocols developed by Microsoft in contraposition to the Apache Hadoop project.[76] This platform aims at merging distributed databases and MapReduce. It is developed for structured and unstructured data, with its own file management system and execution engine. Data are stored in a relational format within a distributed data platform, named Cosmos.[56]

The Scope scripting language resembles SQL (as in the case of Hive) but with integrated C# extensions. This fact allows users to write custom operators to manipulate rowsets where needed. A Scope script consists of a sequence of commands, which are data manipulation operators that take one or more row sets as input, perform some operation on the data, and output a row set. Every row set has a well-defined schema that all its rows must adhere to. Users can name the output of a command using an assignment, and an output can be consumed by subsequent commands simply by referring to it by name. Named inputs/outputs enable users to write scripts in multiple (small) steps, a style preferred by some programmers.

To summarize this subsection, we present in Table 2 the main characteristics for the previously introduced query models.

THE CENTRAL AXIS OF SCALABILITY: THE MAPREDUCE PROGRAMMING MODEL

In this section, our aim is to first go deeper on the description of parallel computation tier and to

TABLE 2 | Summary of the Characteristics of the Query Systems for Big Data

System	Developed by	Language	Type of Language	Data Structures Supported
Hive	Facebook	HiveQL	Declarative (SQL dialect)	Better suited for structured data
Pig	Yahoo!	Pig Latin	Data flow	Complex
Jaql	IBM	Jaql	Data flow	JSON, semi-structured
Dremel/Drill	Google/Apache	DrQL	Declarative (SQL dialect)	Structured and unstructured data
Scope	Microsoft	SQL/C#	Data flow	Structured and unstructured data

explain in detail the features of the MapReduce programming model.[77] Different implementations of the MapReduce framework are possible depending on the available cluster architecture.

We will focus on the Hadoop MapReduce implementation[14] for its wider usage and popularity due to its performance, open-source nature, installation facilities and its distributed file system. This fact is quite important to remark, as we may distinguish between MapReduce (the theoretical framework) and Hadoop MapReduce (the worldwide open-source implementation).

In order to do so, we will first describe the features and advantages of this programming model. Then, we will contrast its functionality versus some traditional approaches such as MPI and parallel databases. Next, we will introduce those key areas of DM that benefit the most from Big Data problems, and we describe several libraries that support MapReduce for solving these tasks. Finally, we will point out some current drawbacks of this approach, stressing several cases where researchers have reported that MapReduce is not the most appropriate solution.

Features and Advantages of Hadoop MapReduce

The MapReduce framework[13] was initially introduced by Google in 2004 for writing massive scale data applications. It was developed as a generic parallel and distributed framework, which allows to process massive amounts of data over a cluster of machines. In this section, we will first introduce the features of this model. Then, we will present the elements that compose a cluster running such a system. Afterwards, we will enumerate some of the goodness that have lead to the success of this new programming model.

Introduction to MapReduce

The MapReduce framework is based on the fact that most of the information processing tasks consider a similar structure, i.e., the same computation is applied over a large number of records; then, intermediate results are aggregated in some way. As it was previously described, the programmer must specify the Map and Reduce functions within a job. Then, the job usually divides the input dataset into independent subsets that are processed in parallel by the Map tasks. MapReduce sorts the different outputs of the Map tasks that become the inputs that will be processed by the Reduce task. The main components of this programming model, which were previously illustrated in Figure 5, are the following ones:

- The job input is usually stored on a distributed file system. The master node performs a segmentation of the input dataset into independent blocks and distributes them to the worker nodes. Next, each worker node processes the smaller problem, and passes the answer back to its master node. This information is given in terms of *<key,value> pairs* which form the processing primitives.
- The former input <key,value> pairs are split and distributed among the available *Map* tasks. Specifically, Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Then, Map functions emit a set of intermediate <key,values> pairs as output. Before the execution of a Reduce function, MapReduce groups all intermediate values associated with the same intermediate key (<key,list(values)> and transforms them to speed up the computation in the Reduce function.
- The intermediate values are supplied to the user's *Reduce* function via an iterator, which allows to handle lists of values that are too large to fit in memory. In particular, Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function [e.g., hash(key) module R]. The number of partitions (R) and the partitioning function are specified by the user. Finally, the Reduce

functions generate an arbitrary number of final <key,values> pairs as output.

The whole process can be summarized as follows: the master node collects the answers to all the subproblems, sorts the Map task outputs by their keys, groups those that have the same key, and shuffles them among the available Reduce tasks using a dynamic scheduling mechanism. In this approach, the runtime assigns Map/Reduce tasks to the available computation resources simplifying the optimal utilization of heterogeneous computational resources while the initial assignment of Map tasks is performed based on the data locality. This approach also provides an automatic load balancing for Map tasks with skewed data or computational distributions.

An illustrative example about how MapReduce works could be finding the average costs per year from a big list of cost records. Each record may be composed by a variety of values, but it at least includes the year and the cost. The Map function extracts from each record the pairs <year, cost> and transmits them as its output. The shuffle stage groups the <year, cost> pairs by its corresponding year, creating a list of costs per year <year, list(cost)>. Finally, the Reduce phase performs the average of all the costs contained in the list of each year.

Hadoop MapReduce Cluster Configuration

A full configured cluster running MapReduce is formed by a master-slave architecture, in which one master node manages an arbitrary number of slave nodes. The file system replicates the file data in multiple storage nodes that can concurrently access the data. As such cluster, a certain percentage of these slave nodes may be out of order temporarily. For this reason, MapReduce provides a fault-tolerant mechanism, such that, when one node fails, it restarts automatically the task on another node. In accordance with the above, a cluster running MapReduce includes several 'daemons' (resident programs) that will work in this server, namely the NameNode, DataNode, JobTracker, and TaskTracker:

- A server that hosts the NameNode is devoted to inform the client which DataNode stores the blocks of data of any HDFS file. As it is related to memory and I/O, this server does not hold any data, neither performs any computations to lower the workload on the machine. The node that hosts the NameNode is the most important one of the cluster, so if it fails, the complete system will crash.
- The DataNode daemon works on the slave machines of the cluster. When accessing an HDFS file, the client communicates directly with the DataNode daemons to process the local files corresponding to the blocks, which will be previously located by the NameNode. A DataNode may communicate also with other DataNodes for replicating its data blocks for the sake of redundancy.
- The JobTracker daemon is the linkage between the parallel application and Hadoop. There is just one JobTracker daemon running in the MasterNode. Its functionality consists of determining which files to process, assigning nodes to different tasks, and monitoring all tasks they are running.[d]
- In addition to the JobTracker, several TaskTrackers manage the execution of individual tasks on each slave nodes. There is a single TaskTracker per node, but each one of them can run many Java virtual machines for parallel execution. If the JobTracker does not receive any signal from the TaskTracker, it assumes that the node has crashed and it resubmits the corresponding task to other node in the cluster.[14]

The Success of MapReduce

We may observe a growing interest of corporations and researchers in Big Data analysis.[78] We have pointed out that the main reason is related to the large number of real applications that require scalable solutions such as sensor networks,[79] intrusion detection systems,[80] or Bioinformatics.[81–83]

This success of the MapReduce framework in this field is based on several issues that imply several advantages for the programmer, mainly facilitating parallel execution:

(1) *Fault-tolerant service*: In large clusters, machine failures should be assumed as common, such that active cluster nodes should be prepared for rebalancing the load. This issue is extremely significant for those DM tasks that require a long time to execute, where it is quite costly to restart the whole job.

For MapReduce, the master pings every Mapper and Reducer periodically. If no response is received for a certain time window, the machine is marked as failed. Ongoing task(s) and any task completed by the Mapper is reset back to the initial state and reassigned by the master to other machines from scratch. Completed Map tasks are re-executed on a failure because their output is stored on the local disk(s) of the failed machine and is therefore inaccessible. Completed Reduce tasks do not need to be re-executed as their output is stored in the global file system.

(2) *Fair task distribution*: The MapReduce system will determine task granularity at runtime and will distribute tasks to compute nodes as processors become available. Additionally, if some nodes are faster than others, they will be given more tasks.
(3) *Move processing to the data*: On grid computing it was common to have both 'processing' and 'storage' nodes linked together by a high-capacity interconnect. However, current applications on Big Data imply data-intensive workloads that are not very processor-demanding. This causes a bottleneck in the cluster network that degrades the system productivity.

The MapReduce alternative is running the code directly within the node where the data block is located, considering that the data is spread across the local disks of nodes in the cluster. The complex task of managing storage in such a processing environment is typically handled by a distributed file system that sits underneath MapReduce.

(4) *Trade latency for throughput*: By definition, data-intensive processing implies that the relevant datasets are too large to fit in memory and must be held on disk. Instead, a sequential process of the data into long streaming operations is carried out. Random accesses must be also avoided for all computations.
(5) *Hide system-level details from the application developer*: Distributed programming is quite challenging to write and debug as the code runs concurrently in unpredictable orders, and access data in unpredictable patterns. The idea behind MapReduce is to isolate the developer from system-level details maintaining a separation between which processes are going to be performed and how those computations are actually carried out on a cluster.

Therefore, the main advantage of this framework is that the programmer is only responsible for the former, i.e., they only need to focus on the Map and Reduce function, everything else being common to all programs. With this small constraint, *all of the details of communication, load balancing, resource allocation, job startup, and file distribution are completely transparent to the user*.

MapReduce Versus Traditional Parallel Approaches

The ubiquitous distributed-memory MPI[47] has been the *de facto* standard for parallel programming for decades. MPI supports a rich set of communication and synchronization primitives for developing parallel scientific applications. However, the challenge for MPI in Big Data applications is related to the application of the checkpointing fault-tolerance scheme, as it is challenging at extreme scale due to its excessive disk access and limited scalability.[84,85]

MPI and MapReduce are not so different regarding their implementation. For example, the intermediate data-shuffle operation in MapReduce is conceptually identical to the familiar *MPI_Alltoall operation*. But in spite of the features they have in common, MPI lacks from the benefit of MapReduce on less reliable commodity systems.[86] Regarding this fact, there have been several efforts for migrating classical MPI-based software to MapReduce. One of the pioneers works on the topic considered the implementation of MapReduce within a Python wrapper to simplify the writing of user programs.[87] Other approaches have been developed under a C++ library for its use in graph analytics.[88]

Despite of the previous fact, we must take into account that any MapReduce application is preferred to an MPI implementation only when it accomplishes two simple conditions[89]: (1) input data must be Big; (2) the overhead due to a 'bad' codification of the Map and Reduce functions should be minimal. This latter issue refers to the use of those optimization mechanisms that are available in MapReduce,[90] and also to a proper implementation in which both functions balance in an appropriate way the workload.

Focusing on DBMSs, traditional relational databases seem to be relegated to the status of legacy technology. But, if these databases do not scale adequately, how is it possible that the biggest companies are still making use of these systems? The answer is that commercial systems can actually scale to stunning amount of transactions per seconds. However, these are expensive solutions and they need high-end servers (even more expensive) to work properly. Hence, there is an unfulfilled need for a distributed, open-source database system.[4]

Nevertheless, regarding usability purposes, DBMSs do also a great job of helping a user to maintain a dataset, adding new data over time, maintaining derived views of the data, evolving its schema, and supporting backup, high availability, and consistency in a variety of ways.[3] Even more: some of the advantages enumerated for key-value storage (NoSQL), such as schema later, or better availability and faster response time through relaxed consistence (as explained in the *Database Management* section), can be also partially accomplished by traditional DBMS with respect to binary large objects, and the support of semi-structured data in XML format.

When performing some well-structured analytical tasks, the goodness of DBMS over MapReduce and NoSQL approaches is related to the lack of indexes and a declarative language of the latter.[4] There are many cases, especially for Map functions, for which the function is too complicated to be expressed easily in a SQL query, such as fault-tolerant parallel execution of programs written in higher-level languages (such as Pig Latin[71]) across a collection of input data.[90]

However, this can be viewed as a double-edged sword depending on the system's user. For example, Machine Learning scientists may find unnatural (if not impossible!) to develop their scripts into a declarative way. Hence, they might prefer more conventional, imperative ways of thinking about these algorithms. Other programmers may find that writing the SQL code for each task is substantially easier than writing MapReduce code.[91]

In spite of all the aforementioned issues, MapReduce excels in several factors.[90] For example, a single MapReduce operation easily processes and combines data from a variety of storage systems. Tools based on MapReduce provide a more conventional programming model, an ability to get going quickly on analysis without a slow import phase, and a better separation between the storage and execution engines. On the contrary, the input for a parallel DBMS must first be copied into the system, which may be unacceptably slow, especially if the data will be analyzed only once or twice after being loaded. In particular, a benchmark study using the popular Hadoop and two parallel DBMSs,[91] showed that the DBMSs were substantially faster than the MapReduce system *only* once the data is loaded.

According to these facts, MapReduce-based systems may offer a competitive edge with respect to traditional parallel solutions in terms of performance, as they are elastically scalable and efficient. Specifically, we have highlighted their goodness versus MPI and DBMS, as MapReduce systems are able to provide the functionality for both doing 'traditional' SQL-like queries for analysis, e.g., using systems such as Hive or Pig, and also for automated data analysis such as DM.

Data Processing Algorithms for Big Data with MapReduce Open-Source Libraries

It is straightforward to acknowledge that, the more data we have, the more insight we can obtain from it. However, we have also pointed out that current implementations of data processing algorithms suffer from the curse of dimensionality and they can also scale to certain problems. Standard DM algorithms require all data to be loaded into the main memory, becoming a technical barrier for Big Data because moving data across different locations is expensive, even if we do have enough main memory to hold all data for computing.

Because of the former, many efforts have been carried out to develop new approaches, or adapt previous ones, into the MapReduce parallel programming model, as well as providing a Cloud Computing platform of Big Data services for the public.[23] In general, when coding an algorithm into MapReduce, the Map function is commonly used as simply dividing the initial problem into different subsets. However, this process may create small datasets with lack of density. This can be considered as a hitch for some problems such as classification. On the contrary, the Reduce function is considered as the most creative stage. This is due to the fact that it is necessary to build a global quality model associated to the problem that is aimed to solve.

Regarding the structure of Big Data problems, the list DM areas with this type of applications exist essentially covers all major types. For the sake of clarity, we focus on those that have attracted the highest interest from researchers. Specifically, these are related to classification, frequent pattern mining, clustering, and recommender systems:

- Classification techniques decide the category a given 'object' belongs to with respect to several input attributes.[92,93] It mainly works by determining whether a new input matches a previously observed pattern or not, i.e., matching a learned discrimination function.

In this area, several implementations have been proposed for different classification architectures such as Support Vector Machines,[94] Fuzzy Rule-Based Classification Systems,[95] Hyper Surface Classification,[96] rough sets,[97] and ensembles of classifiers.[98,99] Additionally, there are many applications in engineering that directly benefit from parallel classification models in MapReduce. Among them, we may stress fault detection and diagnosis,[100,101] remote sensing images,[102] and Bioinformatics.[103–106]

- Frequent pattern mining, also known as association rule mining,[107,108] discovers interesting relations among items (objects or variables) in large databases. It basically aims at finding groups of elements that commonly appear 'together' within the database and that can be transformed into rules of causality $A \Rightarrow B$, i.e., when element A shows up, then it is likely to have also B.

For Market Basket Analysis, two extensions from the well-known Apriori algorithm to the MapReduce framework have been proposed in Ref 109 We must also stress the 'Plute' algorithm, developed for mining sequential patterns.[110]

- Clustering techniques, as its name suggests, aim to group a large number of patterns according to their similarity.[111,112] In this way, the clustering identifies the structure, and even hierarchy, among a large collection of patterns.

Several clustering approaches have been shown to be properly adapted to the MapReduce framework. In Ref 113, Particle Swarm Optimization for defining the cluster parameters is considered. In Ref 114 DBCURE-MR, a parellelizable density-based clustering algorithm is proposed. Finally, the Cop-Kmeans method with cannot-link constraints is presented in Ref 115.

- Recommender systems are designed for inferring likes and preferences and identify unknown items that are of interest.[116,117] They are mostly observed in online systems such as in e-commerce services, social networks, and so on.

Among the examples of these kinds of systems, we may stress TWILITE,[118] which helps people find users to follow for the Twitter application. We may also find a TV program recommender system,[119] and a system based on collaborative users applied in movies databases proposed in Ref 120.

There is a myriad of different implementations that may overwhelm any interested researcher to address Big Data problems. Fortunately, several projects have been built to support the implementation and execution of DM algorithms in a straightforward way. Among them, the most relevant one is possibly the Mahout Machine Learning library.[121] It contains scalable Machine Learning implementations written in Java and built mainly upon Apache's Hadoop-distributed computation project, which was previously described in detail.

First of all, we must point out two significant issues: (1) On the one hand, it is just a library. This means that it does not provide a user interface, a prepackaged server, or an installer. It is a simply framework of tools intended to be used and adapted by developers. (2) On the second hand, it is still under development (currently is under version 0.9). It is a quite recent approach and it is far from being thoroughly documented. However, it includes a wide amount of methods and it is continuously in expansion with new DM approaches. Currently, it supports the aforementioned main kind of learning tasks. It addition, it includes dimension reduction techniques, and other miscellaneous approaches, all of which are summarized in Table 3.

In April 2014, Mahout has said goodbye to MapReduce. The justification for this change is twofold: on the one hand organizational issues, as it was onerous to provide support for scalable ML; on the other hand, technical considerations, as MapReduce is not so well suited for ML, mainly due to the launching overhead, especially noticeable for iterative algorithms, different quality of implementations, and/or and the unsuitability of some methods to be codified as Map and Reduce operations.

Hence, the future of Mahout is the usage of modern parallel-processing systems that offer richer programming models and more efficient executions, while maintaining the underlying HDFS. The answer to these constraints is Apache Spark,[122] which will be also described in detail in the following sections. In this way, future implementations will use the DSL linear algebraic operations following Scala & Spark Bindings. These are basically similar to R (Matlab)-like semantics, allowing automatic optimization and parallelization of programs.

Although Mahout has gained great popularity among DM practitioners over Big Data problems,[123] it is not the unique solution for this type of tasks. In this review we would like to stress five additional open-source approaches as an alternative to Mahout for data processing algorithms:

- NIMBLE[124] allows the development of parallel DM algorithms at a higher level of abstraction, thus facilitating the programmer using reusable (serial and parallel) building blocks that can be efficiently executed using MapReduce. Furthermore, NIMBLE provides built-in support to process data stored in a variety of formats; it also

TABLE 3 | DM Tasks and Algorithms Implemented in the Mahout Software Tool Version 0.9

Type of Task	List of Algorithms
Classification	Naive Bayes/Complementary Naive Bayes
	Multilayer perceptron
	Random forest
	Logistic regression
	Hidden Markov models
Clustering	Canopy clustering
	k-means clustering
	Fuzzy k-means
	Streaming k-means
	Spectral clustering
Collaborative filtering	User-based collaborative filtering
	Item-based collaborative filtering
	Matrix factorization with alternating least squares
	Weighted matrix factorization, SVD++, Parallel SGD
Dimension reduction	Singular value decomposition
	Stochastic singular value decomposition
	Principal components analysis
	Lanczos algorithm
Topic models	Latent Dirichlet allocation
Miscellaneous	Frequent pattern mining
	RowSimilarityJob—compute pairwise—similarities between the rows of a matrix
	ConcatMatrices—combine two matrices—or vectors into a single matrix
	Collocations—find colocations of tokens in text

allows facile implementation of custom data formats.

- SystemML[125] is similar to NIMBLE. It allows to express algorithms in a higher-level language, and then compiles and executes them into a MapReduce environment. This higher-level language exposes several constructs including linear algebra primitives that constitute key buildings blocks for a broad class of supervised and unsupervised DM algorithms.
- Ricardo is a system that integrates R (open-source statistical analysis software) and Hadoop.[126] The main advantage of this approach is that users who are familiarized with the R environment for statistical computing, can still used the same functionality but with larger data, as it supports the data management capabilities of Hadoop.
- Rhipe stands for R and Hadoop Integrated Programming Environment.[127,128] Its main goals are (1) not losing important information in the data through inappropriate data reductions; (2) allowing analysis exclusively from within R, without going into a lower level language.
- Wegener et al.[129] achieved the integration of Weka[6] (an open-source Machine Learning and Data Mining software tool) and MapReduce. Standard Weka tools can only run on a single machine, with a limitation of 1-GB memory. After algorithm parallelization, Weka breaks through the limitations and improves performance by taking the advantage of parallel computing to handle more than 100-GB data on MapReduce clusters.

To conclude this section, it is worth to point out that we are attending a generational shift regarding ML libraries and software tools,[130] which is summarized in Figure 6. The first generation comprises the traditional tools/paradigms such as SAS, R, or KNIME, which are typically only vertically scalable. In this section we have focused on the second generation approaches, which work over Hadoop/MapReduce. However, we must be aware of a third generation that involves those programming frameworks that go beyond MapReduce such as Spark. The basis of these novel models will be introduced in the next Sections.

We must also point out that there are also several open-source and proprietary platforms that have been used in business domains such as Pentaho Business Analytics, Jaspersoft BI suite, and Talend Open Studio among others,[131] but they are out of the scope of this work.

Beyond MapReduce: Global Drawbacks of the MapReduce Approach

We have stressed the goodness of MapReduce as a powerful solution designed to solve different kinds of Big Data problems. However, it is not a panacea: there are some scenarios in which this functional programming model does not achieve the expected performance, and therefore alternative solutions must be chosen.

The first drawback is straightforward: it does not provide any significant improvement in performance when the work cannot be parallelized, according to Amdahl's law.[132] But there are some other issues that

Generation	1st Generation	2nd Generation	3rd Generation
Examples	SAS, R, Weka, SPSS, KNIME, KEEL...	Mahout, Pentaho, Cascading	Spark, Haloop, GraphLab, Pregel, Giraph, ML over Storm
Scalability	Vertical	Horizontal (over Hadoop)	Horizontal (beyond Hadoop)
Algorithms available	Huge collection of algorithms	Small subset: sequential logistic regression, linear SVMs, Stochastic Gradient Descendent, k-means clustering, Random forest, etc.	Much wider: CGD, ALS, collaborative filtering, kernel SVM, matrix factorization, Gibbs sampling, etc.
Algorithms Not available	Practically nothing	Vast no.: Kernel SVMs, Multivariate logistic Regression, Conjugate Gradient Descendent, ALS, etc.	Multivariate logistic regression in general form, k-mean clustering, etc. – Work in progress to expand the set of available algorithms
Fault-tolerance	Single point of failure	Most tools are FT, as they are built on top of Hadoop	FT: Haloop, Spark Not FT: Pregel, GraphLab, Giraph

FIGURE 6 | Machine Learning software suites: a three-generational view.

should be taken into account prior to the migration this type of systems, which are enumerated below:

- Not all algorithms can be efficiently formulated in terms of Map and Reduce functions, as they are quite restrictive.

In particular, among all possible deficiencies of MapReduce, the greatest critic reported is the implementation of *iterative jobs*.[133] In fact, many common Machine Learning algorithms apply a function repeatedly to the same dataset to optimize a parameter. The input to any Hadoop MapReduce job is stored on the HDFS. Hence, whenever a job is executed, the input has to be reload the data from disk every time. This is done regardless of how much of the input has changed from the previous iterations, incurring in a significant performance penalty.

- MapReduce has also some problems in processing networked data, i.e., graph structures.[134]

The main difference in the processing of regular data structures (tables) and relational models (graphs) relies on different problem decompositions. Table structures are simpler as the algorithm must only process individual records (rows). However, for the networked data, single processing of a graph vertex usually requires access to the neighborhood of this vertex. As, in most cases, the graph structure is static, this represents wasted effort (sorting, network traffic, among others), as it must be accessed at every iteration via the distributed file system, showing the same behavior as in the previous case.

- Processing a lot of small files, or performing intensive calculations with small size data is another issue.

As stated previously, for each MapReduce job that is executed, some time is spent on background tasks, incurring in high startup costs (in Hadoop, they can be tens of seconds on a large cluster under load). This places a lower bound on iteration time.[135]

- Regarding interactive analytics, it has not shown a good behavior when processing transactions.

The hitch here is that this considers random access to the data. MapReduce is often used to run *ad hoc* exploratory queries on large datasets, through SQL interfaces as explained in previous Sections. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with MapReduce each query incurs significant latency (i.e., tens of seconds) because it runs as a separate MapReduce job and reads data from disk.[122] Therefore, its usage is not adequate for low latency data access. Even worse, in some cases there are very high memory requirements, as all the data used for local processing must fit in local memory.

- We must also point out security and privacy issues, not because it is an inner problem of MapReduce, but because it must be implemented within a Cloud Computing environment.

Users are reluctant to the use of a public cloud as they cannot be sure that it guarantees the confidentiality and integrity of their computations in a way that is verifiable.[25] In addition to security challenges, there remains the issue of accountability with respect to an incorrect behavior. If data leaks to a competitor, or a computation returns incorrect results, it can be difficult to determine whether the client or the service provider are at fault.[26]

- Finally, we must stress the complexity of installing and having ready a complete cluster with a complete stack of protocols, such as the one described within the Cloud Computing Section. Fortunately, this issue may be overcome making use of Cloudera[e],[136] a system that provides an enterprise-ready, 100% open-source distribution that includes Hadoop and related projects.

A 'PLAN-B' FOR MAPREDUCE: PROGRAMMING FRAMEWORKS

According to the issues that were raised in the previous part of the section, several alternatives have been proposed in the specialized literature. Depending on the features of the problem we are dealing with, we may choose the one that better suits our necessities.[137]

Considering the previous comments, in this section we provide a representative list of programming frameworks/platforms that have been either adapted from the standard MapReduce, or developed as new framework, aiming to achieve a higher scalability for Big Data applications.[130] A summary of these tools is shown in Figure 7, where we divide each platform with respect to their paradigm and use.

In the following, we describe them according to this division structure, i.e., horizontal and vertical.

Directed Acyclic Graph Model

The Directed Acyclic Graph (DAG) model defines the dataflow of the application, where the vertices of the

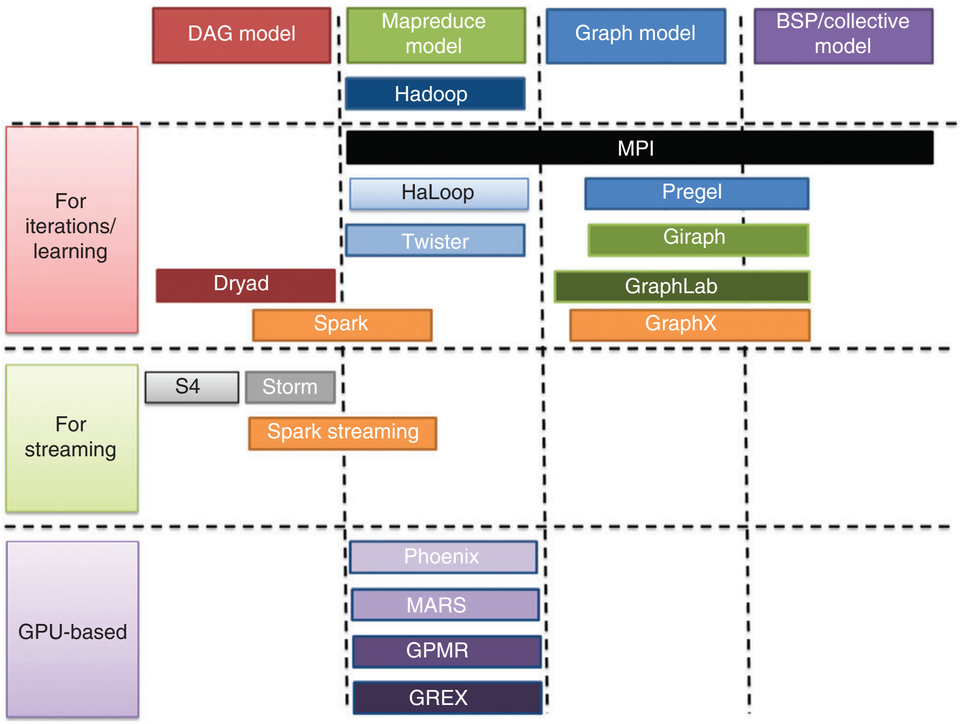

FIGURE 7 | Alternative frameworks for the standard MapReduce model.

graph define the operations that are to be performed on the data. The 'computational vertices' are written using sequential constructs, devoid of any concurrency or mutual exclusion semantics.

The main exponent of this model is **Dryad**,[138] a Cloud Computing framework proposed by Microsoft Research as a general-purpose execution environment for distributed, data-parallel applications. In this framework, applications are modeled as directed acyclic graphs, where each vertex is a program and the edges represents the data channel.

As the core of MapReduce, it allows automatic management of scheduling, distribution, and fault tolerance. DryadLINQ[139] is the base programming API for Dryad and hence it is more suitable for applications that process structured data. It generates Dryad computations from the LINQ Language-Integrated Query extensions to C#.

Iterative MapReduce

We have pointed out that one of the main drawbacks of MapReduce is that is not so well suited for iterative algorithms, mainly due to the launching overhead that is present even if the same task has been carried out already. For this reason, there has been many efforts to develop new approaches that can address this issue, which we present below:

- **Haloop**[140] is a new programming model and architecture that provides support for iterative algorithms by scheduling tasks across iterations in a manner that exploits data locality, and by adding various caching mechanisms. Its main contribution is the reuse of Mappers and Reducers when they do the same job. It also implements planning loops, i.e., tasks are assigned to the same nodes at every turn. By caching loop invariants no resources are spent by reloading repeated information several times. There is also a local 'Reducer' to compare the loop condition efficiently. It maintains the fault tolerance of the standard MapReduce and a compatibility with Hadoop.
- **Twister**[141] is a distributed in-memory MapReduce system with runtime optimized operations for iterative MapReduce computations, from which intermediate data are retained in memory if possible, thus greatly reducing iteration overhead.

Its main features are the following ones: (1) addition of a 'configure' phase for loading static data in both Map and Reduce tasks; (2) using a higher granularity for the Map tasks; (3) a new 'Combine' operation that acts as another level of reduction; and (4) the implementation of a set of programming extensions to MapReduce, such as *MapReduceBCast*(*Value value*), which facilitates sending a set of parameters, a resource name, or even a block of data to all Map tasks.

The disadvantage in this case is that, as it requires data to fit into the collective memory of the cluster in order to be effective, it cannot cope with jobs that require the processing of terabytes of data. Another disadvantage of Twister is its weak fault tolerance compared to Hadoop.

- **Spark**[122] and **Spark2**[142] are developed to overcome data reuse across multiple computations. It supports iterative applications, while retaining the scalability and fault tolerance of MapReduce, supporting in-memory processes.

With this aim, the authors include a new abstraction model called Resilient Distributed Datasets (RDDs), which are simply a distributed collection of items. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators. In a nutshell, it provides a restricted form of shared memory based on coarse-grained transformations rather than fine-grained updates to shared state. RDDs can either be cached in memory or materialized from permanent storage when needed (based on lineage, which is the sequence of transformations applied to the data). However, Spark does not support the group reduction operation and only uses one task to collect the results, which can seriously affect the scalability of algorithms that would benefit from concurrent Reduce tasks, with each task processing a different subgroup of the data

Similar to Mahout for Hadoop,[121] Spark implements a Machine Learning library known as MLlib, included within the MLBase platform.[143] MLlib currently supports common types of Machine Learning problem settings, as well as associated tests and data generators. It includes binary classification, regression, clustering, and collaborative filtering, as well as an underlying gradient descent optimization primitive.

Bulk Synchronous Parallel/Graph Model

The Bulk Synchronous Parallel (BSP) model[144] is a parallel computational model for processing iterative graph algorithms. With this scheme, computations are 'vertex-centric' and user defined methods and

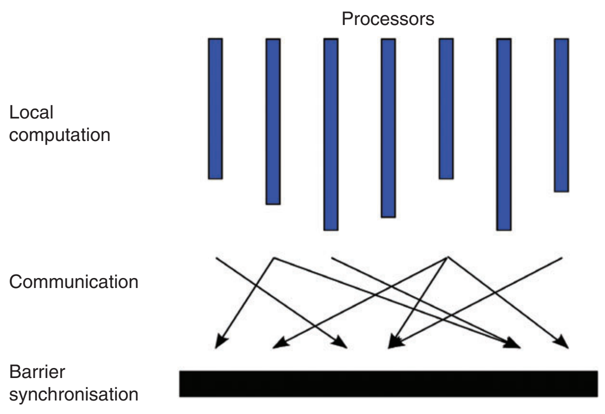

FIGURE 8 | BSP model workflow.

algorithms proceed in supersteps concurrently, with synchronization barriers between each one of them. Each vertex has a state and is able to receive messages sent to it from the other vertexes in the previous superstep. A superstep consists of three ordered stages (Figure 8):

(1) Concurrent computation: computation on locally stored data.
(2) Communication: messages in a point-to-point manner (collective).
(3) Barrier synchronization: wait and synchronize all processors at end of superstep.

In this way, more complex graph models allow a better representation of the dataflow of the application, i.e., cyclic models define that iterativity. For this reason, they are quite popular for some Big Data problems, such as social networks. A notorious example is given by Facebook to analyze the social graph formed by users and their connections (distance between people). Below we enumerate some BSP implementations:

- **Pregel**[145] was first developed by Google based on the premise that there are many practical computing problems that concern large graphs, such as the Web graph and various social networks. Therefore, it is not based on the MapReduce model, but it implements the BSP model. While the vertex central approach is similar to the MapReduce Map operation which is locally performed on each item, the ability to preserve the state of each vertex between the supersteps provides the support for iterative algorithms. In this implementation, all states including the graph structure, are retained in memory (with periodic checkpoints).
- **Giraph**[146] is an open-source graph-processing framework that implements the BSP computation. It includes a Hadoop Map-only job, from which the graph is loaded and partitioned across workers, and the master then dictates when workers should start computing consecutive supersteps. Therefore, it can be run on any existing Hadoop infrastructure, providing the API and middleware of Pregel, as well as adding fault-tolerance, and in-memory processing.
- **GraphX**[147] extends Spark with a new graph API. Its main goal is to unify graph-parallel and data-parallel computations in one system with a single composable API.
- **GraphLab**[148] is a project started at Carnegie Mellon University in 2009 designed to efficiently run in both shared and distributed-memory systems. It is basically a graph-based, high performance, distributed computation framework in an optimized C++ execution engine. Graphlab provides MapReduce-like operations, called Update and Sync functions. The Update function is able to read and modify overlapping sets of data, whereas the Sync function can perform reductions in the background while other computation is running. GraphLab uses scheduling primitives to control the ordering in which update functions are executed. It also includes powerful ML toolkits for topic modeling, graph analytics, clustering, collaborative filtering, graphical models, and computer vision, among others.

Stream Processing

The management of data streams is quite important as these types of applications are typically under the umbrella of Big Data. In what follows, we enumerate some of the most significant approaches for stream processing:

- **Storm**[149] in an open-source project designed for real-time processing of streaming data, in contrast to Hadoop which is for batch processing. To implement real-time computation on Storm, users need to create different topologies, i.e., a graph of computation created and submitted in any programming language. There are two kinds of node in topologies, namely, spouts and bolts. A spout is one of the starting points in the graph, which denotes source of streams. A bolt processes input streams and outputs new streams. Each node in a topology contains processing logic, and links between nodes indicate how data should be processed between nodes. Therefore, a topology is a graph representing the transformations of the stream, and each node in the topology executes in parallel.

- **S4**[150] is a general-purpose, distributed, scalable, fault-tolerant, pluggable open-source computing platform for processing continuous unbounded streams of data. S4 provides a runtime distributed platform that handles communication, scheduling, and distribution across containers. Applications are built as a graph of processing elements, which communicate asynchronously by sending events on streams that interconnect them. Events are dispatched to nodes according to their key. The implementation of a S4 job is designed to be modular and pluggable for easily and dynamically processing large-scale stream data.
- **Spark Streaming**[151] is an extension of the core Spark API that enables stream processing of live data streams. It features 'DStream', represented as a sequence of RDDs. As it is integrated under the Spark framework, it can handle Spark's learning and graph-processing algorithms.

MapReduce on Graphics Proccesing Units

Although we have focused on the Cloud Computing environments for developing Big Data applications, it is also possible to address this task using the capabilities of graphics processing units (GPUs). Next, we show some proposals that get advantage of this:

- **Phoenix**[152] implements MapReduce for shared-memory systems. Its goal is to support efficient execution on multiple cores without burdening the programmer with concurrency management. Because it is used on shared-memory systems it is less prone to the problems we encountered with iterative algorithms as long as the data can fit into memory.
- **MARS**[153] was created as an alternative to Phoenix for the execution of MapReduce in a GPU or multi-core environment.[154] The motivation is that GPUs are parallel processors with higher computation power and memory bandwidth than CPUs, also improving at a higher rate.[155] Thus, it follows a similar scheme to MapReduce but adapted for a general-purpose GPU environment, which implies several technical challenges.
- **GPMR**[156] is MapReduce library that leverages the power of GPU clusters for large-scale computing. GPMR allows flexible mappings between threads and keys, and customization of the MapReduce pipeline with communication-reducing stages (both PCI-e and network).
- **GREX**[157] uses general-purpose GPUs for parallel data processing by means of the MapReduce programming framework. It supports a parallel split method to tokenize input data of variable sizes in parallel using GPU threads. It also evenly distributes data to Map/Reduce tasks to avoid data partitioning skews. Finally, it provides a new memory management scheme to enhance the performance by exploiting the GPU memory hierarchy.

LESSONS LEARNED

In this article we have identified the problem that arises in the management of a large amounts of fast and heterogenous data in DM, which is known as the Big Data Problem. This situation is present in a great number of applications, especially for those based on the use of BI. We have pointed out the handicap to efficiently manage this quantity of information with standard technologies, both regarding storage necessities and computational power. This fact makes clear that a new paradigm must be provided in order to support the scalability of these systems: Cloud Computing.[158] Although it is possible to deploy Big Data technologies in a noncloud cluster of computers, we must excel the properties of these systems for easing the development of these tasks.

Throughout this article we have described the structure and properties of a Cloud Computing environment and we have focused on the execution engine for allowing the scalability in the computation of DM algorithms, i.e., the MapReduce approach. From the realization of this thorough study, we may emphasize five important lessons learned that may help other researchers to understand the intrinsic features of this framework:

1. There is a growing demand in obtaining 'insight' from as much amount of information as possible, but we must be careful about what we are actually able to take on. In order to process Big Data, we must take into account two main requirements: (1) to deploy a proper computational environment, either a private or public cloud; and (2) technical staff for developing the software needed for the DM process.

In the former case, although we acknowledge that the majority of the machines can be built on commodity hardware (scaling-out instead of scaling-up), the master node of the system should be robust enough to support the required scalability, i.e., number of jobs; and the costs of the Information Technology crew for installing and maintaining such a cluster are not trivial. To ease this task, the Cloudera project[136] has established as an enterprise-ready source distribution that includes Hadoop and related projects. In the case of a public cloud, users can easily deploy Hadoop on Amazon EC2 and Microsoft Azure cloud computing platforms, by using Amazon Elastic MapReduce[159] and HDInsight.[160] The hitch here is that we must face security and accountability of the system, as the main issues users are reluctant with.

However, maybe the most important point is related to the people that must manage the data and translate it into useful information. Currently, being a Data Scientist is undoubtedly the most demanding career regarding its ability in managing statistics and Machine Learning algorithms. There is a necessity of correctly choosing the model structure for a given problem, tuning appropriately the parameters of the systems, and also being able to browse, visualize, and report the results for the understanding of the algorithms' results.

(2) In accordance with the above, we have stressed the goodness of the MapReduce model as a new methodology that can help current researchers and data scientists for the development of fast and scalable applications in DM and BI. Two main advantages of this programming model are: (1) the transparency for the programmer, who just needs to focus on the development of the Map and Reduce functions, and not in the inner operation of the system (better separation between the storage and execution engines), and (2) the robust fault tolerance that allows a higher scalability and reliability for long running jobs. In addition to the former, we have stressed the use of key/value pairs (NoSQL) instead of relational tables in case we are dealing with unstructured or semi-structured data, such as text documents or XML files. This new database model provides higher flexibility for working this type of data.

(3) In order to fully benefit from the MapReduce features, a certain degree of expertise is needed. Specifically, the programmer must carefully tune several factors such as providing a good storage system (e.g., HDFS), exploiting indexes, and the use of an efficient grouping and scheduling algorithm.[161,162] This supposes a constraint for the achievement of the highest performance with the MapReduce approach.

(4) Relational (and parallel) DBMSs leverage its features to provide the user with a simple environment for storing, accessing, and obtaining information from data. However, they require a slow 'import phase' prior to perform the data exploration, and several users may not find conformable with a declarative/relational framework.

Nevertheless, we must not consider MapReduce as an adversary to classical approaches for managing the Big Data problem, but as a complement to them. In fact, it is possible to write almost any parallel-processing task as either a set of database queries or a set of MapReduce jobs. Parallel DBMSs excel at efficient querying large data sets, whereas MapReduce style systems excel at complex analytics and ETL tasks. Neither is good at what the other does well, such that both technologies are complementary rather than rivals.[162]

In fact, we have stressed several query systems that act as a compromise solution, such as Hive, Pig, or Dremel, among others. They are built upon the upper layer of the Cloud Computing stack, developed trying to link the MapReduce framework to DBMS, such that complex analytics can be carried out into the DBMS with embedded queries.

(5) Finally, although the Hadoop stack has become the *de facto* general-purpose, large-scale data processing platform of choice, we must first study the nature of the problem to solve. First of all, it is suitable for data-intensive parallelizable algorithms, but when facing iterative and/or graph-based algorithms, the performance that can be achieved is below its full potential.[135] Additionally, it does not support online transactions, as the way it is implemented does not support the random reading and writing of a few records.

For this reason, several alternative approaches have been developed for extending the MapReduce model to different work scenarios.[130] Among these new approaches, those related with the Spark environment[122,142,151] have gained the greatest popularity; as a example, we may refer to the recent migration of Mahout from Hadoop to Spark. However, no single programming model or framework can excel at every problem; there are always tradeoffs between simplicity, expressivity, fault tolerance, performance, etc. The former issue makes us recall the argument by Lin[133]: '*If all you have is a hammer, throw away everything that's not a nail!*'. Hence, we must remark that although MapReduce has its limitations, as it cannot be applied for every problem we have, the useful current properties of this approach should make us adopt this framework as much as we can for Big Data problems.

CONCLUSIONS

At present, the Big Data problem in DM and BI has been established as one of the hot topics in the areas of business and research. Owing to the novelty of this concept several questions yet exist. Specifically, we shall refer to its exact definition, features, and especially the most properly methodology to solve this problem successfully.

According to the above, the aim when developing this work was to unify the vision of the latest state of the art on the subject. In particular, emphasizing the importance of this new field of work with respect to its application in BI tasks, and exposing the use of Cloud Computing as the right tool when compared to more traditional solutions. This new paradigm of work is based on the use of elastic computing resources. It allows users to execute requests dynamically, such that it can achieve a high degree of scalability even with low-end hardware resources. We have described the architecture of this model, stressing those current implementations that can be found in each of the component layers, i.e., the file system, the execution layer, DBMSs based on NoSQL, and those systems at the query level.

Within the Cloud Computing paradigm, the main point of this work has been focused on the execution layer, and more specifically on the MapReduce programming model. We have described the main features of this approach and its 'pros and cons' when compared versus some reference parallel computational models. In addition, we have enumerated several alternatives to the standard MapReduce model that have been proposed in the literature. These systems aim to extend the application of MapReduce for environments where it shows certain deficiencies, such as the implementation of iterative or graph-based algorithms.

In summary, our main purpose was to provide an opportunity for any interested reader in the field of Big Data, to become familiar with the most relevant information of this promising topic.

NOTES

[a] http://stevetodd.typepad.com/my_weblog/big-data/
[b] http://cruncher.ncl.ac.uk/bdcomp/index.pl
[c] http://www.synapse.org/dream
[d] In Hadoop version 2.0, the functionality of JobTracker has been further subdivided into several daemons. Notwithstanding, the overall working is equivalent.
[e] https://www.cloudera.com

ACKNOWLEDGMENTS

This article is funded by King Abdulaziz University, under grant no. 3-611-1434-HiCi. The authors, therefore, acknowledge technical and financial support of KAU. This work was also partially supported by the Spanish Ministry of Science and Technology under projects TIN2011-28488, TIN-2012-33856, and the Andalusian Research Plans P12-TIC-2958, P11-TIC-7765, and P10-TIC-6858. V. López holds a FPU scholarship from Spanish Ministry of Education.

REFERENCES

1. Gupta R, Gupta H, Mohania M. Cloud computing and big data analytics: what is new from databases perspective? In: *1st International Conference on Big Data Analytics* (*BDA*), New Delhi, India, 2012, 42–61.
2. Agrawal D, Das S, Abbadi AE. Big data and cloud computing: current state and future opportunities. In: *14th International Conference on Extending Database Technology* (*EDBT*), Uppsala, Sweden, 2011, 530–533.
3. Madden S. From databases to big data. *IEEE Internet Comput* 2012, 16:4–6.
4. Kraska T. Finding the needle in the big data systems haystack. *IEEE Internet Comput* 2013, 17:84–86.
5. Alpaydin E. *Introduction to Machine Learning*. 2nd ed. Cambridge, MA: MIT Press; 2010.
6. Witten IH, Frank E, Hall MA. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Amsterdam: Morgan Kaufmann; 2011.
7. March ST, Hevner AR. Integrated decision support systems: a data warehousing perspective. *Decis Support Syst* 2007, 43:1031–1043.

8. Watson HJ, Wixom BH. The current state of business intelligence. *Computer* 2007, 40:96–99.
9. Negash S, Gray P. Business intelligence. *Commun Assoc Inf Sys* 2004, 13:177–195.
10. O'Neil C, Schutt R. *Doing Data Science*. 1st ed. Sebastopol, CA: O'Reilly; 2013.
11. Provost F, Fawcett T. *Data Science for Business. What You Need to Know about Data Mining and Data-Analytic Thinking*. 1st ed. Sebastopol, CA: O'Reilly; 2013.
12. Han J, Haihong E, Le G, Du J. Survey on nosql database. In: *6th International Conference on Pervasive Computing and Applications*, (*ICPCA*), Port Elizabeth, South Africa, 2011, 363–366.
13. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Commun ACM* 2008, 51:107–113.
14. Lam C. *Hadoop in Action*. 1st ed. Greenwich, Connecticut (USA): Manning; 2011.
15. Buyya R, Broberg J, Goscinski A. *Cloud Computing: Principles and Paradigms*. Chichester: John Wiley & Sons; 2011.
16. Mell P, Grance T. The NIST definition of cloud computing (draft) recommendations of the national institute of standards and technology. *NIST Spec Publ* 2011, 145.
17. Plummer D, Bittman T, Austin T, Cearley D, Cloud DS. Defining and describing an emerging phenomenon. Technical Report, Gartner, 2008.
18. Hummer W, Satzger B, Dustdar S. Elastic stream processing in the cloud. *WIREs Data Mining Knowl Discov* 2013, 3:333–345.
19. Papazoglou M, Van Den Heuvel W-J. Service oriented architectures: approaches, technologies and research issues. *VLDB J* 2007, 16:389–415.
20. Ouf S, Nasr M, Amr M, Mosaad M, Kamal K, Mostafa F, Said R, Mohamed S. Business intelligence software as a service (SAAS). In: *2011 IEEE 3rd International Conference on Communication Software and Networks* (*ICCSN*), Xian, China, 2011, 641–649.
21. Velte AT, Velte TJ, Elsenpeter R. *Cloud Computing: A Practical Approach*. New York City (USA): McGraw Hill; 2010.
22. Marx V. The big challenges of big data. *Nature* 2013, 498:255–260.
23. Wu X, Zhu X, Wu G-Q, Ding W. Data mining with big data. *IEEE Trans Knowl Data Eng* 2014, 26:97–107.
24. Schlieski T, Johnson BD. Entertainment in the age of big data. *Proc IEEE* 2012, 100:1404–1408.
25. Peltier TR, Peltier J, Blackley JA. *Information Security Fundamentals*. Boca Raton, FL: Auerbach Publications; 2005.
26. Xiao Z, Xiao Y. Achieving accountable mapreduce in cloud computing. *Future Gener Comput Syst* 2014, 30:1–13.
27. Zikopoulos PC, Eaton C, deRoos D, Deutsch T, Lapis G. *Understanding Big Data—Analytics for Enterprise Class Hadoop and Streaming Data*. 1st ed. New York City (USA): McGraw-Hill Osborne Media; 2011.
28. Labrinidis A, Jagadish HV. Challenges and opportunities with big data. *PVLDB* 2012, 5:2032–2033.
29. Peralta D, Triguero I, Sanchez-Reillo R, Herrera F, Benítez JM. Fast fingerprint identification for large databases. *Pattern Recognit* 2014, 47:588–602.
30. Peralta D, Galar M, Triguero I, Benítez JM, Miguel-Hurtado O, Herrera F. Minutiae filtering to improve both efficacy and efficiency of fingerprint matching algorithms. *Eng Appl Artif Intel* 2014, 32:37–53.
31. Waller MA, Fawcett SE. Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management. *J Bus Logistics* 2013, 34:77–84.
32. Hey T, Trefethen AE. The UK E-science core programme and the grid. *Future Gener Comput Syst* 2002, 18:1017–1031.
33. Wang F-Y, Carley KM, Zeng D, Mao W. Social computing: from social informatics to social intelligence. *IEEE Intell Syst* 2007, 22:79–83.
34. Linden G, Smith B, York J. Amazon.com recommendations item-to-item collaborative filtering. *IEEE Internet Comput* 2003, 7:76–80.
35. Ngai EWT, Wat FKT. Literature review and classification of electronic commerce research. *Inf Manage* 2002, 39:415–429.
36. Mattmann CA. Computing: a vision for data science. *Nature* 2013, 493:473–475.
37. Provost F, Fawcett T. Data science and its relationship to big data and data-driven decision making. *Big Data* 2013, 1:51–59.
38. Cherkassky V, Mulier F. *Learning from Data: Concepts, Theory, and Methods*. Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications and Control Series. New York, NY: John Wiley & Sons; 1998.
39. Tan P-N, Steinbach M, Kumar V. *Introduction to Data Mining*. London (U.K.): Pearson; 2006.
40. Chen CP, Zhang C-Y. Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inf Sci* 2014, 275:314–347.
41. Assent I. Clustering high dimensional data. *WIREs Data Mining Knowl Discov* 2012, 2:340–350.
42. Bacardit J, Llorà X. Large-scale data mining using genetics-based machine learning. *WIREs Data Mining Knowl Discov* 2013, 3:37–61.
43. Rothnie JBJ, Bernstein PA, Fox S, Goodman N, Hammer M, Landers TA, Reeve CL, Shipman DW, Wong E. Introduction to a system for distributed databases (sdd-1). *ACM Trans Database Syst* 1980, 5:1–17.

44. DeWitt DJ, Ghandeharizadeh S, Schneider DA, Bricker A, Hsiao H-I, Rasmussen R. The gamma database machine project. *IEEE Trans Knowl Data Eng* 1990, 2:44–62.
45. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. BigTable: a distributed storage system for structured data. *ACM Trans Comput Syst* 2008, 26:1–26.
46. Shafer J, Agrawal R, Mehta M. Sprint: a scalable parallel classifier for data mining. In: *22th International Conference on Very Large Data Bases* (*VLDB '96*), Mumbai, 1996, *544–555*.
47. Snir M, Otto S. *MPI-The Complete Reference: The MPI Core*. Cambridge, MA: MIT Press; 1998.
48. The Apache Software Foundation. Hadoop, an open source implementing of mapreduce and GFS, 2012
49. Creeger M. Cloud computing: an overview. *ACM Queue* 2009, 7:2.
50. Vaquero LM, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds: towards a cloud definition. *SIGCOMM Comput Commun Rev* 2008, 39:50–55.
51. Shim K, Cha SK, Chen L, Han W-S, Srivastava D, Tanaka K, Yu H, Zhou X. Data management challenges and opportunities in cloud computing. In: *17th International Conference on Database Systems for Advanced Applications (DASFAA'2012)*. Berlin/Heidelberg: Springer; 2012, 323.
52. Kambatla K, Kollias G, Kumar V, Grama A. Trends in big data analytics. *J Parallel Distrib* 2014, 74:2561–2573.
53. White T. *Hadoop: The Definitive Guide*. 1st ed. Sebastopol, CA: O'Reilly; 2009.
54. Ghemawat S, Gobioff H, Leung S-T. The google file system. In: *19th Symposium on Operating Systems Principles*, 2003, Bolton Landing, NY, 29–43.
55. Murty J. *Programming Amazon Web Services—S3, EC2, SQS, FPS, and SimpleDB: Outsource Your Infrastructure*. Ist ed. Sebastopol, CA: O'Reilly; 2008.
56. Chaiken R, Jenkins B, Larson P, Ramsey B, Shakib D, Weaver S, Zhou J. SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB* 2008, 1:1265–1276.
57. Grossman RL, Gu Y. Data mining using high performance data clouds: experimental studies using sector and sphere. In: *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD*), 2008, Las Vegas, NV, 920–927.
58. Grossman RL, Gu Y, Sabala M, Zhang W. Compute and storage clouds using wide area high performance networks. *Future Gener Comput Syst* 2009, 25: 179–183.
59. Härder T, Reuter A. Principles of transaction-oriented database recovery. *ACM Comput Surv* 1983, 15: 287–317.
60. Kim Y, Shim K. Parallel top-k similarity join algorithms using mapreduce. In: *28th International Conference on Data Engineering* (*ICDE*), Arlington, VA, 2012, 510–521.
61. Okcan A, Riedewald M. Processing theta-joins using mapreduce. In: Sellis TK, Miller RJ, Kementsietsidis A, Velegrakis Y, eds. *SIGMOD Conference*. New York, NY: ACM; 2011, 949–960.
62. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper Syst Rev* 2007, 41:205–220.
63. Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D.. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: *29th Annual ACM Symposium on Theory of Computing* (*STOC*), New York, NY, 1997, 654–663.
64. Dimiduk N, Khurana A. *HBase in Action*. 1st ed. Greenwich, Connecticut (USA): Manning; 2012.
65. Lakshman A, Malik P. Cassandra: a decentralized structured storage system. *Oper Syst Rev* 2010, 44:35–40.
66. Russell J, Cohn R. *Hypertable*. 1st ed. Key Biscayne, FL, (USA): Bookvika publishing; 2012.
67. Chodorow K, Dirolf M. *MongoDB: The Definitive Guide Powerful and Scalable Data Storage*. 1st ed. Sebastopol, CA: O'Reilly; 2010.
68. Severance C. Discovering javascript object notation. *IEEE Comput* 2012, 45:6–8.
69. Anderson JC, Lehnardt J, Slater N. CouchDB: The Definitive Guide Time to Relax. 1st ed. Sebastopol, CA: O'Reilly; 2010.
70. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive—a warehousing solution over a map-reduce framework. *VLDB J* 2009, 2:1626–1629.
71. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig latin: a not-so-foreign language for data processing. In: *2008 ACM SIGMOD International Conference on Management of data* (*SIGMOD*), Vancouver, Canada, 2008, 1099–1110.
72. Gates A, Natkovich O, Chopra S, Kamath P, Narayanam S, Olston C, Reed B, Srinivasan S, Srivastava U. Building a highlevel dataflow system on top of mapreduce: the pig experience. *PVLDB* 2009, 2: 1414–1425.
73. Beyer KS, Ercegovac V, Gemulla R, Balmin A, Eltabakh MY, Kanne C-C, Özcan F, Shekita EJ. Jaql: a scripting language for large scale semistructured data analysis. *VLDB J* 2011, 4:1272–1283.
74. Melnik S, Gubarev A, Long J, Romer G, Shivakumar S, Tolton M, Vassilakis T. Dremel: interactive analysis of web-scale datasets. *PVLDB* 2010, 3:330–339.
75. Hausenblas M, Nadeau J. Apache drill: interactive ad-hoc analysis at scale. *Big Data* 2013, 1:100–104.

76. Zhou J, Bruno N, Wu M-C, Larson P-K, Chaiken R, Shakib D. SCOPE: parallel databases meet mapreduce. *VLDB J* 2012, 21:611–636.

77. Lin J, Dyer C. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. California (USA): Morgan and Claypool Publishers; 2010.

78. Shim K. Mapreduce algorithms for big data analysis. *PVLDB* 2012, 5:2016–2017.

79. Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E. A survey on sensor networks. *IEEE Commun Mag* 2002, 40:102–105.

80. Patcha A, Park J-M. An overview of anomaly detection techniques: existing solutions and latest technological trends. *Comput Netw* 2007, 51:3448–3470.

81. Elo L, Schwikowski B. Mining proteomic data for biomedical research. *WIREs Data Mining Knowl Discov* 2012, 2:1–13.

82. Glaab E, Bacardit J, Garibaldi JM, Krasnogor N. Using rule-based machine learning for candidate disease gene prioritization and sample classification of cancer gene expression data. *PLoS One* 2012, 7:e39932.

83. Swan AL, Mobasheri A, Allaway D, Liddell S, Bacardit J. Application of machine learning to proteomics data: classification and biomarker identification in postgenomics biology. *Omics* 2013, 17:595–610.

84. Cappello F, Geist A, Gropp B, Kalé LV, Kramer B, Snir M. Toward exascale resilience. *Int J High Perform Comput Appl* 2009, 23:374–388.

85. Jin H, Chen Y, Zhu H, Sun X-H. Optimizing hpc fault-tolerant environment: an analytical approach. In: *39th International Conference on Parallel Processing* (*ICPP*), San Diego, CA, 2010, 525–534.

86. Jin H, Sun X-H. Performance comparison under failures of mpi and mapreduce: an analytical approach. *Future Gener Comput Syst* 2013, 29:1808–1815.

87. Tu T, Rendleman CA, Borhani DW, Dror RO, Gullingsrud J, Jensen M, Klepeis JL, Maragakis P, Miller PJ, Stafford KA, Shaw DE. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In: *ACM/IEEE Conference on Supercomputing* (*SC*), Austin, TX, 2008, 1–12.

88. Plimpton SJ, Devine KD. Mapreduce in mpi for large-scale graph algorithms. *Parallel Comput* 2011, 37:610–632.

89. Srinivasan A, Faruquie TA, Joshi S. Data and task parallelism in ilp using mapreduce. *Mach Learn* 2012, 86:141–168.

90. Dean J, Ghemawat S. MapReduce: a flexible data processing tool. *Commun ACM* 2010, 53:72–77.

91. Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M.. A comparison of approaches to large-scale data analysis. In: *2009 ACM SIGMOD International Conference on Management of data*, Providence, RI, 2009, 165–178.

92. Duda RO, Stork DG, Hart PE. *Pattern Classification*. New York, NY; Chichester: John Wiley & Sons; 2000.

93. Hastie T, Tibshirani R, Friedman J. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd ed. New York, NY; Berlin/Heidelberg: Springer; 2009.

94. He Q, Du C, Wang Q, Zhuang F, Shi Z. A parallel incremental extreme SVM classifier. *Neurocomputing* 2011, 74:2532–2540.

95. López V, del Río S, Benítez JM, Herrera F. Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data. *Fuzzy Set Syst* in press. doi: 10.1016/j.fss.2014.01.015.

96. He Q, Wang H, Zhuang F, Shang T, Shi Z. Parallel sampling from big data with uncertainty distribution. *Fuzzy Set Syst* in press. doi: 10.1016/j.fss.2014.01.016:1–14.

97. Zhang J, Rui Li T, Ruan D, Gao Z, Zhao C. A parallel method for computing rough set approximations. *Inf Sci* 2012, 194:209–223.

98. Palit I, Reddy CK. Scalable and parallel boosting with mapreduce. *IEEE Trans Knowl Data Eng* 2012, 24:1904–1916.

99. Río S, López V, Benítez J, Herrera F. On the use of mapreduce for imbalanced big data using random forest. *Inf Sci* in press. doi: 10.1016/j.ins.2014.03.043.

100. Bahga A, Madisetti VK. Analyzing massive machine maintenance data in a computing cloud. *IEEE Trans Parallel Distrib Syst* 2012, 23:1831–1843.

101. Magoulès F, Zhao H-X, Elizondo D. Development of an rdp neural network for building energy consumption fault detection and diagnosis. *Energy Build* 2013, 62:133–138.

102. Wang P, Wang J, Chen Y, Ni G. Rapid processing of remote sensing images based on cloud computing. *Future Gener Comput Syst* 2013, 29:1963–1968.

103. Schatz MC. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics* 2009, 25: 1363–1369.

104. Schönherr S, Forer L, Weißensteiner H, Kronenberg F, Specht G, Kloss-Brandstätter A. Cloudgene: a graphical execution platform for mapreduce programs on private and public clouds. *BMC Bioinformatics* 2012, 13:200.

105. Taylor RC. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC Bioinformatics* 2010, 11:S1.

106. Wang Y, Goh W, Wong L, Montana G. Random forests on hadoop for genome-wide association studies of multivariate neuroimaging phenotypes. *BMC Bioinformatics* 2013, 14:S6.

107. Han J, Cheng H, Xin D, Yan X. Frequent pattern mining: current status and future directions. *Data Mining Knowl Discov* 2007, 15:55–86.

108. Zhang C, Zhang S. *Association Rule Mining: Models and Algorithms*. Berlin/Heidelberg: Springer; 2002.
109. Woo J. Market basket analysis algorithms with mapreduce. *WIREs Data Min Knowl Discov* 2013, 3:445–452.
110. Qiao S, Li T, Peng J, Qiu J. Parallel sequential pattern mining of massive trajectory data. *Int J Comput Intell Syst* 2010, 3:343–356.
111. Jain AK. Data clustering: 50 years beyond k-means. *Pattern Recognit Lett* 2010, 31:651–666.
112. Xu R, Wunsch D. *Clustering*. 1st ed. Hoboken, New Jersey (USA): Wiley-IEEE Press; 2009.
113. Aljarah I, Ludwig SA. Parallel particle swarm optimization clustering algorithm based on mapreduce methodology. In: *4th World Congress Nature and Biologically Inspired Computing (NaBIC)*, Mexico City, Mexico, 2012, 104–111.
114. Kim Y, Shim K, Kim M-S, Sup Lee J. Dbcure-mr: an efficient density-based clustering algorithm for large data using mapreduce. *Inf Syst* 2014, 42:15–35.
115. Yang Y, Rutayisire T, Lin C, Li T, Teng F. An improved cop-kmeans clustering for solving constraint violation based on mapreduce framework. *Fundam Inf* 2013, 126:301–318.
116. Ricci F, Rokach L, Shapira B, Kantor PB, eds. *Recommender Systems Handbook*. Berlin/Heidelberg: Springer; 2011.
117. Schafer JB, Frankowski D, Herlocker J, Sen S. Collaborative filtering recommender systems. In: Brusilovsky P, Kobsa A, Nejdl W, eds. *The Adaptive Web*. Berlin/Heidelberg: Springer-Verlag; 2007, 291–324.
118. Kim Y, Shim K. Twilite: a recommendation system for twitter using a probabilistic model based on latent dirichlet allocation. *Inf Syst* 2013, 42:59–77.
119. Lai C-F, Chang J-H, Hu C-C, Huang Y-M, Chao H-C. Cprs: a cloud-based program recommendation system for digital TV platforms. *Future Gener Comput Syst* 2011, 27:823–835.
120. Zigkolis C, Karagiannidis S, Koumarelas I, Vakali A. Integrating similarity and dissimilarity notions in recommenders. *Expert Syst Appl* 2013, 40:5132–5147.
121. Owen S, Anil R, Dunning T, Friedman E. *Mahout in Action*. 1st ed. Manning; 2011.
122. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: *HotCloud*, Boston, MA 2010, 1–7.
123. Ericson K, Pallickara S. On the performance of high dimensional data clustering and classification algorithms. *Future Gener Comput Syst* 2013, 29:1024–1034.
124. Ghoting A, Kambadur P, Pednault EPD, Kannan R. Nimble: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011, San Diego, CA, 334–342.
125. Ghoting A, Krishnamurthy R, Pednault E, Reinwald B, Sindhwani V, Tatikonda S, Tian Y, Vaithyanathan S. Systemml: declarative machine learning on mapreduce. In: *27th International Conference on Data Engineering (ICDE)*, 2011, Washington, DC, 231–242.
126. Das S, Sismanis Y, Beyer KS, Gemulla R, Haas PJ, McPherson J. Ricardo: integrating R and Hadoop. In: *2010 International Conference on Management of data (SIGMOD)*, Indianapolis, IN, 2010, 987–998.
127. Guha S, Hafen R, Rounds J, Xia J, Li J, Xi B, Cleveland WS. Large complex data: divide and recombine (D&R) with RHIPE. *Stat* 2012, 1:53–67.
128. Guha S, Hafen RP, Kidwell P, Cleveland W. Visualization databases for the analysis of large complex datasets. *J Mach Learn Res* 2009, 5:193–200.
129. Wegener D, Mock M, Adranale D, Wrobel S. Toolkit-based high-performance data mining of large data on mapreduce clusters. In: *IEEE International Conference on Data Mining Workshops*, Miami, FL, 2009, 296–301.
130. Agneeswaran V. *Big Data Analytics Beyond Hadoop: Real-Time Applications with Storm, Spark, and More Hadoop Alternatives*. 1st ed. London (U.K.): Pearson; 2014.
131. Wise L. *Using Open Source Platforms for Business Intelligence*. Amsterdam: Morgan Kaufmann; 2012.
132. Amdahl GM. Validity of the single-processor approach to achieving large scale computing capabilities. In: *Sprint Joint Computer Conference of the American Federation of Information Processing Societies*, Atlantic City, NJ, 1967, 483–485.
133. Lin J. Mapreduce is good enough? *Big Data* 2013, 1:BD28–BD37.
134. Kajdanowicz T, Kazienko P, Indyk W. Parallel processing of large graphs. *Future Gener Comput Syst* 2014, 32:324–337.
135. Srirama SN, Jakovits P, Vainikko E. Adapting scientific computing problems to clouds using mapreduce. *Future Gener Comput Syst* 2012, 28:184–192.
136. Chen Y, Ferguson A, Martin B, Wang A, Wendell P. Big data and internships at cloudera. *ACM Crossroads* 2012, 19:35–37.
137. Zhang J, Wong J-S, Li T, Pan Y. A comparison of parallel large-scale knowledge acquisition using rough set theory on different mapreduce runtime systems. *Int J Approx Reason* 2014, 55:896–907.
138. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D.. Dryad: distributed data-parallel programs from sequential building blocks. In: *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys)*, New York, NY, 2007, 59–72.
139. Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson L, Gunda PK, Currey J. Dryadlinq: a system for

general-purpose distributed data-parallel computing using a high-level language. In: *Operating Systems Design and Implementation*, San Diego, CA, 2008, 1–14.

140. Bu Y, Howe B, Balazinska M, Ernst MD. The haloop approach to large-scale iterative data analysis. *PVLDB* 2012, 21:169–190.

141. Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, Fox G. Twister: a runtime for iterative mapreduce. In: *ACM International Symposium on High Performance Distributed Computing* (*HPDC*), 2010, New York, NY, 810–818.

142. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *9th USENIX Conference on Networked Systems Design and Implementation*, San Jose, CA, 2012, 1–14.

143. Kraska T, Talwalkar A, Duchi J, Griffith R, Franklin M, Jordan M. Mlbase: a distributed machine learning system. In: *Conference on Innovative Data Systems Research*, Asilomar, CA, 2013, 1–7.

144. Valiant LG. A bridging model for parallel computation. *Commun ACM* 1990, 33:103–111.

145. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G.. Pregel: a system for large-scale graph processing. In: *2010 ACM SIGMOD International Conference on Management of data*, Indianapolis, IN, 2010, 135–146.

146. Martella C, Shaposhnik R, Logothetis D. *Giraph in Action*. 1st ed. Manning; 2014.

147. Xin RS, Gonzalez JE, Franklin MJ, Stoica I. Graphx: a resilient distributed graph system on spark. In: *GRADES'13*, New York, NY, 2013, 1–6.

148. Low Y, Kyrola A, Bickson D, Gonzalez J, Guestrin C, Hellerstein JM. Distributed graphlab: a framework for machine learning in the cloud. *PVLDB* 2012, 5:716–727.

149. Anderson Q. *Storm Real-Time Processing Cookbook*. 1st ed. Sebastopol, CA: O'Reilly; 2013.

150. Neumeyer L, Robbins B, Nair A, Kesari A. S4: distributed stream computing platform. In: *2010 IEEE Data Mining Workshops* (*ICDMW*), Sydney, Australia, 2010, 170–177.

151. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized streams: fault-tolerant streaming computation at scale. In: *24th ACM Symposium on Operating Systems Principles* (*SOSP*), Farmington, PA, 2013, 423–438.

152. Ranger C, Raghuraman R, Penmetsa A, Bradski G, Kozyrakis C. Evaluating mapreduce for multi-core and multiprocessor systems. In: *IEEE 13th International Symposium on High Performance Computer Architecture* (*HPCA*), Phoenix, AZ, 2007, 1–6.

153. Fang W, He B, Luo Q, Govindaraju N. Mars: accelerating mapreduce with graphics processors. *IEEE Trans Parallel Distrib Syst* 2011, 22:608–620.

154. Chu C-T, Kim SK, Lin Y-A, Yu Y, Bradski GR, Ng AY, Olukotun K.. Map-reduce for machine learning on multicore. In: *Advances in Neural Information Processing Systems 19*, *Twentieth Annual Conference on Neural Information Processing Systems* (*NIPS*), Vancouver, Canada, 2006, 281–288.

155. Ailamaki A, Govindaraju NK, Harizopoulos S, Manocha D. Query co-processing on commodity processors. In: *32nd International Conference on Very Large Data Bases* (*VLDB*), Seoul, Korea, 2006, 1267–1267.

156. Stuart JA, Owens JD. Multi-gpu mapreduce on gpu clusters. In: *25th IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), Anchorage, AK, 2011, 1068–1079.

157. Basaran C, Kang K-D. Grex: an efficient mapreduce framework for graphics processing units. *J Parallel Distrib Comput* 2013, 73:522–533.

158. Pan Y, Zhang J. Parallel programming on cloud computing platforms—challenges and solutions. *J Convergence* 2012, 3:23–28.

159. Schmidt K, Phillips C. *Programming Elastic MapReduce*. Ist ed. Sebastopol, CA: O'Reilly; 2013.

160. Nadipalli R. *HDInsight Essentials*. Ist ed. Birmingham: PACKT Publishing; 2013.

161. Jiang D, Ooi B, Shi L, Wu S. The performance of mapreduce: an in-depth study. *PVLDB* 2010, 3:472–483.

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