J Grid Computing (2017) 15:295–321
DOI 10.1007/s10723-017-9408-0

MapReduce and Its Applications, Challenges, and Architecture: a Comprehensive Review and Directions for Future Research

Seyed Nima Khezr · Nima Jafari Navimipour

Received: 28 October 2016 / Accepted: 8 August 2017 / Published online: 26 August 2017

Abstract Profound attention to MapReduce framework has been caught by many different areas. It is presently a practical model for data-intensive applications due to its simple interface of programming, high scalability, and ability to withstand the subjection to flaws. Also, it is capable of processing a high proportion of data in distributed computing environments (DCE). MapReduce, on numerous occasions, has proved to be applicable to a wide range of domains. However, despite the significance of the techniques, applications, and mechanisms of MapReduce, there is no comprehensive study at hand in this regard. Thus, this paper not only analyzes the MapReduce applications and implementations in general, but it also provides a discussion of the differences between varied implementations of MapReduce as well as some guidelines for planning future research.

Keywords MapReduce · Big Data · Cloud · Hadoop

1 Introduction

Nowadays, with the excessive growth in the information and data, their analysis has become a burdensome challenge. MapReduce is a fault-tolerant, simple, and scalable framework for data processing that enables its users to process these massive amounts of data [1]. It is a framework for efficient large-scale data processing which is presented by Google in 2004 in order to tackle the issue of processing large amounts of data with reference to the Internet-based applications [2]. These large input data need to be indexed, stored, retrieved, analyzed and also mined to allow a simple and continues access to these data and information [3]. MapReduce is one of the forerunners in the so-called "NoSQL" trend to steer it away from mainstream relational databases [4]. Nowadays, there are four factors, including processing, storing, visualization, and analyzing large data in modern organizations and enterprises. The MapReduce can automatically run the applications on a parallel cluster of hardware and in addition, it can process terabytes and petabytes of data more rapidly and efficiently. Therefore, its popularity has grown swiftly for diverse brands of enterprises in many fields. It provides a highly effective and efficient framework for the parallel execution of the applications, data allocation in distributed database systems, and fault-tolerance network communications [5, 6]. The main objective of MapReduce is to facilitate data parallelization, data distribution and load balancing in a simple library [7]. The easy availability and accessibility of the MapReduce platforms, such as Hadoop, makes it sufficient for a productive parallelization and execution of data-intensive tasks [8]. The used

S. N. Khezr · N. J. Navimipour (✉)
Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran
e-mail: jafari@iaut.ac.ir

algorithms in the MapReduce have an allowance to manage the data-intensive applications, parallel executions and fault control mechanisms [9]. Programmers who use the library of MapReduce must consider two functions, i.e. a Map and a Reduce function [10, 11]. The Map function receives a key/value pair as input and creates the intermediate key/value pairs for extra processing. The Reduce function merges all the intermediate key/value pairs and then creates the last output [12].

Google's MapReduce and its open-source implementation, Hadoop, have become highly popular in recent years. The software environment of Apache Hadoop delivers a distributed implementation of the data storage and MapReduce computing [13]. Lately, Apache Hadoop has drawn strong attention due to its applicability for big data processing [14] and it provides execution of the tasks over a cluster of many machines which are based on commodity hardware [15]. The programs of MapReduce in Hadoop are usually written in Java, even though it also supports the use of stand-alone Map and Reduces kernels, which can be written as shell scripts or in other languages [16]. Its popularity has increased by the use of some significant factors such as simplicity, automatic parallelizability, accepted scalability, and commodity hardware implementation capability [17]. Also, large-scale data-intensive cloud computing [18, 19] with the MapReduce framework is becoming pervasive for many academic establishments, enterprises, governments, and industrial organizations [20]. Cloud computing as a progressive Internet-based technology [21–23] provides a highly available, scalable, and flexible computing platform for many kinds of applications [24–27]. However, the MapReduce programming model used in the cloud computing has many limitations. Hence, developers have to excel in programming with the MapReduce model and spend adequate time to understand the variegated features of different cloud platforms. With automatic parallelization software, the usage of cloud computing will be limitless for many applications [28]. There are also many applications which use MapReduce for parallel and efficient execution, including data mining [29], web ranking [30], inverted indexing [31], k-means [32], self-join [33], ontology [34], bioinformatics [35] and graph analysis [36].

Despite the importance of MapReduce framework and its prevalence, there have not been any detailed and comprehensive study to analyze its implementation and application. Therefore, this paper aims to review and survey the MapReduce applications, architecture and the environment needed for its implementation as well as specifying the main differences between them. Into the bargain, a taxonomy to differentiate the considered applications and implementations coupled with some suggestions for future research are provided. In outline, the contributions of this paper are as follow:

- Providing an anatomy of various usages of MapReduce framework for big data processing and cloud computing.
- Offering a review and analysis of the MapReduce applications to accentuate the advantages and disadvantages in each domain.
- Overviewing the existing implementations of MapReduce framework in cloud environments in order to underscore the advantages and disadvantages of each framework.
- Outlining the key areas for enhancing the MapReduce framework in the future.

This paper has been structured as follows: Firstly, the related works are analyzed and reviewed in Section 2. Secondly, then big data and the MapReduce architecture are described and investigated respectively in Section 3. Then, Section 4 sheds light on the implementation of MapReduce models. Subsequently, Section 5 analyses the MapReduce applications in different kinds of domains. Section 6 juxtaposes and summarizes the reviewed mechanisms. The open issues are mapped out in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

Some studies on MapReduce in the cloud environment have been carried out as yet. This paper aims to deal with the application and the implantation aspects of MapReduce framework. In fact, a broad variety of MapReduce frameworks has been developed during the last few years. These reviews are more or less described separately in the corresponding literature. This section refers to some review papers which pertain to the MapReduce framework and outlines their main advantageous and disadvantageous features.

Lee et al. [37] has proposed the parallel data processing with MapReduce survey. To investigate various technical aspects of the MapReduce and its pros and cons are the main goals of this paper. Also, it suggests a new classification for MapReduce improvements. However, this paper does not discuss the major limitations of MapReduce, especially in cloud environments.

Reviewing the state-of-the-art about MapReduce parallel programming models is done in [38]. What they pursue in their survey is underlying MapReduce paradigm, describing several widely used open-source runtime systems and discussing the major shortcomings of original MapReduce Moreover, this study describes MapReduce optimization approaches and categorizes them according to their characteristics and capabilities. evertheless, this paper does not include the disadvantages and advantages of different methods.

Khezr and Navimipour [39] have considered MapReduce and its application in the optimization algorithms. The paper presents some MapReduce applications in the optimization algorithms like a genetic, PSO, cuckoo search, and ant colony algorithm. This paper does not examine the exploitation of MapReduce application techniques such as parallel computation, multi-core systems, data allocation, graph processing and, various application areas.

Also, in [40], the MapReduce issues including applications and implementation have been evaluated. The issue has been discussed and some articles have been reviewed. However, the discussed applications and implementations of MapReduce are not reviewed in adequate detail.

Finally, a review of MapReduce limitations, optimizations, and open issues have been presented in [41]. This survey revolves around Hadoop framework and it's identified limitations. But, the paper has merely discussed other implementations of MapReduce and no more.

It is important to point out that the reviewed papers did not present the use cases and benefits of MapReduce application in alternative areas like algorithms, cloud computing, data mining, health care and so forth. This paper has classified the exploring and sharing the MapReduce framework. On the other hand, the existing implementations of MapReduce framework and their pros and cons are not included in the published papers thoroughly. Therefore, in the rest of this paper, these weaknesses have been taken into account.

3 Basic Concepts

This section introduces big data use cases in MapReduce applications and provides a brief overview of MapReduce architecture.

3.1 Big Data

Over the recent years, the volume and complexity of interactions between information systems have been steadily increasing [4]. Big data analytics (organizing, collecting and analyzing huge sets of data) are among today's most frequently discussed topics in research and practice [42]. The world is being changed by data-driven approaches including access to large amounts of data and available opportunities in commerce, science, and computing applications [15, 43]. This category of applications is greatly parallel and well suited for the MapReduce programming that lets users execute large-scale data analyses such that the application execution layer handles the task scheduling, system architecture and data partitioning [44]. As many industries and organizations handle increasing amounts of data, big data processing is being considered as a most important step. Currently, distributed computing frameworks are being widely used for big data processing. These systems allow the user to write applications through a set of high-level operations and automatically handle the complex aspects of distributed computing such as scheduling and fault tolerance [45, 46]. Apache's Hadoop and Google's MapReduce, its open-source implementation, are the de facto software systems for big-data applications. The MapReduce framework aims to create a huge amount of intermediate data. Big data applications take a large quantity of input data in most of the applications [47]. Big data generally refer to a heterogeneous class of business applications that operate on large amounts of data [48]. It also has challenges related three primary features: volume, velocity, and variety. Big Data has 3Vs; Volume (a large amount of data), Velocity (data arrives at high speed) and Variety (heterogeneous resources). In big data definition, big refers to a dataset which makes data concept of growing so much that it becomes difficult to manage it by using existing data management concepts and tools. MapReduce is playing a very significant role in the processing of big data is an undeniable fact [49]. Since MapReduce gains a lot of popularity for its

fault tolerance, flexibility, scalability, and simplicity, it becomes most suitable framework for processing and analysis big data operations [50].

3.2 MapReduce Architecture

Google created MapReduce to process large amounts of unstructured or semi-structured data, such as web documents and logs of web page requests, on large shared-nothing clusters of commodity nodes. It produced various kinds of data such as inverted indices or URL access frequencies [51]. The MapReduce has three major parts, including Master, Map function and Reduce function. The Master is responsible for managing the back-end Map and Reduce functions and offering data and procedures to them [52, 53]. A MapReduce application contains a workflow of jobs where each job makes two user-specified functions: Map and Reduce. The Map function is applied to each input record and produces a list of intermediate records. The Reduce function (also called Reducer) is applied to each group of intermediate records with the same key and produces a list of output records [45]. MapReduce program is expected to be done on several computers and nodes when it is performed on Hadoop [54]. Therefore, a master node runs all the necessary services to organize the communication between Mappers and Reducers. An input file (or files) is separated into the same parts called input splits. They pass to the Mappers in which they work parallel together to provide the data contained within each split. As the data is provided by the Mappers, they separate the output; then each Reducer gathers the data partition by each Mapper, merges them, processes them, and produces the output file. An example of this data flow is shown in Fig. 1.

The main phases of MapReduce architecture are Mapper, Reducer, and shuffle which are presented below:

Mapper: a Mapper processes input data which are assigned by the master to perform some computation on this input and produce intermediate results in the form of key/value pairs [55].
Reducer: The Reduce function receives an intermediate key and a set of values of the key. It combines these values together to form a lesser set of values [56].

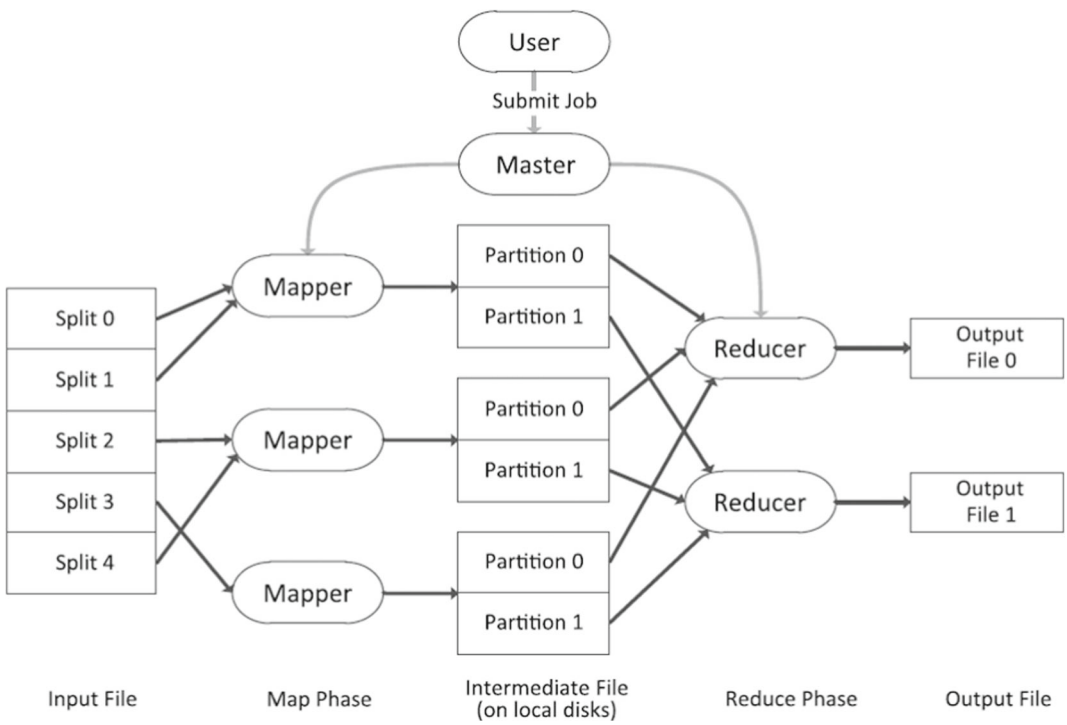

Fig. 1 An example of data flows in the MapReduce architecture [7]

Shuffle: In MapReduce framework, after the Map task is finished, there are usually large amounts of middle data to be moved from all Map nodes to all Reduce nodes in the shuffle phase, the shuffle transfer data from the Mapper disks rather than their main memories and the intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together and it needs to transfer the data from the local Map nodes to Reduce nodes through the network [57].

MapReduce is designed to be fault-tolerant because failures are a common phenomenon in large scale distributed computing [12]. Google's MapReduce architecture seems to be a good choice for several reasons [58]:

- Information processing tasks benefit from parallel and distributed architecture with simply the programming of Map and Reduce methods.
- MapReduce architecture has the ability to process Terabytes of data on PC clusters with handling failures.
- Most of the data recovery and excavating information can be taken into MapReduce architecture, similar to the pattern based annotation algorithms. Distributed Grep is one of the basic examples for MapReduce using Ontea pattern approach with regular expressions as well.

4 MapReduce Implementations

The world's largest companies like Yahoo, Facebook, Amazon, and IBM use the MapReduce model as a tool for the cloud computing applications through implementing Hadoop, an open source code of MapReduce, produced by Apache software foundation [9]. Some implementations with a different approach are developed later, such as the Dryad [59], Phoenix [60], Mars [61], Twister [62] and GridGain [63]. The contributions of this section are to provide an overview of existing implementation of MapReduce applications.

4.1 Google's MapReduce

Google performs the original MapReduce implementation aims to have large clusters of networked machines. The first version of the MapReduce library was written in February 2003, but it gets some significant changes. Its library automatically handles parallelization and data distribution [64]. Google File System (GFS) as a distributed file system makes a duplicate of data blocks on multiple nodes for enhanced reliability, fault tolerance and is intended to view machine failures as a default rather than an irregularity. MapReduce is highly scalable. Therefore, it can be run on clusters comprising of thousands of low-cost machines [65].

4.2 Hadoop

MapReduce has some other implementations, including Mars, Phoenix, Hadoop and Google's implementation. Among them, Hadoop has become the most popular one due to its open source feature [66]. The most general implementation of the MapReduce model is the Hadoop framework, which lets applications to run on large clusters [67]. It is applicable for

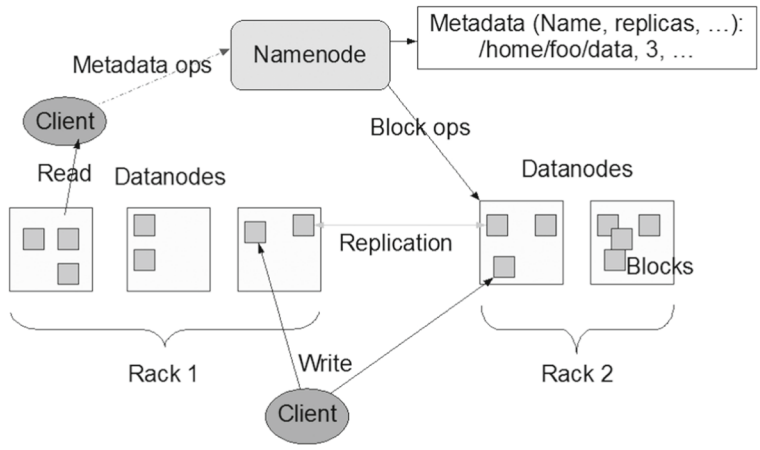

Fig. 2 Hadoop distributed file system (HDFS) [74]

performing the cloud-based large-scale data-parallel applications by providing the reliability and data transfer capabilities [67, 68]. Apache's Hadoop is an open-source implementation of Google's Map/Reduce framework. It enables distributed, data-intensive and parallel applications by analyzing a great job into smaller tasks and a massive data set into smaller partitions in a way that each task processes a different partition in parallel [69]. HDFS, the Hadoop Distributed File System, is a distributed file system which is designed (Fig. 2) to hold an immense number of data (terabytes or even petabytes) and provide high throughput access to the information. HDFS consists of a single NameNode and a master server to manage the file system and provide the access to files by the clients. Furthermore, a file breaks into one or more blocks that they are stored in a set of DataNodes. The DataNodes are responsible for doing 'read' and 'write' requests from the file system's clients. They also perform block creation, replication, and deletion [70]. High-throughput, fault tolerance and elastic scalability are the features of Hadoop and Google frameworks for MapReduce [71]. Amazon, Facebook, Twitter are considered as leading technology business based on a policy of co-locating and processing data to accelerate the performance [72, 73].

Also, Table 1 contains details about various aspects of Hadoop as well as its pros and cons.

4.3 Hadoop+

Hadoop+ is a heterogeneous MapReduce framework, which enables GPUs and CPUs for processing the big data and leveraging the heterogeneity model to assist users in selecting the computing resources for different purposes. Figure 3 shows the summary of the Hadoop+ framework. Hadoop provides the Map and Reduce primitives and PMap and PReduce are provided by Hadoop+ to programmers [75].

The PMap and PReduce in Hadoop+ enable programmers to write explicit parallel CUDA/OpenCL functions running on GPUs as plug-ins, as shown by the box of "User-Provided PMap/PReduce Function" in Fig. 3. Meanwhile, users can also use the Map and Reduce functions in Hadoop. In Hadoop+, users can provide Map, PMap or both, and Reduce, PReduce or both. Hadoop+ provides different input parameters for Map and PMap to support explicit parallel Map functions. In particular, key and value are the input of Map, while the input of PMap is a dataset, i.e., a list of (key, value). Meanwhile, Reduce and PReduce have the same input parameters, which are the outputs with

Table 1 The pros and cons of Hadoop and its components

Pros	Cons
Hadoop is a platform which provides both distributed storage and computational capabilities.	Hadoop uses HDFS and MapReduce, whose master processes are single points of failure. However, active work is in progress for high availability versions.
Hadoop is extremely scalable. In fact, Hadoop was the first to be considered as to fix a scalability issue that existed in Nutch (the open source crawler and search engine).	Security is one of the major concerns. It does offer a security model but it is disabled by default owing to its extreme complexity.
One of the major components of Hadoop is HDFS, which is optimized for high throughput.	Hadoop does not offer storage nor network level encryption, which causes serious concern for data application in government sectors.
HDFS uses large block sizes for processing big data (gigabytes, petabytes...).	HDFS proves to be inefficient for handling small files and has no transparent compression.
HDFS is capable of replicating files for a specified number of times and is tolerant of software and hardware failure as it can automatically re-replicate the data blocks on nodes that have failed.	MapReduce is a shared-nothing architecture (SN); hence, tasks that require global synchronization or sharing of mutable data are not a good fit which can pose challenges for some algorithms.

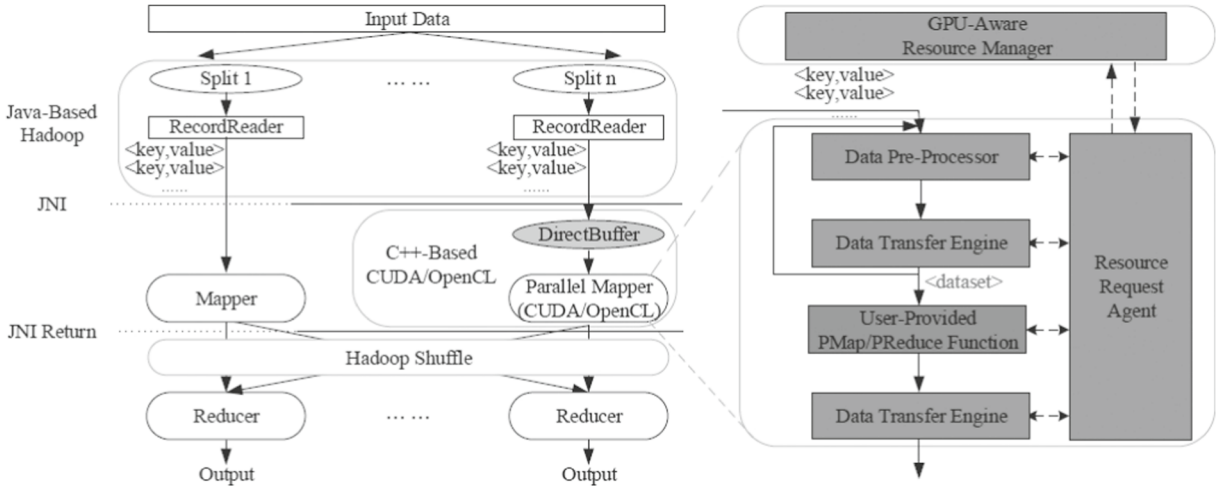

Fig. 3 Overview of Hadoop+ framework [75]

the same key from all Map tasks [75]. Table 2 contains the details about various aspects of the Hadoop+ framework as well as its pros and cons.

4.4 GridGain

The GridGain[1] is another open-source MapReduce implementation. It is an enterprise open-source grid computing made for Java. The GridGain and Hadoop DFS are compatible and it provides a substitute to Hadoop's MapReduce. GridGain offers a distributed, real-time, in-memory and scalable data grid to have a connection between applications and data sources. From the technological point of view, the biggest difference is the primary process of Map tasks assignment to the nodes. GridGain acts as a bridge based on Java in the processing of big data to have high performance and finishes fast in-memory MapReduce implementation. The programmer can process terabytes of data, on 1000s of nodes in less than a second using GridGain. In the MapReduce algorithm, the task is split into subtasks and workers drag the split parts as soon as they have free processor time. In GridGain, the subtasks are pushed to the nodes. This proves to be an advantage since it gives more load balancing capabilities. This benefit depends on situations and the user's needs. As a part of the extra functionality, it introduces some additional complexity that the developer has to plan in advance so that no worker does stay without reason idle. Although GridGain has less popularity, it shows to be better documented and it is interested in beginners [73]. Also, Table 3 contains the details about various aspects of GridGain as well as its pros and cons.

4.5 Mars

Mars is developed for Graphics Processing Units (GPUs) using MapReduce framework. It limits the GPUs complication by means of MapReduce connection and automatically takes task partitioning, data distribution, and parallelization on the processors forward [61]. Figure 4 illustrates the workflow of Mars, assuming the data resides on the disk at the beginning. Its planner operates on the CPU and plans tasks to the graphics processing unit. Mars has three stages, including Map, Group, and Reduce. Before the Map stage, Mars preprocesses the input data from the disk, transforming the input data to key/ value pairs (input records) in the main memory. Following that, it transfers input records from the main memory to the graphics processing unit device memory. In the Map stage, Map Split dispatches input records to GPU threads that the workload for all threads is even. Each thread performs the user-defined MapCount function to calculate a local histogram of the number. Then, the runtime performs a graphics processing unit -based prefix sum on the local histograms to obtain the output size and the write position for each thread. Finally, after the output buffer is allocated to the device memory, each graphics processing unit thread executes the

[1] GridGain, http://www.gridgain.com/

Table 2 The pros and cons of the Hadoop+ framework and its components

Pros	Cons
Hadoop+ exploits the GPU capability and achieves 1.4× to 16.1× speedups over Hadoop for 5 real applications when running individually.	It is not congruous with distributed computing applications.

user-defined Map function and outputs the results. There will be no write conflict between concurrent threads when each thread has computed its write position before and it has no contrast with other threads.

In the Group stage, the sort-based and hash-based strategies are available for grouping records by key. Yet, some applications need sorting all output records and the hash-based strategy has to perform additional sort within each hash bucket. In the Reduce stage, Reduce Split dispatches each group of records with the identical key to a graphics processing unit thread. However, it causes a load imbalance between threads, since the number of records of different groups may vary widely. Furthermore, the MapReduce framework of Mars enables the integration of graphics processing unit-accelerated code to distributed environment, like Hadoop, with the least effort [76]. Table 4 contains the details about various aspects of Mars as well as its pros and cons.

4.6 Phoenix

Phoenix implements MapReduce for shared-memory systems and it aims to support efficient implementation on multiple cores without troubling the programmer with concurrency management. It has a simple API, which has (Application Programming Interface) an efficient runtime to apply parallelization, resource management, and fault recovery [77]. It is used by application programmers to target for multi-core and multiprocessor systems. Parallelism is performed by shared-memory *threads*. First, a user provides the runtime with the Map/Reduce functions for applying on the data. The runtime uses multiple worker threads to execute the computation. In the Map phase (see Fig. 5), the input data are split into *chunks*, and the Map function is invoked on each chunk. This generates intermediate key/value pairs. In Reduce phase, for each unique key, the Reduce function is called with the values for the same key as an argument and reduces them to a single key/value pair. The results of the Reduce tasks are merged and sorted by keys to yield the last output [60].

Table 5 contains details about various aspects of Phoenix as well as its pros and cons.

4.7 Tiled-MapReduce

The chip multiprocessor is very popular and it runs data-parallel applications in clusters on a single machine with many cores. MapReduce as a programming model is used to program large scale clusters. It

Table 3 The advantages and disadvantage of GridGain and its components

Pros	Cons
It is a more generic distributed computation middleware that allows you to easily farm out arbitrary tasks to nodes.	GridGain only provides distributed computation support and does not include a distributed file system like HDFS.
This unique property of GridGain's MapReduce implementation has a profound effect on the ability to develop grid applications with the advanced load balancing, failover, and collision resolution logic.	GridGain's framework does not sort the data between the Mapper and Reducer. However, Hadoop provides an automatically distributed sort of the data between the Maps and Reduces.
GridGain effectively helps to adapt task execution to non-deterministic nature of execution on the grid.	GridGain does not have any support for non-Java applications while Hadoop supports both C++ and text-based applications.

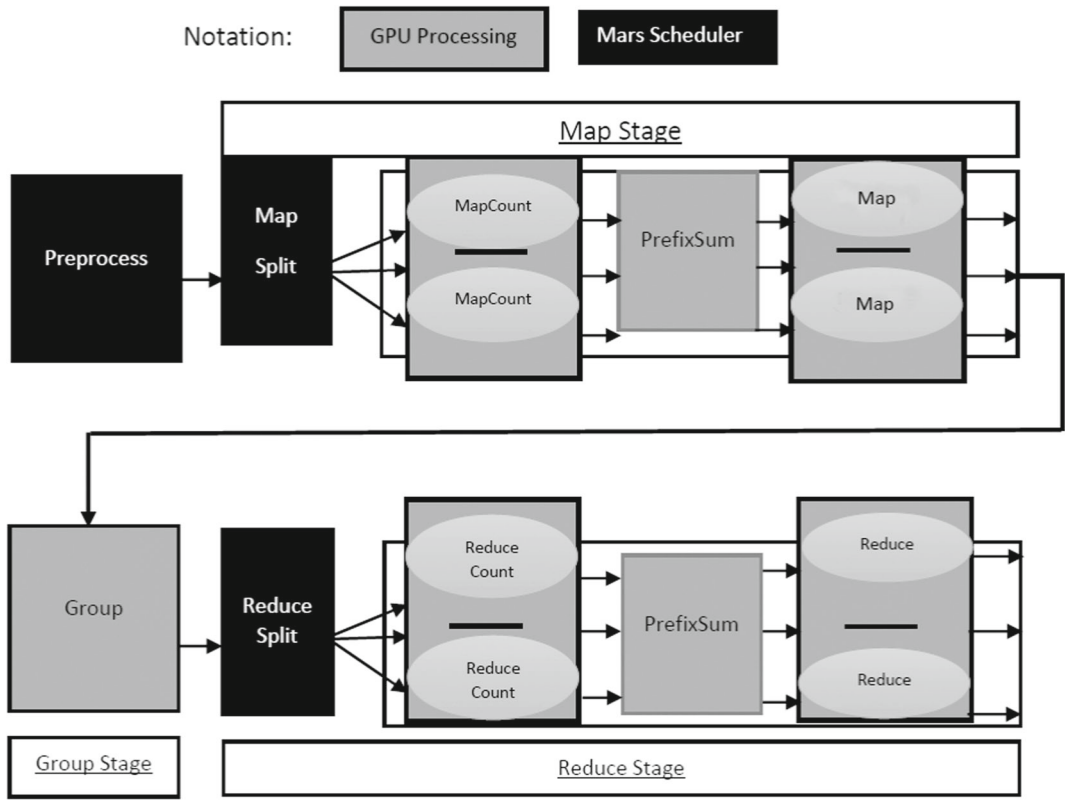

Fig. 4 The workflow of Mars on the graphics processing unit [76]

controls the multicore platform [78], by solving large-scale data-parallel problems. Multi-core CPU systems and cell processors are performed on parallel platforms [79]. The environmental differences between clusters and multicores cause new design spaces and it has chances to improve the performance of MapReduce on multicore. Tiled-MapReduce, which uses the "tiling strategy" to partition a large MapReduce job into a number of small sub-jobs and handles the sub-jobs iteratively. MapReduce as a promising programming model for multicore platforms uses the power of such processing resources [78]. Tiled-MapReduce extends the general Reduce phase to process the partial results of all iterations, rather than the intermediate data. The output of the Reduce phase is compatible with the output of the general Reduce phase. To

Table 4 The pros and cons of Mars and its components

Pros	Cons
It is an atomic-free and the first implementation of MapReduce on GPUs.	It causes expensive preprocessing phases in order to adjust the resulting writing of different threads.
It executes two preprocessing kernels before the Map and Reduces phases, the counting, and prefix summing kernels.	It classifies the clues to sort the results of the Map function, which has inefficient and time-consuming sorting in grouping intermediate results.
It allows programmers and users to take advantage of different processors on a single machine.	One of the major disadvantages of Its preprocessing design is that the Map and Reduce functions need to be executed twice.

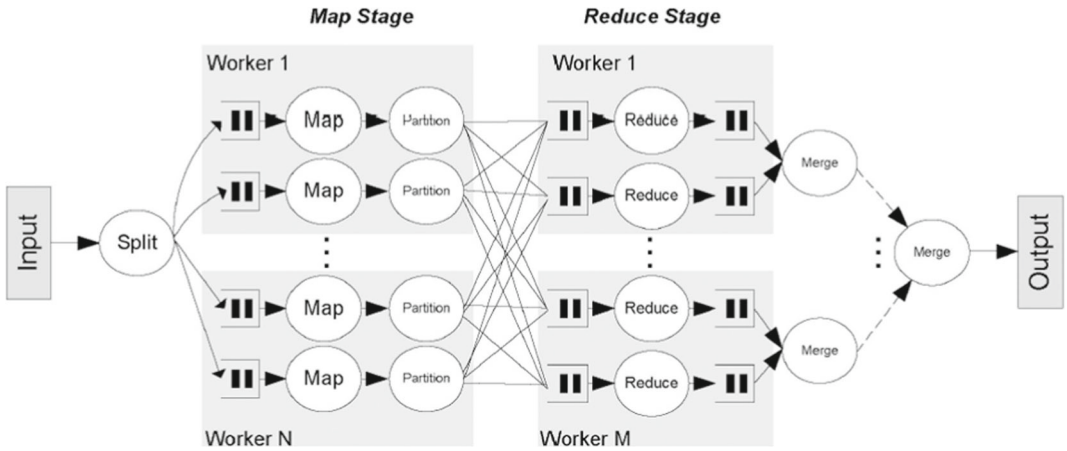

Fig. 5 The basic data flow for the Phoenix runtime [77]

differentiate with that in the final Reduce phase, Tiled-MapReduce, rename the Reduce phase within one sub job as the Combine phase. The lower part of Fig. 6 illustrates the processing phases in Tiled-MapReduce [78].

The overall execution flow of a Tiled-MapReduce job and the implementation of Tiled-MapReduce runtime are illustrated in the top part. The mr_dispatcher function involves a Tiled-MapReduce job, which initializes and configures the dispatcher according to the arguments and runtime environments (e.g., available resources). Then, the dispatcher spawns N workers and binds them to CPU cores. The dispatcher also iteratively splits a chunk from input data in the Iteration Window, whose size is dynamically adjusted according to the runtime configuration. The chunk of data will be further split into M pieces, which forms M map tasks. In the Map phase, a worker selects a map task from the iteration window whenever it is idle and invokes the programmer-provided Map function, which parses the input data and generates intermediate key/value pairs. The emit_intermediate function, which is provided by the runtime, will be invoked to insert a key/value pair to intermediate buffer, which is organized as an M by R matrix of buckets, where R is the number of Reduce tasks. In the Combine phase, the Worker selects the Reduce tasks in turn and invokes the programmer-provided combine function to process a column in the Intermediate Buffer. The structure of the iteration buffer is an/I by R matrix of buckets, where/is the total number of iterations. When all sub-jobs have finished, the Workers invoke the programmer provided Reduce function to do the final Reduce operation on the data in each column of the iteration buffer. The Reduce function inserts the final result of a key to the final buffer by invoking the emit function. Finally, the results of all Reduce tasks are merged and sorted into a single Output Buffer [78].

Table 5 The pros and cons of Phoenix and its components

Pros	Cons
It is a MapReduce runtime targeted for shared-memory multi-cores and multiprocessors. It leads to similar performance for most applications.	It performs well on small-scale systems with uniform access latencies but this fact does not remain true in the large-scale shared-memory system.
Phoenix provides significant speedups with both systems for benchmarks and all processor counts.	The performance of the in-memory data structures used to store and retrieve intermediate data are crucial in overall system performance.
Phoenix automatically handles key scheduling decisions during parallel execution and it can also recover from permanent and transient errors in Mapper and Reducer.	It has some laxity in terms of scalability and NUMA-awareness since it is implemented in C, the 2-D array structure.

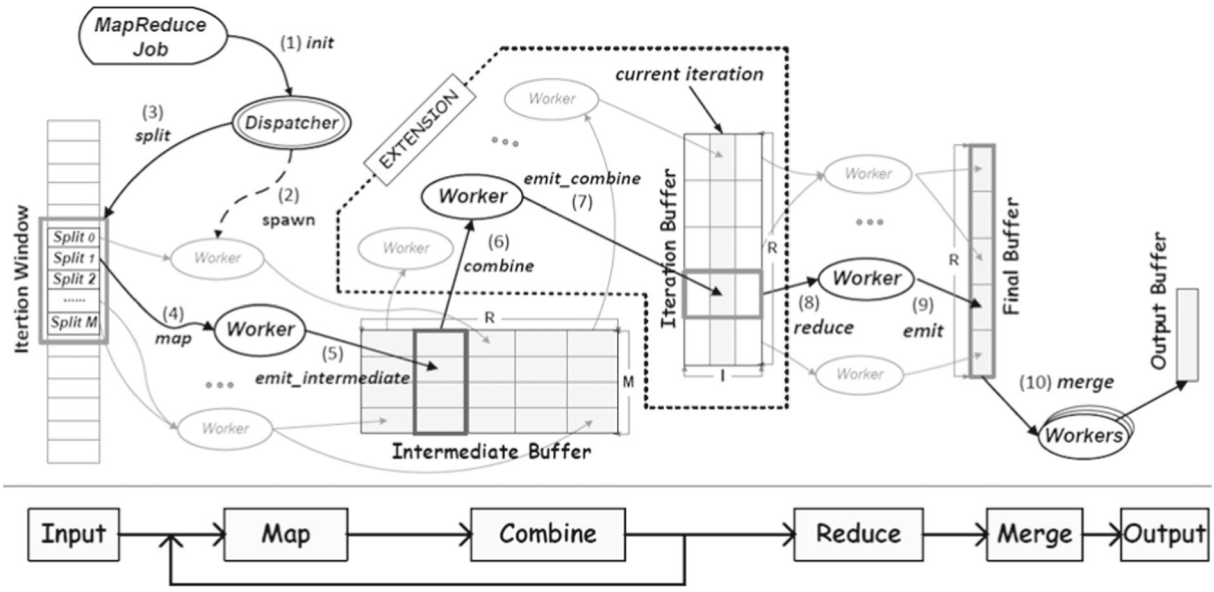

Fig. 6 The workflow of Tiled-MapReduce [78]

Table 6 contains the details about various aspects of Tiled-MapReduce as well as its pros and cons.

4.8 Twister

MapReduce programming model has facilitated the implementations of many data parallel applications. The simplicity of the programming model and the high quality of the services and their simplicity are used by MapReduce requests to parallel computing communities enormously.

Twister identifies a set of improvement in the programming model as well as its architecture according to the different scientific methods based on MapReduce to develop MapReduce runtime. In Fig. 7, the *Map/Reduce* tasks operated with these two types of data products are illustrated which can be used to load (read) any static data at the *Map* and *Reduce* tasks. For example, key and value pairs are defined as variable data in the typical Map phase of the computation and the static data (already loaded) produces a set of output *(key, value)* pairs. It also introduces an optional reduction phase named "combine" to connect the results of the Reduce phase into a single value. The user program and the combined operation are done in a single process space, which allows its output directly accessible to the user program [62]. Table 7 shows the advantages and disadvantages of Twister as well as its details.

4.9 Sector and Sphere

Sector and Sphere are two important factors for high performing of data cloud. The sector can manage data across distributed data centers. Similarly, the Sphere supports user-defined functions (UDFs) over data both within and across data centers. A Map UDF performs the MapReduce- style programming in the sphere

Table 6 The pros and cons of Tiled-MapReduce

Pros	Cons
It provides better data locality and task parallelism.	Tiled-MapReduce investigates the potential use of the tiling strategy in the single-node version of MapReduce.
Tiled-MapReduce explores several optimizations, it improves the memory and saves up to 85% memory.	–

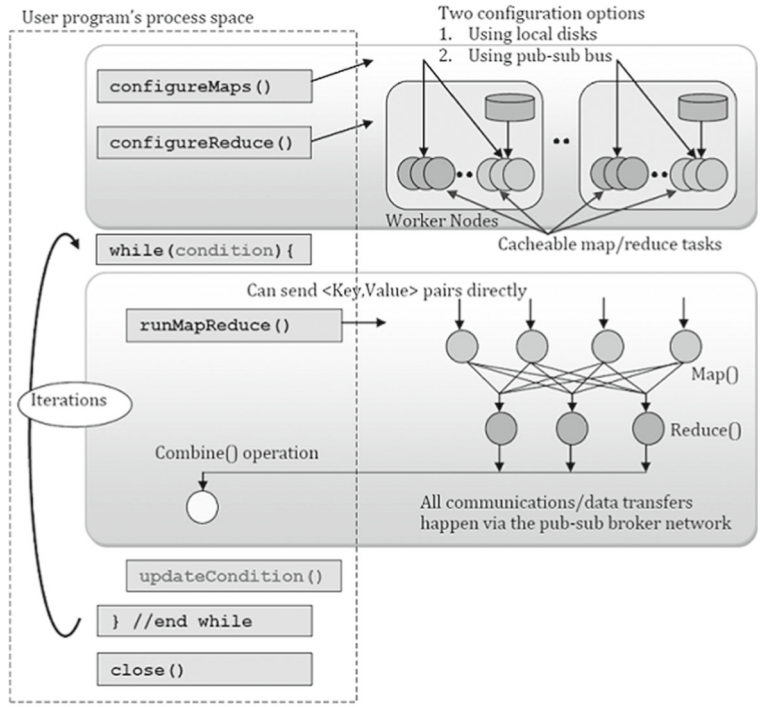

Fig. 7 The workflow of Twister programming model (Twister, http://www.iterativemapreduce.org/)

[80]. Sphere as a MapReduce runtime system provides distributed data storage, distribution, and processing over large clusters of commodity computers across a single or multiple data centers. This part has three features, including secure, high performance and scalable distributed file system. The sphere provides Sector data files on the storage nodes by simple programming, which runs faster than Hadoop. On the other side, Twister as a programming model strongly supports iterative MapReduce computations efficiently. Twister has high performance compared with other similar implementations such as DryadLINQ and Hadoop for extremely large data parallel applications [62]. Table 8 contains the details about various aspects of Sector and Sphere as well as its pros and cons.

4.10 iMapReduce

iMapReduce is an adapted Hadoop runtime which allows users to specify the iterative computation with the separated Map and Reduce functions. It is used to perform the repeating algorithms based on a large cluster environment. The common features of repeating algorithms are achieved in this way and it provides the built-in support for these features. It presents the continuous tasks to reduce the job/task initial-

Table 7 The pros and cons of Twister and its components

Pros	Cons
Twister proves to be an efficient support for Iterative MapReduce computations (extremely faster than Hadoop or Dryad/DryadLINQ).	In Scheduling Task, Google's MapReduce and Hadoop use a dynamic scheduling mechanism which is more efficient than Twister.
It provides features to support MapReduce computations like distinction on static and variable data.	A possible disadvantage of this approach is that it does require the user to break up large datasets into multiple files.
It combines varied phases to collect all Reduce outputs.	–

Table 8 The advantages and disadvantage of Sector and Sphere and its components

Pros	Cons
Sector manages the large distributed datasets with high-reliability, high-performance I/O, and a uniform access. The sphere is used to simplify data access, increase data I/O bandwidth and exploit wide-area, high-performance networks using the sector-distributed storage system. Sphere presents a very simple programming interface by hiding data movement, load balancing, and fault tolerance.	Sector breaks up large datasets into multiple files and uses a device to complete it. Sector assumes that the user to develop code for working with large datasets is as complicated as the user to split a large dataset into multiple files if required.

ization overhead, provides efficient data management to avoid the shuffling of static data among tasks, and allows asynchronous Map task execution when it would be possible. Figure 8 shows the data flow in the MapReduce implementation on the left side. Each MapReduce should load the input data from DFS before the function starts. Then, it derives the intermediate key-value pairs because of the Reduce function operation on the intermediate data and it drives the output of this iteration, which is written to DFS. The same process happens in the next iteration since the Map function loads the iterated data from DFS again.

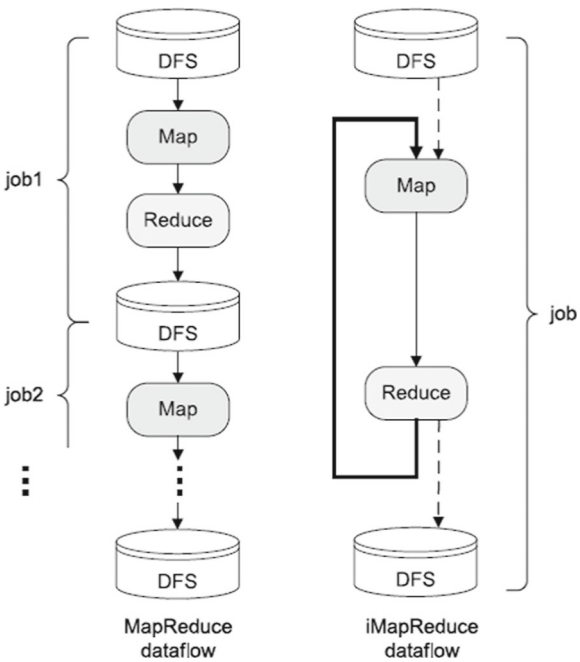

Fig. 8 The workflow of MapReduce and iMapReduce [81]

Additionally, the repeated DFS has an expensive loading/dumping. Hadoop provides locality optimization that reduces the remote communication. The same operations are done at each iteration. It means that the same Map and Reduce functions are done. iMapReduce uses Map/Reduce tasks persistently. That is the Map/Reduce operations in Map/Reduce tasks continues functioning till the iteration is completed. Further, iMapReduce enables the Reducer's output to be passed to the Map for the next round iteration [81].

The data flow in iMapReduce is shown on the left. The dashed line indicates that the data loading from DFS happens only once in the initialization stage, and the output data are written to DFS only once when the iteration completes [81]. Moreover, Table 9 contains details about various aspects of iMapReduce as well as its pros and cons.

5 MapReduce Applications

The MapReduce application facilitates the performance of many data parallel applications [62]. The MapReduce is the main factor in many important applications and it can improve system parallelism [82]. It gets considerable attention, for data-intensive and computation-intensive applications on machine clusters [83]. It is used as an efficient distributed computation tool for variable problems, e.g., search, clustering, log analysis, different types of join operations, matrix multiplication, pattern matching, and analysis of social networks [84]; It allows researchers to investigate in different domains. MapReduce is used in many big data application such as short message mining [85], genetic algorithms [86], k-means

Table 9 The pros and cons of iMapReduce and its components

Pros	Cons
iMapReduce results in the context of various applications, show up to 5 times of faster operation compared with traditional Hadoop MapReduce.	iMapReduce is not suitable for more iterative computations.

clustering algorithm [32], DNA fragment [87], intelligent transportation system [88], Healthcare scientific applications [89], Fuzzy rule based classification systems [90], heterogeneous environments [9], cuckoo search [91], extreme learning machine [92], Random Forest [93], energy proportionally [94], Mobile Sensor Data [95], semantic web [96] and so many. Therefore, in this section indicates the brief analyzes of these applications and a brief look to various domains of MapReduce applications.

5.1 Distributed Grep

Distributed Grep searches plain-text data sets for lines matching a regular expression [97]. It is used to search for a given pattern in a large number of files. Finding the top searched papers requires a web administrator to search web server logs in accordance with a given pattern.[2] Figure 9 illustrates the very popular example in order to explain how Map-Reduce works in this scope.

In Fig. 9, a line of input data is sent to the Map function; each Map function takes its line of input and splits it into words. The Mapper outputs a (key, value) pair of the word and the value 1. In the Reducer, the keys are grouped together and the values for similar keys are added. Thus, Reduce function forms an aggregation phase for keys.

5.2 Word Count

Word count is considered as MapReduce application which counts the occurrences of each word in large text data [12] and extracts a small amount of data from a large dataset [20]. Figure 10 gives an overview of how the MapReduce paradigm operates in order to count the number of words in a file; each instance of the Mapper receives one line of input. Each Mapper receipts its line of input and splits it into words. The Mapper outputs a (key, value) pair of the word and the value 1. Since all the lines are independent of each other, all Mappers run in parallel.

The word count operation takes place in three stages: Mapper, Shuffle and a Reducer. In Mapper, first the input file is split into words and then make alterations to key and value pairs with these words—the key is being the word itself and value '1'. For example, consider the sentence "Book Pen Apple Apple Pen Pen Ball Ball Book" in Mapper the sentence would be split as words and from the initial key value pair as <Book, 1> <Pen, 1> <Apple, 1> <Apple, 1> <Pen, 1> <Ball, 1> < Ball, 1> <Book, 1>. After the Map task is complete, the body of intermediate data is transferred from Mapper to Reducer in the shuffle phase. The data transfer by shuffle happens from the Mapper disks rather than their main memories and the intermediate result will be sorted by the keys to group the pairs with the same keys together. In Reducer, the keys are grouped together, and the values for similar keys are added. So there are only one pair of similar keys 'Book' the values for these keys would be added so the output key-value pairs would be <Book, 2> <Pen, 3> <Apple, 2> <Ball, 2> this would give the number of occurrence of each word in the input and Reducer forms an aggregation phase for keys and shows the final result as Fig. 10 clearly reveals the word count process.

5.3 TeraSort

TeraSort is a standard MapReduce sorting algorithm with a custom Reducer in which each reducer receives a sorted list of N – 1 sampled keys with predefined ranges [98]. This package includes three Map/Reduce applications for Hadoop to compete in the annual terabyte competition.

- Termagant is a Map/Reduce program in which the data is produced.

[2] http://web.cs.wpi.edu/~cs4513/d08/OtherStuff/MapReduce-TeamC.ppt

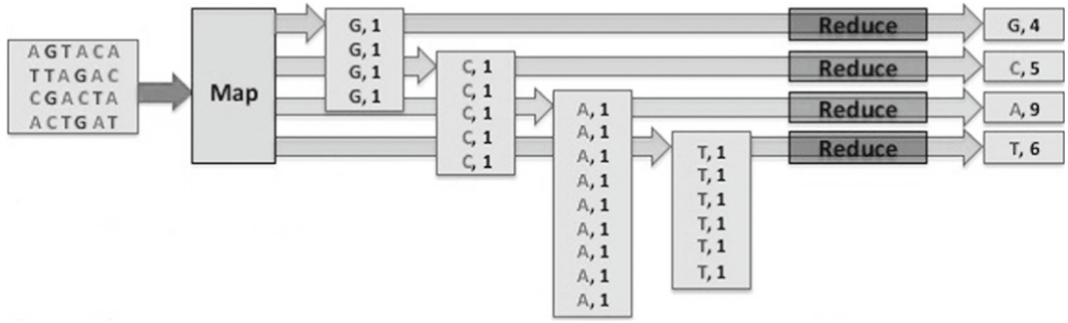

Fig. 9 The overall MapReduce distributed Grep process (http://www.slideshare.net/romain_jacotin/the-google-mapreduce)

- The input data is sorted by TeraSort and it uses Map/Reduce to sort the data into a total order.
- TeraValidate is a Map/Reduce program, which makes possible to validate the sorted output.
- TeraSort is a standard Map/Reduce type but it does not include a custom partitioner that uses a sorted list of N-1 sampled keys that define the key range for each Reduce. In specific, it sends all keys such as the sample *[i-1] < =key <sample[i]* to Reduce i. TeraValidate certifies the globally sorted of the output. It creates one Mapper a file in the output directory and each Map ensures that each key is less than or equal to the previous one [99].

5.4 Inverted and Ranked Inverted Index

An inverted index is utilized to store a mapping from content, such as words or numbers, to its location in a database file or a document. MapReduce application takes a list of documents as input and produces word-to-document indexing. Basically, each document is

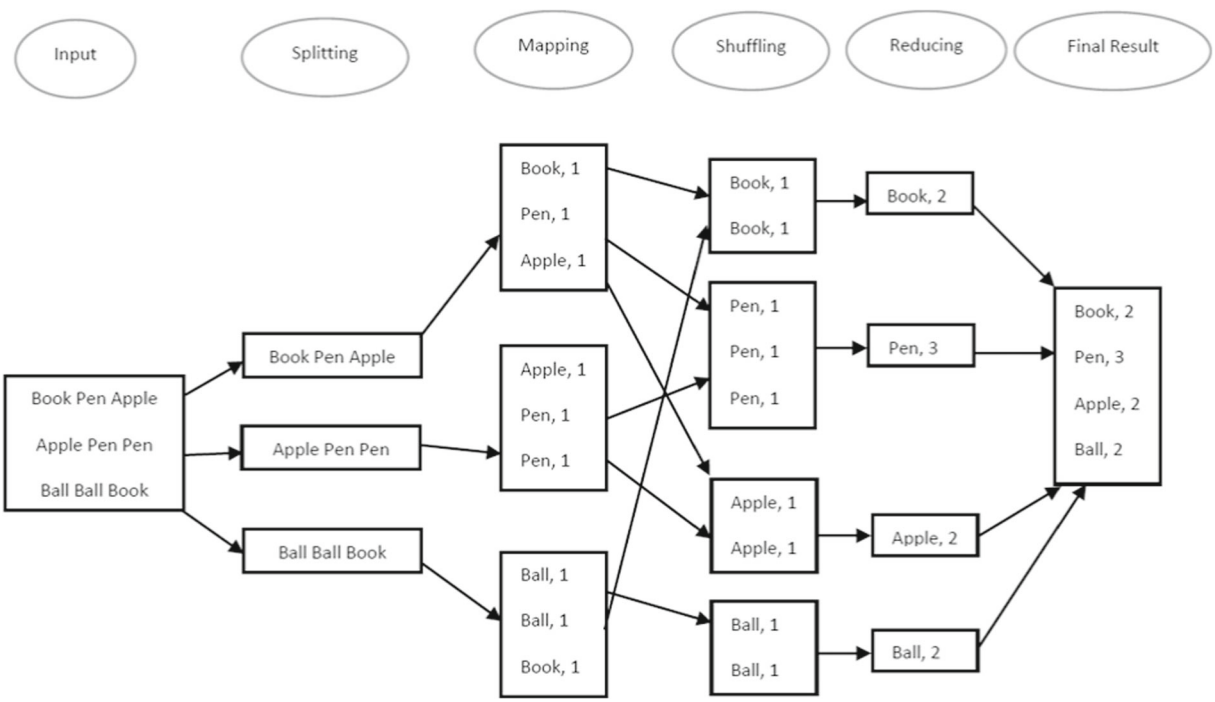

Fig. 10 The overall MapReduce word count process

analyzed by the Mapper and emits a sequence of <word, document ID> pairs. The Reducer accepts all pairs for a given word, sorts the corresponding document IDs and emits a < word, list (document ID) > pair. A simple inverted index is created using all output pairs. This computation should be enhanced to follow up word positions.[3]

One type of MapReduce application is called Ranked-inverted-index which takes a list of words and their incidences of each document and produces the lists of documents containing the given words to decrease frequency [97]. Furthermore, it takes lists of words and their frequencies in a file and generates lists of files containing the given words in decreasing order of frequency [100].

5.5 Term-Vector

A MapReduce defines the most frequent words in a host, which is useful in analyzing of a host's relevance to a search. Map phase emits *<host, term vector>* tuples where *term vector* is itself a tuple of the form *<word, 1>*. The words with below frequency are discarded by reducing phase., It sorts the rest of the list considering to count and emits tuples of the form *<host, list(term vector)>* [101].

A term vector may look like this [102]:

[("word1", 8), ("word2", 4), ("word3", 7)]

A Map phase is used to represent a term vector. Reduce phase then receives a set of term vectors for a given host from several Map instances:

("hostX", [("word1", 8), ("word2", 4), ("word3", 7)])
("hostY", [("word1", 3), ("word5", 6), ("word3", 3)])
("hostZ", [("word1", 1), ("word6", 3), ("word2", 9)])

5.6 Random Forest

Random forests (RF) is very popular in classification, prediction, studying variable importance, variable selection, outlier detection, and classification. The RF popularity is due to its high performance in relation with other classification algorithms [103]. The techniques have an influence on big data by fast, scalable and parallel implementations. MapReduce is the best solution to get this goal [104]. RF classifier helps this to handle this technique to handle imbalanced datasets in the big data scenario. Specifically, oversampling, undersampling and cost-sensitive learning have been adapted to big data using MapReduce. In consequence, these techniques are able to manage datasets as large as needed to correctly identify the underrepresented class. The Random Forest classifier provides a solid basis for the comparison because of its performance, robustness, and versatility [93].

5.7 Spark

Spark as a cluster computing framework supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce. It is used to maintain the scalability and fault tolerance of MapReduce by means of an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only group of objects, which are divided by a set of machines that can be rebuilt if a partition is lost. The main idea in Spark is the construction of resilient distributed dataset (RDD), which is a read-only collection of objects maintained in memory across iterations and supports fault recovery. Spark provides three simple data abstractions for programming clusters: resilient distributed datasets (RDDs), and two restricted types of shared variables, namely, broadcast variables and accumulators. These abstractions have the ability to represent some challenges by regarding its limitation for existing cluster computing frameworks, including iterative and interactive computations [105].

5.8 Extreme Learning Machine

Extreme learning machine (ELM) has gained momentum from various research fields in MapReduce framework. applications due to its Fast convergence and good generalization performance are considered as its features [106]. Different types of ELM have been proposed in practical applications [107], such as basic ELM [108], random hidden layer feature

[3] http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce

mapping based ELM [109], incremental ELM [110], kernel based ELM [111], etc. Online sequential extreme learning machine (OS-ELM) is one of the improved extreme learning machine algorithms to support online sequential learning efficiently. It analyzes the dependency relationships of matrix calculations of OS-ELM and proposes a parallel OS-ELM (POS-ELM) based on MapReduce [1]. However, it is able to deal with very large-scale training dataset in big data applications. Elastic ELM is a new MapReduce-based elastic extreme learning machine framework, which covers the shortage of ELM whose learning ability is weak for updated large-scale training dataset [106]. Due to the exponentially increasing volume of training data in massive learning applications, the centralized ELM with kernels suffers from the great memory consumption of large matrix operations. Also, it has high communication cost, which does not allow these matrix operations to directly apply on shared-nothing distributed computing model like MapReduce. Distributed Kernelized ELM (DK-ELM) is an implementation of ELM with kernels on MapReduce [92].

5.9 DNA Fragment

A DNA sequence contains the genetic information of an organism. To understand the organism's build, it is necessary to achieve the complete sequence of DNA and understanding the structure and composition of a gene. The sequence assembly technology is the basis of DNA sequence data processing. Fragment assembly is one of the most important problems of sequence assembly. Algorithms for DNA fragment assembly using de Bruijn graph have been widely used but these algorithms require a large amount of memory and running time to be formulated. De Bruijn approach suffers from the loss of information. Three key factors must be considered to develop a sequence assembly parallel algorithm, including avoiding graph division which can cause an adverse impact on splicing results, overcoming memory limitations in large sequence assembly by distributing de Bruijn graph, and putting MapReduce to use, is an ideal solution for sequence assembly algorithm. MapReduce separates de Bruijn graph division and the employment of this framework as well as developing a sequence assembly parallel algorithm, which can effectively improve the computational efficiency and remove the memory limitations of the assembly algorithm [87].

5.10 Mobile Sensor Data

Mobiles are equipped with different sensors like accelerometer, magnetic field, and air pressure meter, which help in the process of the extracting situational information about the user like location, state, etc. One of the massive jobs is to provide the extracted sensor data in which data is removed from mobiles to the public cloud. The utilization of parallel computing using MapReduce on the cloud for training and recognition of human activities from accelerometer sensor data is based on classifiers that can easily scale in performance and accuracy in this process. It aims to recognize and predict the human activity patterns. The data should be gathered as close as possible to real scenarios for training goals. In general, data collected in laboratories does not cover the activity patterns that people use in their life. Furthermore, gathering training data for such human activity pattern discovery is a tricky process, especially when it involves embedded sensors such as the accelerometer in mobile devices. Accelerometer sensors must be set in smartphones to face these problems. It is used for collecting the data in real scenarios and cloud services and storing training repositories, which are further classified using parallel computing (MapReduce framework) in the cloud. The user helps in gathering real accelerometer data which is further used for training and prediction purposes by the classifiers in the cloud [95].

5.11 Social Networks

The Social Networks Service (SNS), is becoming more and more popular and an exponential growth in a number of empirical studies have been carried out in this active field [112–114]. Nowadays, it not suitable to use the traditional analysis methods due to the rapid growth of the network. MapReduce framework can solve large-scale social network analysis problem by making use of the power of multi-machines, which is qualified in processing large-scale social network data. Hadoop takes the place of many traditional anal-

ysis methods to conduct a series of analyses on large-scale social networks, including several distributions like clustering coefficient and diameter [115]. A social network, the graph of relationships and interactions within a group of individuals, plays a fundamental role in spreading information, ideas, and influence among all its members. Due to its importance, existing algorithms of influence maximization problem (IMP) for a social network are not efficient enough to cope with real-world social networks. MapReduce and Hadoop should be parallelized to handle big social networks as the platform or other implementations of MapReduce [116].

5.12 Algorithms

Algorithms are the key factors in developing or engineering the software and applications. They have always been indispensable in all aspects of computer sciences. Algorithm plays an important role in most of the computer sciences, including, networks, databases, web technologies or Core Programming, without algorithms none of them would operate. This section introduces some algorithms that are implemented with MapReduce framework to allow the handling of data-intensive applications, abstractions for parallelism, and fault control.

5.12.1 Genetic Algorithm

Genetic Algorithms are a class of development algorithms. The use of Genetic Algorithm is a quickly evolving field of research in numerous spheres such as chemistry, bioinformatics, economics, biology, etc. It is better to use Parallel Genetic Algorithms since processing Genetic Algorithms generally takes a very long time for large problems. The iteration method of Genetic Algorithms cannot directly be executed with Map and Reduce functions. MapReduce should be extended to support such algorithms and Parallel Genetic Algorithm needs a phase for global selection at the end of every iteration. This requires a coordinator client for coordinating the execution of Parallel Genetic Algorithms iterations which are achieved by adding a second Reduce phase after the iterations and a client for coordinating the implementation of iterations.

Figure 11 illustrates an MRPGA (MapReduce for Parallelizing Genetic Algorithms) architecture. First of all, the coordinator generates offspring and performs mutation. Then, it sends the offspring to the master for evaluation. The master divides the offspring into m pieces for m Map tasks. The value of m is selected so as to maximize parallelism for Map tasks. Commonly, this value is greater than the number of machines. Each piece of offspring is sent to a machine with a Mapper worker. Each Reducer worker is assigned with Reduce tasks for the 1st phase of reducing operations. Normally, the input is taken from the local machine. The Reduce function is used to choose the local optimum individuals that are then kept on the local machine. Also, a Reducer worker is allocated to perform the final Reduce function. This worker collects all the results generated in the

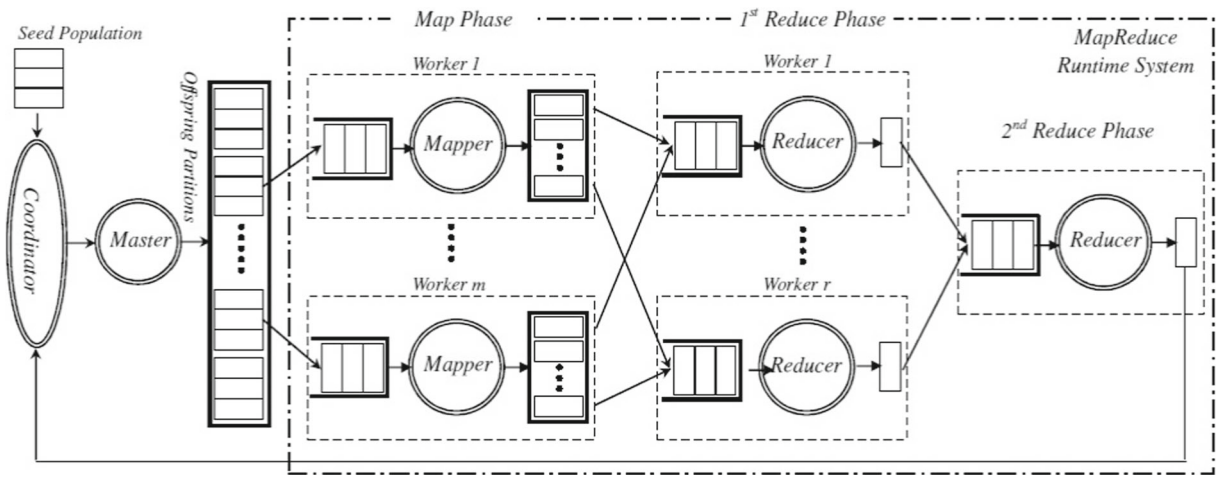

Fig. 11 Architecture MRPGA [86]

1st phase of reducing operation. The final Reduce function is invoked to produce the global optimum individuals as final results. The final results are sent to the client for the next round of the evolutionary algorithm [86].

5.12.2 PSO Algorithm

Particle Swarm Optimization (PSO) algorithm was first introduced by Eberhart and Kennedy [117]. It is inspired by experiments with simulated bird flocking and became popular because of its simplicity and little tuning efforts [118, 119]. Large amounts of data suffer from web content, commercial transaction information, or bioinformatics data, which requires minutes or hours for each function evaluation. To optimize such functions, PSO must be parallelized. MapReduce framework shows this parallelization in PSO model without explicitly addressing any of the details [120].

5.12.3 Cuckoo Search Algorithm

Cuckoo search (CS) is a new metaheuristic optimization algorithm for solving structural optimization tasks [121, 122]. A new approach is used to incorporate cuckoo search to the analysis of big data using latest cloud computing with MapReduce paradigm, which has been widely applied to solve large-scale data-intensive problems [123]. It combines the divide-and-conquer algorithm with MapReduce programming paradigm to apply them in the sequential cuckoo search method. Finally, this approach has a much better runtime performance when processing large dataset [91].

5.12.4 Ant Colony Algorithm

Ant colony algorithm (ACO) is a class of metaheuristics, which can solve combinatorial optimization problems effectively by the behavior of real ants [124–126]. This algorithm is considered as a form of adaptive memory programming [127]. In the algorithm, the social behavior of ants are simulated and it could be a good choice to existing algorithms. Parallelization is the most effective way to solve large-scale ant colony optimization algorithms over a large dataset. MapReduce algorithm is used to solve the problem by available methods and it makes sufficient use of the simplicity and scalability of the MapReduce model. As shown in Fig. 12, in the Search Space Replication Approach, the input is replicated m copies, where m is the number of Map tasks.

Each Mapper executes the ACO algorithm separately while emitting its output with a constant integer as the key and its ACO result as output. Since the outputs have the same keys, they will be sent to the only Reducer. In the Reducer, the optimal value is chosen and recorded as output. In the first approach, the input is not divided and no information exchange happens between execution units. In the second approach, the solution space is divided to Mapper. The segmentation is done carefully and each part has a convenient location to show how ACO algorithms can be modeled into the MapReduce framework and implementation of ACO on Hadoop [128].

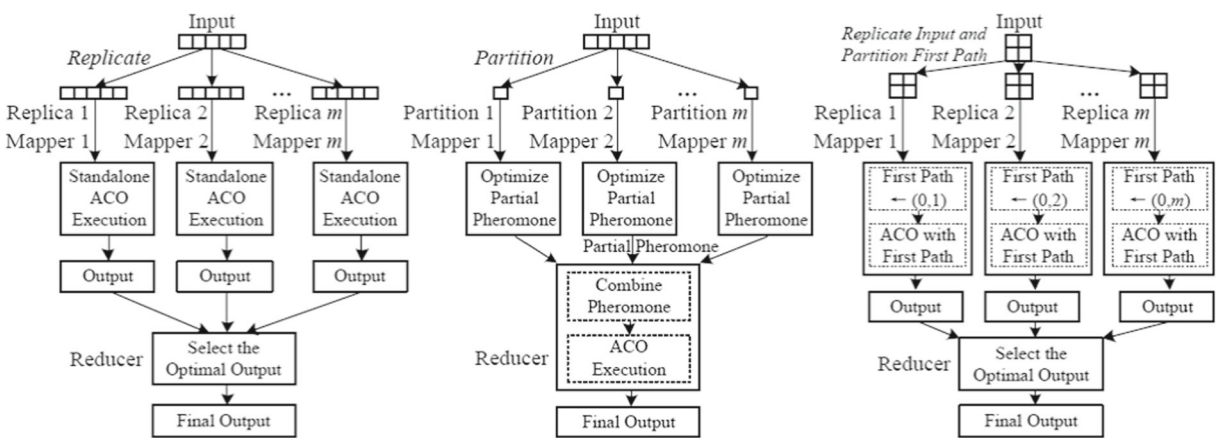

Fig. 12 Parallel ACO approach with MapReduce [128]

5.12.5 Fuzzy Algorithm

The era of big data has great volume, high speed, and complex structures. Data can be produced by offline and online transactions, sensors, social networks, Worldwide Web and through our daily life activities. Big data applies Fuzzy sets due to their abilities to represent and quantify aspects of uncertainty [129]. For example, underwater image segmentation has to deal with the rapidly increasing volume of images and videos. The MapReduce framework as a workable solution is used to manage this issue. Therefore, a MapReduce-based fast fuzzy c-means algorithm (MRFFCM) has been offered to paralyze the segmentation of the images. It uses a two-layer distribution model to cluster the large-scale images and adopt an iterative MapReduce process to parallelize the FFCM algorithm. Also, a combinational segmentation technique is utilized to raise the algorithm's efficiency. The improvement of fuzzy c-means algorithm and the use of parallel programming model speed up the processing of massive data [130].

6 Results and Comparison

MapReduce is one of the most well-known and significant parallel processing techniques in cloud computing. MapReduce is a programming model for large-scale parallel processing, or a software framework in Hadoop or other platforms such as GridGain, Phoenix, Twister, and Mars. Each platform that implements the concept of MapReduce has different properties in specific provinces. Nevertheless, Apache Hadoop is an open-source software framework implementation of MapReduce is being improved by many software researchers and developers. Many companies and researchers use big data by means of Hadoop in the past few years although it requires improvements [131].

Hadoop and Sphere, mostly adopt the initial programming model and the architecture presented by Google. These architectures concentration is on performing single step MapReduce with an optimized fault tolerance. It is possible to store most of the data outputs to some form of file system throughout the computation. A new *Map/Reduce* task is created in each iteration by loading or accessing any static data repetitively in these runtimes. These features can be justified for single-step MapReduce computations and they introduce considerable performance overheads for many iterative applications, for instance, iMapReduce, a distributed framework that supports iterative processing. It allows users to specify the iterative computation with the Map and Reduce functions and to support the automatic iterative processing within a job [81]. In Hadoop, each Mapper and Reducer may generate zero or more key/value pairs. Hadoop presents HDFS that provides storing to 1000's of nodes and Petabyte of data reliability and scalability. It provides an automatically distributed sort of the data between the Maps and Reduces. Hadoop uses a much more flexible serialization, which can use either java.io or user-defined serialization. It provides a web interface to track the job's progress. Hadoop supports both C++ and text-based applications and it has more supporters than GridGain. But, GridGain is not suitable for large data sets.

In GridGain system, each Mapper and Reducer returns a single value. It provides only distributed computation support and does not have a distributed file system. GridGain's framework does not have the data classification between the Maps and the Reduce stage like Hadoop. Counters or combiners are not supported by GridGain implementation. It does not provide a web interface to track the job's progress. Also, it does not have any support for non-Java applications [73]. Hadoop can easily handle analytical risk over big data. It controls the processing of the input datasets completely. However, MapReduce can be easily used by developers without having much experience and knowledge of databases, but with a little knowledge of Java. It gives a satisfied and acceptable performance in scaling large clusters for developers [132]. Hadoop and GridGain both realize the MapReduce framework. Hadoop has a relatively high latency because of HDFS. On the contrary, GridGain is a computational tool of MapReduce and has low latency [63].

Mars has low performance in comparison with its CPU-based counterparts and the native implementation without MapReduce. Mars proves the effectiveness of GPU-oriented optimization strategies. On average, the MarsCUDA is 22 times quicker than the CPU-based MapReduce, Phoenix, and is three times slower than the hand-tuned native CUDA implementation. Additionally, the applications developed with Mars had a code size reduction up to seven times, compared with hand-tuned native CUDA code [61]. Mars develops on an NVIDIA G80 GPU, which has

one hundred processors and is evaluated in comparison with Phoenix. It hides the programming complexity of the GPU behind the simple and familiar MapReduce interface. It is up to 16 times quicker than its CPU-based counterpart for six common web applications on a quad-core machine [76]. Table 10 provides details about various aspects of MapReduce implementations.

Also, this article aims to show various existing MapReduce applications connecting with other algorithms and frameworks to handle big data easily and to render them time and cost-effective for programmers in performance tuning. This paper comprehensively explains the MapReduce applications such as WordCount, Distributed grep, TeraSort, Inverted and Ranked Inverted Index and Term-vector and how these typical MapReduce applications work. Developers are able to write and create applications to process huge volumes of unstructured data in parallel, across a distributed cluster of processors such as extreme learning machine, multi-core systems, social networks and DNA fragment using MapReduce framework. This paper shows how MapReduce could cope with algorithms to the handling and supporting of data-intensive applications and provide high scalability to supports automatic parallelization, distribution of computations, fault tolerance, and task management. Table 11 shows the advantages and disadvantages of MapReduce applications.

7 Open Issues

The discussion of the human challenges and future directions of research continue in the era of big data with a great set of digital information [133]. Big data and large-scale data processing techniques have become an important developing area. Big data applications create development in the diverse large-scale data management systems in various organizations from traditional database vendors to new emerging Internet-based enterprises [134]. MapReduce is an enabling technology of cloud computing. Big data systems and the MapReduce model are an active research

Table 10 The pros and cons of MapReduce implementations

MapReduce implementations	Pros	Cons
Google MapReduce	It makes a duplicate of data blocks on multiple nodes for enhanced reliability and fault tolerance.	It is a batch-based architecture which means it does not lend itself to use cases that need real-time data access.
Hadoop	Scalability and availability are the distinguishing features to achieve data replication and fault tolerance system.	Cluster management is difficult in Hadoop. In the cluster, operations like debugging, distributing software, collection logs are too hard.
GridGain	GridGain approach of giving tasks to the control of sub-task distribution, enables early and late load balancing algorithms.	It doesn't have any support for non-Java applications. GridGain has only one supporter that is GridGain.
Mars	Mars is the first MapReduce implementation on GPUs. Mars exploits the massive thread parallelism within the GPU.	Its implementation does not employ atomic operations, it requires expensive preprocessing kernels to coordinate output from different threads to the global memory.
Tiled-MapReduce	Providing convergence and good generalization performance.	The cost of implementation is high.
Phoenix	Phoenix is a well-organized implementation on the multi-core CPU.	The scalability of Phoenix does not appear to be enhanced.
Twister	It supports typical MapReduce computations and has tools to manage data easily.	A possible disadvantage of this approach is that it does require the user to break up large datasets into multiple files.

Table 11 The pros and cons of MapReduce applications

MapReduce applications	Pros	Cons
Distributed Grep	A generic search tool used in many data analyses.	Suffering from prolonged response time in large clusters.
Word count	Counting the occurrences of each word and massive document collection.	Limiting memory usage.
TeraSort	Offers load balancing time in large clusters.	Suffering from overhead.
Inverted and ranked inverted index	Consisting of a collection of postings lists, one associated with each unique term in the collection.	Lots of pairs to sort and shuffle.
Term-vector	Useful in analyses of a host's relevance to a search.	Requiring many sequential tasks.
Random Forest	Provides high scalability.	It has been observed to over fit for some datasets with noise classification.
Extreme Learning Machine	Providing convergence and good generalization performance.	It causes an additional uncertainty problem, both in approximation and learning.
Spark	Spark performs better when all the data fits in the memory, especially on dedicated clusters.	Spark needs a lot of memory.
Algorithms	Handling of data-intensive applications.	It is time-consuming.
DNA fragment	Developing a sequence assembly parallel algorithm.	Require a large amount of memory.
Mobile Sensor Data	Help in the process of the extracting context of the user like location, situation etc.	Difficult to implement.
Social Networks	Access large samples of respondents quickly.	Develop appropriate data analytic techniques for huge samples.

area, which are in the early stages. Although many vital steps forward have been made, there are still several open issues in this area. There lies a bright future ahead for Big Data, it has an important role in businesses and organizations realizing the value of analyzing and storing information. Developing ways to process the vast amounts of data available in business innovation, health discoveries, algorithms, bioinformatics, and science progress allow us to find novel ways to solve problems. Nowadays, with MapReduce framework, a large amount of data can easily process and analyze the impossible problems in the past. The application of MapReduce concepts and techniques has yet to blossom to its full potential in different domains. There are still some MapReduce application techniques that have not been examined adequately in parallel computation, graph processing, multi-core systems, optimizing frameworks and data allocation. Various open issues are considered to organize promising directions for future research for example how to Optimize execution model based on different application and cluster characteristics, new communication strategy, how mappers could communicate with each other easily and faster and overlapping phases to improve the MapReduce architecture.

This survey has already outlined the framework of knowledge discovery in MapReduce applications, it also brings several new future works. One interesting work in the future includes Identifying different patterns and applications of the MapReduce and optimizing MapReduce model based on different application and cluster characteristics or on the heterogeneous cluster are considered as future goals. These may improve the system performance and increase fault tolerance on different applications. Cloud computing considers the most promising technology due

to its cost-efficiency, flexibility, and its vision of unlimited and unrestricted access to computing for everybody [135–137]. In the cloud environment, the fault-tolerance for data sets and database is a challenging research issue due to increasingly larger volumes of datasets for example, when nodes are malware and malicious, the final results may be untrustworthy and inaccurate [138, 139]. One of the interesting future research refers to the MapReduce model to increase the fault-tolerance in the database in the wider area. There are many open issues for big data and MapReduce applications, in particular in the computational algorithms, multicore systems, data allocation and many other fields. Some characteristics and open issues of these challenges have been discussed in this paper, such as parallel computation aspects and the capability of being flexible enough to collect and analyze a different kind of information. The MapReduce is exponentially growing up in different domains in the world in an uncontrollable way.

8 Conclusion

MapReduce has efficiency and scalability in most of the studies [140]. It is used for generating and processing big Data in various different applications. The purpose of this essay is to review, MapReduce, its architecture, big data and an appropriate use of programming model in conjunction with the applications of MapReduce in big data have been discussed thoroughly. Also, we have surveyed and analyzed the implementations of MapReduce. The applications of MapReduce framework in different contexts like the cloud, multi-core system, and parallel computation have been investigated precisely. This paper examines and categorized a number of applications which have been surveyed in MapReduce Framework based on Graph processing, Join and parallel queries, optimizing frameworks, multi-core systems, and data allocation. The goal of the MapReduce Framework is to provide an abstraction layer between the fault-tolerance, data distribution and other parallel systems tasks, and the implementation details of the specific algorithm. Obviously, the requirements of MapReduce applications is growing rapidly. This survey gives the reader a general review of the MapReduce applications and it will be a good introductory reference to improve the article that is easier to comprehend.

References

1. Wang, B., Huang, S., Qiu, J., Liu, Y., Wang, G.: Parallel online sequential extreme learning machine based on MapReduce. Neurocomputing **149**, 224–232 (2015)
2. Marozzo, F., Talia, D., Trunfio, P.: P2P-MapReduce: parallel data processing in dynamic Cloud environments. J. Comput. Syst. Sci. **78**, 1382–1402 (2012)
3. Mohamed, H., Marchand-Maillet, S.: MRO-MPI: MapReduce overlapping using MPI and an optimized data exchange policy. Parallel Comput. **39**, 851–866 (2013)
4. Barre, B., Klein, M., Soucy-Boivin, M., Ollivier, P.-A., Hallé, S.: MapReduce for parallel trace validation of LTL properties. In: Runtime Verification, pp. 184–198 (2013)
5. Lu, L., Shi, X., Jin, H., Wang, Q., Yuan, D., Wu, S.: Morpho: a decoupled MapReduce framework for elastic cloud computing. Futur. Gener. Comput. Syst. **36**, 80–90 (2014)
6. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. ACM **53**, 72–77 (2010)
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
8. Kolb, L., Thor, A., Rahm, E.: Multi-pass sorted neighborhood blocking with MapReduce. Comput. Sci. Res. Dev. **27**, 45–63 (2012)
9. Anjos, J.C., Carrera, I., Kolberg, W., Tibola, A.L., Arantes, L.B., Geyer, C.R.: MRA++: scheduling and data placement on MapReduce for heterogeneous environments. Futur. Gener. Comput. Syst. **42**, 22–35 (2015)
10. 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. **55**, 896–907 (2014)
11. Slagter, K., Hsu, C.-H., Chung, Y.-C., Yi, G.: SmartJoin: a network-aware multiway join for MapReduce. Clust. Comput. **17**, 1–13 (2014)
12. Xiao, Z., Xiao, Y.: Achieving accountable MapReduce in cloud computing. Futur. Gener. Comput. Syst. **30**, 1–13 (2014)
13. Plantenga, T.D., Choe, Y.R., Yoshimura, A.: Using performance measurements to improve mapreduce algorithms. Procedia Comput. Sci. **9**, 1920–1929 (2012)
14. Polato, I., Ré, R., Goldman, A., Kon, F.: A comprehensive view of Hadoop research—a systematic literature review. J. Netw. Comput. Appl. **46**, 1–25 (2014)
15. Shamsi, J., Khojaye, M.A., Qasmi, M.A.: Data-intensive cloud computing: requirements, expectations, challenges, and solutions. J. Grid Comput. **11**, 281–310 (2013)
16. Plimpton, S.J., Devine, K.D.: MapReduce in MPI for large-scale graph algorithms. Parallel Comput. **37**, 610–632 (2011)
17. Wolf, J., Balmin, A., Rajan, D., Hildrum, K., Khandekar, R., Parekh, S., et al.: On the optimization of schedules for MapReduce workloads in the presence of shared scans. VLDB J.—Int. J. Very Large Data Bases **21**, 589–609 (2012)
18. Aznoli, F., Navimipour, N.J.: Cloud services recommendation: Reviewing the recent advances and suggesting the future research directions. J. Netw. Comput. Appl. **77**, 73–86 (2017)
19. Vakili, A., Navimipour, N.J.: Comprehensive and systematic review of the service composition mechanisms in the

cloud environments. J. Netw. Comput. Appl. **81**, 24–36 (2017)
20. Yang, H., Luan, Z., Li, W., Qian, D.: MapReduce workload modeling with statistical approach. J. Grid Comput. **10**, 279–310 (2012)
21. Choi, J., Choi, C., Ko, B., Kim, P.: A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment. Soft Comput. **18**, 1697–1703 (2014)
22. Chiregi, M., Navimipour, N.J.: A new method for trust and reputation evaluation in the cloud environments using the recommendations of opinion leaders' entities and removing the effect of troll entities. Comput. Hum. Behav. **60**, 280–292 (2016)
23. Chiregi, M., Navimipour, N.J.: A comprehensive study of the trust evaluation mechanisms in the cloud computing. J. Serv. Sci. Res. **9**, 1–30 (2017)
24. Navimipour, N.J., Rahmani, A.M., Navin, A.H., Hosseinzadeh, M.: Expert Cloud: a Cloud-based framework to share the knowledge and skills of human resources. Comput. Hum. Behav. **46**, 57–74 (2015)
25. Keshanchi, B., Souri, A., Navimipour, N.J.: An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. J. Syst. Softw. **124**, 1–21 (2017)
26. Hazratzadeh, S., Navimipour, N.J.: Colleague recommender system in the Expert Cloud using the features matrix. Kybernetes **45**, 1–30 (2017)
27. Mohammadi, S.Z., Navimipour, J.N.: Invalid cloud providers' identification using the support vector machine. Int. J. Next-Generation Comput. **8**, 82–89 (2017)
28. Zhang, J., Xiang, D., Li, T., Pan, Y.: M2M: a simple Matlab-to-MapReduce translator for cloud computing. Tsinghua Sci. Technol. **18**, 1–9 (2013)
29. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proc. VLDB Endow. **5**, 716–727 (2012)
30. Cormack, G.V., Smucker, M.D., Clarke, C.L.: Efficient and effective spam filtering and re-ranking for large web datasets. Inf. Retr. **14**, 441–465 (2011)
31. Lin, J.: Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 155–162 (2009)
32. Zhao, W., Ma, H., He, Q.: Parallel k-means clustering based on mapreduce. In: Cloud Computing, pp. 674–679. Springer, Berlin (2009)
33. Baraglia, R., De Francisci Morales, G., Lucchese, C.: Document similarity self-join with MapReduce. In: 2010 IEEE 10th International Conference on Data Mining (ICDM), pp. 731–736 (2010)
34. Caruana, G., Li, M., Liu, Y.: An ontology enhanced parallel SVM for scalable spam filter training. Neurocomputing **108**, 45–57 (2013)
35. Liao, R., Zhang, Y., Guan, J., Zhou, S.: CloudNMF: a MapReduce implementation of nonnegative matrix factorization for large-scale biological datasets. Genomics Proteomics Bioinforma. **12**, 48–51 (2014)
36. Svendsen, M., Tirthapura, S.: Mining maximal cliques from a large graph using MapReduce: tackling highly uneven subproblem sizes. J. Parallel Distrib. Comput. **79**, 104–114 (2012)
37. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with MapReduce: a survey. ACM SIGMOD Rec. **40**, 11–20 (2012)
38. Li, R., Hu, H., Li, H., Wu, Y., Yang, J.: Mapreduce parallel programming model: a state-of-the-art survey. Int. J. Parallel Prog. **44**, 832–866 (2016)
39. Khezr, S.N., Navimipour, N.J.: MapReduce and its application in optimization algorithms: a comprehensive study. Majlesi J. Multimed. Process. **4**, 31–33 (2015)
40. Vijayalakshmi, V., Akila, A., Nagadivya, S.: The survey on MapReduce. Int. J. Eng. Sci. Technol. **4**, 3335–3342 (2012)
41. Kalavri, V., Vlassov, V.: Mapreduce: limitations, optimizations and open issues. In: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1031–1038 (2013)
42. Debortoli, S., Müller, O., vom Brocke, J.: Comparing business intelligence and big data skills. Bus. Inf. Syst. Eng. **6**, 289–300 (2014)
43. Lin, J., Dyer, C.: Data-intensive text processing with MapReduce. Synth. Lect. Human Lang. Technol. **3**, 1–177 (2010)
44. Jain, R., Sarkar, P., Subhraveti, D.: Gpfs-snc: an enterprise cluster file system for big data. IBM J. Res. Dev. **57**, 5:1–5:10 (2013)
45. Lee, D., Kim, J.-S., Maeng, S.: Large-scale incremental processing with MapReduce. Futur. Gener. Comput. Syst. **36**, 66–79 (2014)
46. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, pp. 2–2 (2012)
47. Zhao, Y., Wu, J.: Dache: a data aware caching for big-data applications using the MapReduce framework. In: INFOCOM, 2013 Proceedings IEEE, pp. 35–39 (2013)
48. Costa, P., Donnelly, A., Rowstron, A.I., O'Shea, G.: Camdoop: exploiting in-network aggregation for big data applications. In: NSDI, pp. 3–3 (2012)
49. Pandey, S., Tokekar, V.: Prominence of MapReduce in Big Data Processing. In: 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT), pp. 555–560 (2014)
50. Ji, C., Li, Z., Qu, W., Xu, Y., Li, Y.: Scalable nearest neighbor query processing based on Inverted Grid Index. J. Netw. Comput. Appl. **44**, 172–182 (2014)
51. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. J. Grid Comput. **13**, 1–37 (2015)
52. Wu, T.-Y., Chen, C.-Y., Kuo, L.-S., Lee, W.-T., Chao, H.-C.: Cloud-based image processing system with priority-based data distribution mechanism. Comput. Commun. **35**, 1809–1818 (2012)
53. Senger, H., Gil-Costa, V., Arantes, L., Marcondes, C.A.C., Marín, M., Sato, L.M., et al.: BSP cost and scalability analysis for MapReduce operations. Concurr. Comput. Pract. Exp. **28**, 2503–2527 (2016)

54. Idris, M., Hussain, S., Ali, M., Abdulali, A., Siddiqi, M.H., Kang, B.H., et al.: Context-aware scheduling in MapReduce: a compact review. Concurr. Comput. Pract. Exp. **27**, 5332–5349 (2015)
55. Lee, C.-W., Hsieh, K.-Y., Hsieh, S.-Y., Hsiao, H.-C.: A dynamic data placement strategy for Hadoop in heterogeneous environments. Big Data Res. **1**, 14–22 (2014)
56. Aridhi, S., d'Orazio, L., Maddouri, M., Mephu Nguifo, E.: Density-based data partitioning strategy to approximate large-scale subgraph mining. Inf. Syst. **48**, 213–223 (2015)
57. Ding, L., Wang, G., Xin, J., Wang, X., Huang, S., Zhang, R.: ComMapReduce: an improvement of mapreduce with lightweight communication mechanisms. Data Knowl. Eng. **88**, 224–247 (2013)
58. Laclavík, M., Šeleng, M., Hluchý, L.: Towards large scale semantic annotation built on mapreduce architecture. In: Computational Science–ICCS 2008. Springer, pp. 331–338 (2008)
59. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. In: ACM SIGOPS Operating Systems Review, pp. 59–72 (2007)
60. Yoo, R.M., Romano, A., Kozyrakis, C.: Phoenix rebirth: scalable MapReduce on a large-scale shared-memory system. In: IEEE International Symposium on Workload Characterization, 2009. IISWC 2009, pp. 198–207 (2009)
61. Fang, W., He, B., Luo, Q., Govindaraju, N.K.: Mars: accelerating mapreduce with graphics processors. IEEE Trans. Parallel Distrib. Syst. **22**, 608–620 (2011)
62. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., et al.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 810–818 (2010)
63. Pan, J., Biannic, Y.L., Magoules, F.: Parallelizing multiple group-by query in share-nothing environment: a MapReduce study case. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 856–863 (2010)
64. Aarnio, T.: Parallel data processing with MapReduce. In: TKK T-110.5190, Seminar on Internetworking (2009)
65. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: ACM SIGOPS Operating Systems Review, pp. 29–43 (2003)
66. Liu, Y., Li, M., Alham, N.K., Hammoud, S.: HSim: a MapReduce simulator in enabling cloud computing. Futur. Gener. Comput. Syst. **29**, 300–308 (2013)
67. Wang, L., Tao, J., Ranjan, R., Marten, H., Streit, A., Chen, J., et al.: G-Hadoop: MapReduce across distributed data centers for data-intensive computing. Futur. Gener. Comput. Syst. **29**, 739–750 (2013)
68. Rasooli, A., Down, D.G.: Guidelines for Selecting Hadoop Schedulers Based on System Heterogeneity. J. Grid Comput. **12**, 499–519 (2014)
69. Kala Karun, A., Chitharanjan, K.: A review on hadoop—HDFS infrastructure extensions. In: 2013 IEEE Conference on Information & Communication Technologies (ICT), pp. 132–137 (2013)
70. Vaidya, M.: Parallel processing of cluster by map reduce. Int. J. Distrib. Parallel Syst. **3**, 167 (2012)
71. Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C., et al.: SHadoop: improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. J. Parallel Distrib. Comput. **74**, 2166–2179 (2014)
72. O'Driscoll, A., Daugelaite, J., Sleator, R.D.: 'Big data', Hadoop and cloud computing in genomics. J. Biomed. Inform. **46**, 774–781 (2013)
73. Vijayalakshmi, V., Akila, A., Nagadivya, S.: The survey on mapreduce. Int. J. Eng. Sci. **4**, 3335–3342 (2012)
74. Borthakur, D.: The hadoop distributed file system: architecture and design. Hadoop Project Website **11**, 21 (2007)
75. He, W., Cui, H., Lu, B., Zhao, J., Li, S., Ruan, G., et al.: Hadoop+: modeling and evaluating the heterogeneity for MapReduce applications in heterogeneous clusters. In: Proceedings of the 29th ACM on International Conference on Supercomputing, pp. 143–153 (2015)
76. He, B., Fang, W., Luo, Q., Govindaraju, N.K., Wang, T.: Mars: a MapReduce framework on graphics processors. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 260–269 (2008)
77. 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, 2007. HPCA 2007, pp. 13–24 (2007)
78. Chen, R., Chen, H., Zang, B.: Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, pp. 523–534 (2010)
79. Chen, Y., Qiao, Z., Jiang, H., Li, K.-C., Ro, W.W.: Mgmr: Multi-gpu based mapreduce. In: Grid and Pervasive Computing, pp. 433–442. Springer (2013)
80. Gu, Y., Grossman, R.L.: Sector and Sphere: the design and implementation of a high-performance data cloud. Philos. Trans. R. Soc. Lond. A: Math. Phys. Eng. Sci. **367**, 2429–2445 (2009)
81. Zhang, Y., Gao, Q., Gao, L., Wang, C.: imapreduce: a distributed computing framework for iterative computation. J. Grid Comput. **10**, 47–68 (2012)
82. Liu, Q., Todman, T., Luk, W., Constantinides, G.A.: Automated mapping of the MapReduce pattern onto parallel computing platforms. J. Signal Process. Syst. **67**, 65–78 (2012)
83. Qian, J., Miao, D., Zhang, Z., Yue, X.: Parallel attribute reduction algorithms using MapReduce. Inf. Sci. **279**, 671–690 (2014)
84. Derbeko, P., Dolev, S., Gudes, E., Sharma, S.: Security and privacy aspects in MapReduce on clouds: a survey. Comput. Sci. Rev. **20**, 1–28 (2016)
85. Xia, T.: Large-scale sms messages mining based on mapreduce. In: International Symposium on Computational Intelligence and Design, 2008. ISCID'08, pp. 7–12 (2008)
86. Jin, C., Vecchiola, C., Buyya, R.: MRPGA: an extension of MapReduce for parallelizing genetic algorithms. In:

IEEE Fourth International Conference on eScience, 2008. eScience'08, pp. 214–221 (2008)
87. Xu, B., Gao, J., Li, C.: An efficient algorithm for DNA fragment assembly in MapReduce. Biochem. Biophys. Res. Commun. **426**, 395–398 (2012)
88. Hsu, C.-Y., Yang, C.-S., Yu, L.-C., Lin, C.-F., Yao, H.-H., Chen, D.-Y., et al.: Development of a cloud-based service framework for energy conservation in a sustainable intelligent transportation system. Int. J. Prod. Econ. **164**, 454–461 (2015)
89. Zhang, F., Cao, J.: A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications. Futur. Gener. Comput. Syst. **43**, 149–160 (2015)
90. López, V., del Río, S., Benítez, J.M., Herrera, F.: Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data. Fuzzy Sets Syst. (2014)
91. Xu, X., Ji, Z., Yuan, F., Liu, X.: A novel parallel approach of cuckoo search using MapReduce. In: 2014 International Conference on Computer, Communications and Information Technology (CCIT 2014) (2014)
92. Bi, X., Zhao, X., Wang, G., Zhang, P., Wang, C.: Distributed extreme learning machine with kernels based on MapReduce. Neurocomputing **149**, 456–463 (2015)
93. del Río, S., López, V., Benítez, J.M., Herrera, F.: On the use of MapReduce for imbalanced big data using Random Forest. Inf. Sci. **285**, 112–137 (2014)
94. Kim, J., Chou, J., Rotem, D.: iPACS: power-aware covering sets for energy proportionality and performance in data parallel computing clusters. J. Parallel Distrib. Comput. **74**, 1762–1774 (2014)
95. Paniagua, C., Flores, H., Srirama, S.N.: Mobile sensor data classification for human activity recognition using MapReduce on cloud. Procedia Comput. Sci. **10**, 585–592 (2012)
96. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: WebPIE: a web-scale parallel inference engine using MapReduce. Web Semant. Sci. Serv. Agents World Wide Web **10**, 59–75 (2012)
97. Li, Z., Shen, Y., Yao, B., Guo, M.: OFScheduler: a dynamic network optimizer for MapReduce in heterogeneous cluster. Int. J. Parallel Prog. **43**, 1–17 (2013)
98. Rizvandi, N.B., Taheri, J., Moraveji, R., Zomaya, A.Y.: Network load analysis and provisioning of MapReduce applications. In: 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 161–166 (2012)
99. Maurya, M., Mahajan, S.: Performance analysis of MapReduce Programs on Hadoop cluster. In: 2012 World Congress on Information and Communication Technologies (WICT), pp. 505–510 (2012)
100. Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.: Tarazu: optimizing mapreduce on heterogeneous clusters. In: ACM SIGARCH Computer Architecture News, pp. 61–74 (2012)
101. Ahmad, F., Lee, S., Thottethodi, M., Vijaykumar, T.: Puma: purdue mapreduce benchmarks suite (2012)
102. Brandt, A.: Algebraic analysis of MapReduce samples. Bachelor Thesis, University of Koblenz-Landau (2010)
103. Verikas, A., Gelzinis, A., Bacauskiene, M.: Mining data with random forests: a survey and results of new tests. Pattern Recogn. **44**, 330–349 (2011)
104. Miner, D., Shook, A.: MapReduce design patterns: building effective algorithms and analytics for Hadoop and other systems. O'Reilly Media, Inc. (2012)
105. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud **10**, 95 (2010)
106. Xin, J., Wang, Z., Qu, L., Wang, G.: Elastic extreme learning machine for big data classification. Neurocomputing **149**, 464–471 (2015)
107. He, Q., Shang, T., Zhuang, F., Shi, Z.: Parallel extreme learning machine for regression based on MapReduce. Neurocomputing **102**, 52–58 (2013)
108. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings. 2004 IEEE International Joint Conference on Neural Networks, 2004, pp. 985–990 (2004)
109. Huang, G.-B., Chen, L., Siew, C.-K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Netw. **17**, 879–892 (2006)
110. Huang, G.-B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing **70**, 3056–3062 (2007)
111. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**, 489–501 (2006)
112. Alamir, P., Navimipour, N.J.: Trust evaluation between the users of social networks using the quality of service requirements and call log histories. Kybernetes **45**, 1505–1523 (2016)
113. Mohammad Aghdam, S., Navimipour, N.J.: Opinion leaders selection in the social networks based on trust relationships propagation. Karbala Int. J. Modern Sci. **2**, 88–97 (2016)
114. Nourozi, M., Souri, A., Navimipour, N.J.: User relationship management approach for human behavior interactions in the social networks: behavioral modeling and formal verification. Behav. Inf. Technol. (2018, in press)
115. Liu, G., Zhang, M., Yan, F.: Large-scale social network analysis based on mapreduce. In: 2010 International Conference on Computational Aspects of Social Networks (CASoN), pp. 487–490 (2010)
116. Yang, S.-J., Chen, Y.-R.: Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds. J. Netw. Comput. Appl. **57**, 61–70, 11// (2015)
117. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS'95, pp. 39–43 (1995)
118. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99 (1999)
119. Sheikholeslami, F., Navimipour, J.N.: Service allocation in the cloud environments using multi-objective particle swarm optimization algorithm based on crowding distance. Swarm Evol. Comput. **35**, 53–64 (2017)

120. McNabb, A.W., Monson, C.K., Seppi, K.D.: Parallel pso using mapreduce. In: IEEE Congress on Evolutionary Computation, 2007. CEC 2007, pp. 7–14 (2007)
121. Gandomi, A.H., Yang, X.-S., Alavi, A.H.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng. Comput. **29**, 17–35 (2013)
122. Navimipour, N.J., Milani, F.S.: Task scheduling in the cloud computing based on the cuckoo search algorithm. Int. J. Model. Optim. **5**, 44 (2015)
123. Li, H., Wei, X., Fu, Q., Luo, Y.: MapReduce delay scheduling with deadline constraint. Concurr. Comput. Pract. Exp. **26**, 766–778 (2014)
124. Asghari, S., Navimipour, J.N. Cloud services composition using an inverted ant colony optimization algorithm. Int. J. Bio-Inspired Comput. (2017, in press)
125. Asghari, S., Navimipour, J.N. Resource discovery in peer to peer networks using an inverted ant colony optimization algorithm. Peer-to-Peer Netw. Appl. (2017, in press)
126. Azad, P., Navimipour, N.J.: An energy-aware task scheduling in cloud computing using a hybrid cultural and ant colony optimization algorithm. Int. J. Cloud Appl. Comput. **7** (2017, in press)
127. Dréo, J., Siarry, P.: A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In: Ant Algorithms. Springer, pp. 216–221 (2002)
128. Wu, B., Wu, G., Yang, M.: A mapreduce based ant colony optimization approach to combinatorial optimization problems. In: 2012 Eighth International Conference on Natural Computation (ICNC), pp. 728–732 (2012)
129. Wang, H., Xu, Z., Pedrycz, W.: An overview on the roles of fuzzy set techniques in big data processing: trends, challenges and opportunities. Knowl.-Based Syst. **118**, 15–30 (2016)
130. Li, X., Song, J., Zhang, F., Ouyang, X., Khan, S.U.: MapReduce-based fast fuzzy c-means algorithm for large-scale underwater image segmentation. Futur. Gener. Comput. Syst. **65**, 90–101 (2016)
131. Cheng, S.-T., Wang, H.-C., Chen, Y.-J., Chen, C.-F.: Performance analysis using petri net based MapReduce model in heterogeneous clusters. In: Advances in Web-Based Learning–ICWL 2013 Workshops, pp. 170–179 (2013)
132. Jayasree, M.: Data mining: exploring big data using Hadoop and MapReduce (2008)
133. Mesmoudi, A., Hacid, M.-S., Toumani, F.: Benchmarking SQL on MapReduce systems using large astronomy databases. Distrib. Parallel Databases **34**, 1–32 (2015)
134. Wu, L., Yuan, L., You, J.: Survey of large-scale data management systems for big data applications. J. Comput. Sci. Technol. **30**, 163–183 (2015)
135. Müller, G., Sonehara, N., Echizen, I., Wohlgemuth, S.: Sustainable cloud computing. Bus. Inf. Syst. Eng. **3**, 129–131 (2011)
136. Milani, A.S., Navimipour, N.J.: Load balancing mechanisms and techniques in the cloud environments: systematic literature review and future trends. J. Netw. Comput. Appl. **71**, 86–89 (2016)
137. Milani, B.A., Navimipour, N.J.: A comprehensive review of the data replication techniques in the cloud environments: major trends and future directions. J. Netw. Comput. Appl. **64**, 229–238 (2016)
138. Ashouraie, M., Navimipour, N.J.: Priority-based task scheduling on heterogeneous resources in the Expert Cloud. Kybernetes **44**, 1455–1471 (2015)
139. Chiregi, M., Navimipour, N.J.: Trusted services identification in the cloud environment using the topological metrics. Karbala Int. J. Modern Sci. **2**, 203–210 (2016)
140. Sun, Y., Qi, J., Zhang, R., Chen, Y., Du, X.: MapReduce based location selection algorithm for utility maximization with capacity constraints. Computing **97**, 1–21 (2013)