The Journal of Supercomputing (2019) 75:6934–7002
https://doi.org/10.1007/s11227-019-02907-5

MapReduce: an infrastructure review and research insights

Neda Maleki[1] · **Amir Masoud Rahmani**[1] · **Mauro Conti**[2]

Published online: 8 June 2019

Abstract
In the current decade, doing the search on massive data to find "hidden" and valuable information within it is growing. This search can result in heavy processing on considerable data, leading to the development of solutions to process such huge information based on distributed and parallel processing. Among all the parallel programming models, one that gains a lot of popularity is MapReduce. The goal of this paper is to survey researches conducted on the MapReduce framework in the context of its open-source implementation, Hadoop, in order to summarize and report the wide topic area at the infrastructure level. We managed to do a systematic review based on the prevalent topics dealing with MapReduce in seven areas: (1) performance; (2) job/task scheduling; (3) load balancing; (4) resource provisioning; (5) fault tolerance in terms of availability and reliability; (6) security; and (7) energy efficiency. We run our study by doing a quantitative and qualitative evaluation of the research publications' trend which is published between January 1, 2014, and November 1, 2017. Since the MapReduce is a challenge-prone area for researchers who fall off to work and extend with, this work is a useful guideline for getting feedback and starting research.

Keywords MapReduce paradigm · Parallel and distributed programming model · Hadoop · Systematic review

1 Introduction

Over the past years, there has been a flow of data at the scale of petabytes produced by users' jobs [1]. Known as the Big data era, this makes it difficult for enterprises to maintain and extract valuable information for offering efficient and user-friendly services [2]. Due to the nature of services provided by these firms, data are available

✉ Amir Masoud Rahmani
rahmani@srbiau.ac.ir

1 Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

2 Department of Mathematics, University of Padua, Padua, Italy

in different formats such as image, log, text, and video [3]. They also have extensive information in different languages because of many users around the world. Therefore, researchers have found themselves involved in the most complex processes such as the data storage technique, instant data lookup, manipulation, and updating of the data [4].

As the data are extremely large and unstructured, and needs real-time analysis, it has raised a concept in many researchers' mentality that a new platform for data retention, transmission, storage, and processing is required [5]. The platform that is capable of processing and analyzing the large volumes of data with an acceptable velocity and reasonable cost. This necessity, from the point of data platform architecture, led to yield parallel and distributed computing on the clusters and grids. In these environments with cost-effective and high-capacity hardware, programming requires to consider data consistency and integrity, node load balancing, skew mitigation, fair resource allocation, and preemption and non-preemption of jobs. Thus, programmers constantly live in fear of these obstacles [4]. To hide the complexities from users' view of the parallel processing system and abstract the system characteristics, numerous frameworks have been released. The goal of all of these frameworks is focusing the user on his/her production programs and delegating the complexity and controls to the framework [6].

Across all frameworks, MapReduce is known as a certain programming pattern. This pattern is inspired by the functional language Lisp [4], enabling end users to express all kinds of parallel procedures with Map and Reduce functions, without considering the messy parallelism details like fault tolerance, data distribution, and load balancing. It has major importance in handling Big data matter [7].

The basic architecture of the MapReduce framework has two functions called Map and Reduce wherein the former feeds the latter's input to carry out the computing. The significance of this pattern in serially performing batch processing on Big data is clearly visible [6]. In this framework, parallel computing is commenced by distributing map tasks on different nodes and simultaneously processing disparate data partitions called split. Eventually, by aggregating map outputs and applying the reduce function, the final results are produced, thus accomplishing processing [1].

In recent years, the expansion and evolution of MapReduce especially in the context of its open-source implementation "Hadoop" has resulted in features such as energy efficiency of jobs, fault tolerance, load balancing of cluster, scheduling of jobs and tasks, security, performance, and elasticity which has generally propelled the publishing of multiple articles in journals and conferences.

Some other programming models such as Spark [8] and DataMPI [9] are competing with MapReduce. Since MapReduce is an open source with high performance which is used by many big companies for processing batch jobs [10, 11] and is our future research line, we chose to conduct the study on the MapReduce programming model. Table 1 compares the features of MapReduce, Spark, and DataMPI.

With the help of the recent articles, considered in this research study and by using inclusion and exclusion criteria, an illustration of MapReduce topics in a systematic study template is presented, thus making the research simple and explicit for readers. The only systematic literature study [4] on MapReduce, which is a holistic paper, was conducted in 2014, but since then to the present time, no other systematic

Table 1 Comparing of MapReduce, Spark, and DataMPI

Programming model	Project/year	Data processing type	Data model	Compatible with HDFS	Cluster manager	Storage system	Speed	Scalability	Fault tolerance/ execution time	Languages	Cost	Security	Applications benefit
MapReduce	Apache/2008	Batch mode	Key-value pair-based	Yes	Hadoop YARN	Hadoop distributed file system (HDFS)	Low: because of disk-based processing (multiple stages)	More than 40,000 nodes	High because data are written on disks/ longer execution time because of re-execution of all in progress or pending tasks on the failed node	Java, PigLatin, HiveQL	Number of disks (nodes)/ low cost of RAM	Kerberos authentication/ service-level authorization	Suitable for non-iterative and interactive applications
Spark	Apache/2012	Real time stream mode	RDD-based	Yes	Hadoop YARN, Apache Mesos, Cassandra, Open Stack Swift, Amazon S3, Kudu	Hadoop distributed file system, Map R file system	High: because of in-memory processing (single stages)	Up to 10,000 nodes	Relatively Low because the data are kept in memory as resilient distributed dataset (RDD) objects/ shorter execution time because the lost partition will be automatically recomputed by using the original transformations	SparkSQL, Python, Scala, Java, R	Less number of disks (nodes)/ high cost of RAM	Password authentication/as it can be integrated with HDFS, the access control lists supported	Suitable for iterative algorithm-based applications like machine learning

Table 1 (continued)

Programming model	Project/year	Data processing type	Data model	Compatible with HDFS	Cluster manager	Storage system	Speed	Scalability	Fault tolerance/execution time	Languages	Cost	Security	Applications benefit
DataMPI	DataMPI team/2014	Stream, iterative	Key-value pair-based	Yes	Yes	MVAPICH2	Highest	Same as Hadoop	Same as Hadoop	Java	Same as Hadoop	–	Suitable for Streaming and Iterative applications

and comprehensive review has been done. To the best of our knowledge, our study is the first systematic paper from 2014 to November 2017 which is comprehensive and holistic. In this paper, we have considered prominent varied topics of MapReduce which are required to be further investigated. We extracted and analyzed data from the relevant studies of MapReduce to answer the research questions (RQs) and have presented the answers as our work's contribution.

The rest of the paper is organized as follows. Section 2 consists of two parts: In part one, we introduce a brief architectural overview of MapReduce and Hadoop as its mostly regarded implementation, and in part two, we provide our research methodology. Section 3 reviews the selected papers of three phases. In Sect. 4, we answer the research questions and analyze the results to highlight hot and cold issues in the studies and discuss opportunities for future research. Finally, in Sect. 5 we present our conclusions and the limitations of our research.

2 Background and research methodology

2.1 Background

Hadoop is an open-source Apache project [12] that was inspired by Google's proprietary Google File System and MapReduce framework [13]. Hadoop distributed file system provides a fault-tolerant storage of large datasets [12–14]. Figure 1 shows the HDFS architecture. HDFS supports high-performance access to data using three-replica data block placement policy; two in-rack block replica; and one off-rack block replica [15]. It has two major components: one NameNode and Many DataNodes, in which the metadata are stored on NameNode and application data are kept on DataNodes. A dedicated server called Secondary NameNode is employed for file system image recovery in the presence of failure [14] which provides high availability of Hadoop [16]. The NameNode–DataNodes architecture makes the system

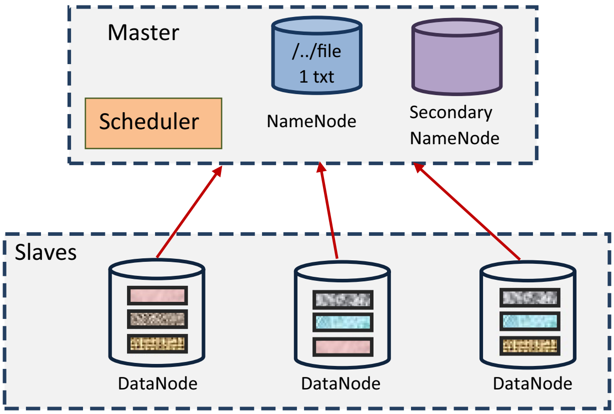


Fig. 1 HDFS architecture [16]

scalable, and all the nodes communicate through TCP protocols [13]. The scheduler for job assignment across the Hadoop cluster resides in the Master node [17].

MapReduce, the processing unit of Hadoop consists of two main components: one JobTracker and many TaskTrackers in which the JobTracker coordinates the user's job across the cluster and the TaskTrackers run the tasks and report to the JobTracker [1, 14, 18, 19]. Figure 2 shows the MapReduce job execution flow. All the input splits key-value pairs are processed in parallel using the mappers [14, 17, 18]. The mapped out files which are called intermediate data are partitioned based on the key, sorted in each partition, and then written on the local disk of the DataNodes [1, 20]. Reducers fetch remotely the data related to the similar key and produce the reduce output files which are stored on HDFS [14, 20].

Hadoop ecosystem consists of many projects which can be categorized as (1) NoSQL databases and their handler projects such as HBase, Sqoop, and Flume; (2) data collecting and processing projects such as Kafka, Spark, and Storm; (3) workflow and streaming data analysis projects such as Pig, Hive, Mahout, Spark's MLlib, and Drill; (4) administration projects like ZooKeeper and Ambari for providing and coordinating the services in the distributed environment of Hadoop cluster; and (5) security projects such as centralized role-based Sentry, non-role-based Ranger, and Knox [14, 19, 21, 22]. Furthermore, we can name some of Hadoop distributions such as MapR, Cloudera, Hortonworks DataPlatform, Pivotal DataSuite, and IBM InfoSphere and some Hadoop repositories including HealthData, National Climate Datacenter, and Amazon Web Services datasets [23].

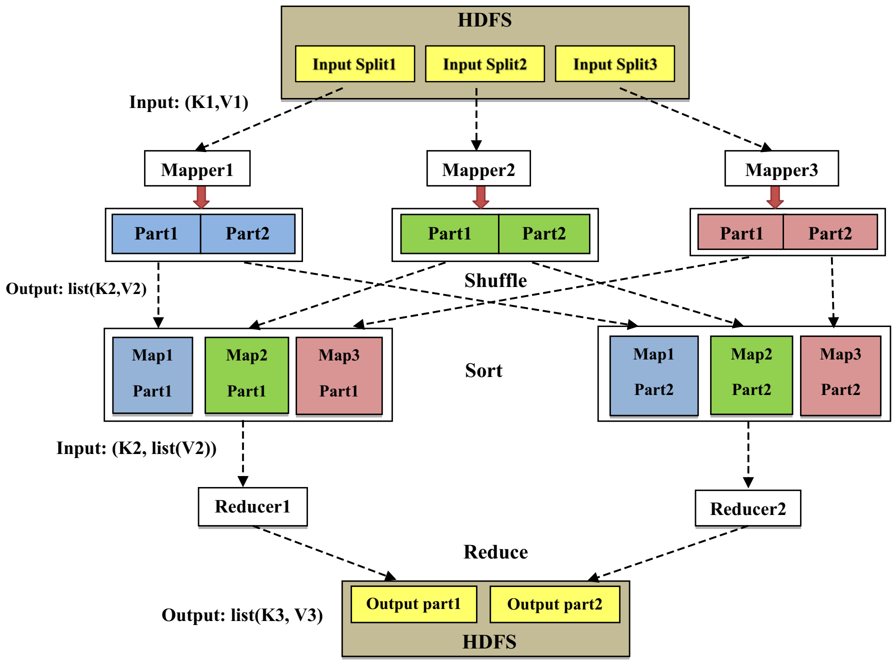


Fig. 2 MapReduce job execution flow [20]

2.2 Research methodology

According to [24–26], we classify and select the articles based on the following protocol:

- According to our research area, some research questions are defined.
- According to these research questions, keywords are found.
- Search strings are made based on these keywords, i.e., by logical and proximity search of keywords in the validated databases as a source to find the targeted papers.
- Final papers are screened based on some inclusion and exclusion criteria.

2.2.1 Research questions

Research questions are classified into two categories: quantitative and qualitative. Hence, based on this category, we bring up the research questions:

RQ1 What topics have been considered most in MapReduce field?
RQ2 What are the main parameters, investigated by the studies?
RQ3 What are the main artifacts produced by this research?
RQ4 What experimental platforms have been used by the researchers for analysis and evaluation?
RQ5 What kind of benchmarks and dataset have been used in the experiments?
RQ6 What are the open challenges and future directions in Hadoop MapReduce?

2.2.2 Paper selection process

We use the following libraries as sources to direct the search process:

- IEEE Xplore (http://www.ieee.org/web/publications/xplore/).
- ScienceDirect—Elsevier (http://www.elsevier.com).
- SpringerLink (http://www.springerlink.com).
- ACM Library (http://dl.acm.org/).

We organize the researches in three phases. In each phase, we define search terms for finding systematic mapping and literature studies, regular surveys, and primary studies, respectively.

Phase 1. Finding Systematic Studies

We applied "*" to represent zero or more alphanumeric characters for the word "study" for finding its variants like "study" and "studies" and parentheses used if the word "systematic" is not in title but in abstract or keywords.

Phase 2. Finding Survey Studies

We first applied search string "Title: MapReduce AND (Title: survey OR Title: review)." However, since we wanted to exclude "systematic review" from our results, we used the "NOT" operator in the search string.

Table 2 Search strings

Phase 1	S1	((Systematic) OR title: mapping OR title: literature) AND (Title: stud* OR title: review) AND (title: "MapReduce")
	S2	((Systematic) OR title: mapping OR title: literature) AND (Title: stud* OR title: review) AND (title: "Map-Reduce")
Phase 2	S3	"Title: MapReduce AND (title: survey OR Title: review) NOT (systematic)"
Phase 3	S4	"MapReduce AND Hadoop"

Table 3 Inclusion and exclusion criteria for study selection

Inclusion criteria
Studies are published from January 2014 to November 2017
Studies focused on Hadoop MapReduce and its various aspects
Studies addressed challenges in MapReduce
The paper which has resolved a challenge in MapReduce and not a challenge has been resolved by MapReduce
Approach and its validation have been logically presented
Exclusion criteria
Studies are published in languages other than English
Studies are not indexed in the ISI
Interdisciplinary journals have been excluded
Studies that do not answer or are irrelevant to the research questions
Ph.D. theses, academic blogs, editorial notes, technical reports, open access journals have been excluded

Phase 3. Finding Primary Studies

Since the mostly regarded implementation of MapReduce is Hadoop, in order to have a holistic search, our search strings were made from the terms like "MapReduce" and "Hadoop." As the results were too many, we refined the hit list by using an advanced search option, in title, abstract, and author keywords [27]. The three-phase search strings are shown in Table 2.

To assure that only the qualified publications are included from January 2014 up to November 2017, we applied the following inclusion criteria (Table 3) to select the final papers:

Using this strategy, we found 66 papers for conducting the study in which five studies [4, 26, 28–30] have conducted the systematic review, six studies [2, 31–35] have done a survey in Hadoop MapReduce, and the rest which will be reviewed in Sect. 3.2 are the primary studies in MapReduce field. Figure 3 shows the adopted process of article selection in the study.

Figure 4 shows the number of articles per year from January 2014 to November 2017. It is observed that the publication of papers in the field of Hadoop MapReduce infrastructure level has an increasing trend. Figure 5 shows the number of shares of each publisher to the publications.

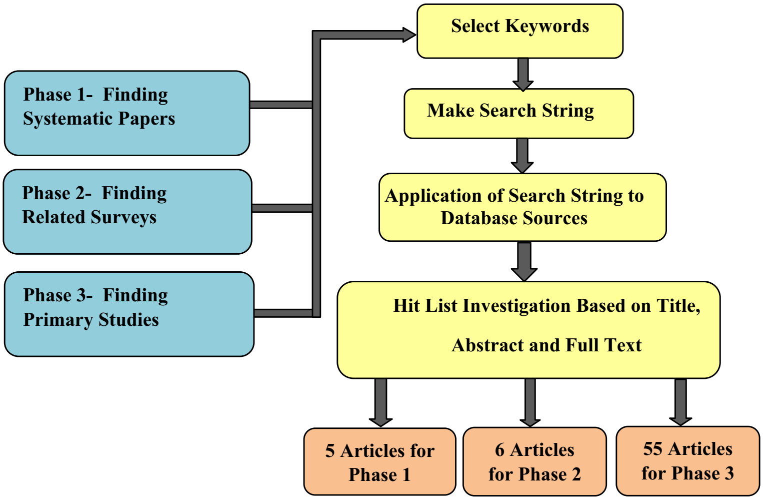


Fig. 3 Schematic map of article selection process

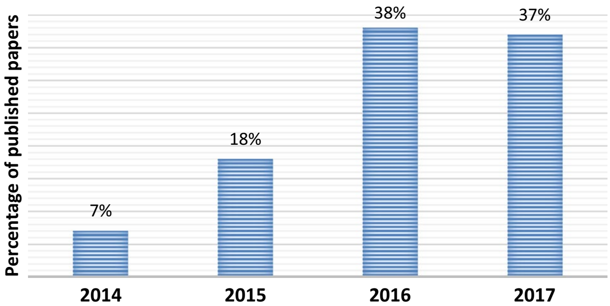
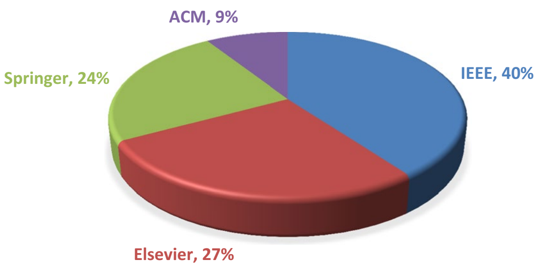


Fig. 4 Annual distribution of publications, from January 2014 to November 2017

Fig. 5 Percentage of published papers based on various publishers



2.2.3 Studies classification

According to the researches, we could reach a good prospect of the main existing challenges in the MapReduce framework. We classified the studies in seven categories according to their main research focus. Figure 6 shows the taxonomy.

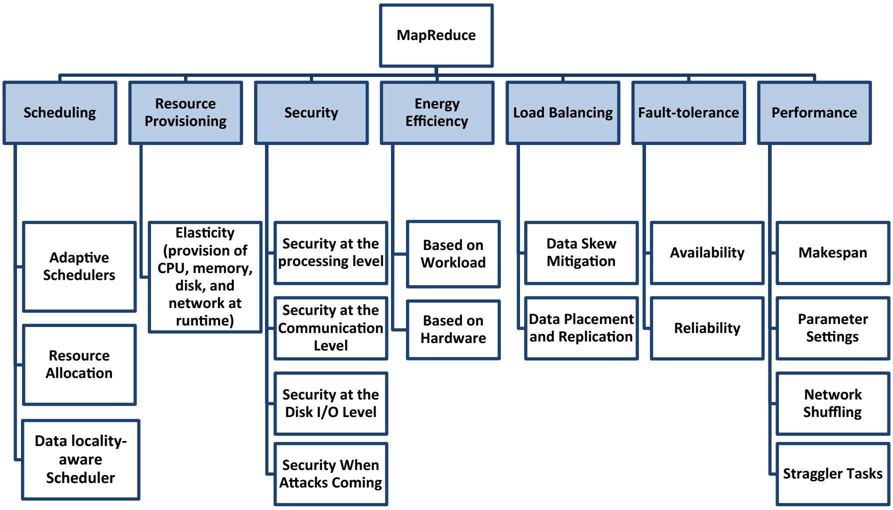


Fig. 6 Taxonomy of the MapReduce studies

We explain each topic category concisely from right to left direction in taxonomy.

- The studies in which the system efficiency is the main concern, indicators such as makespan (jobs completion time), network traffic in transferring data from map tasks to reduce tasks during shuffle phase and number of disks I/O, tuning system parameters, and dealing with stragglers (slow tasks) in the cluster are highly essential for consideration.
- Reliability and availability parameters are important when the studies intend to consider the fault tolerance of a MapReduce cluster. Since the master node is a single point of failure, how to design the fault-tolerant mechanisms to keep the master node available is the main concern. When a data node fails, keeping access to the requested data by tasks is another concern in the fault tolerance topic. Furthermore, considering solutions when map and reduce tasks fail during the processing is another key point.
- When some reduce tasks have more input data which cause an unbalanced load across the cluster, the data skew parameter should be considered. Moreover, considering efficient data access as the main focus, where to place data across the cluster and the replication number of each data block are the major parameters.
- Mitigating energy consumption as the major objective, the cluster characteristics and application type should be noted. How to launch a task near its data (data locality) to improve job execution time and subsequently the energy consumption is an important concern in the energy efficiency studies.
- The studies which are focused on security, data security when it is transferring (data in motion) or stored (data in rest), secure map and reduce tasks execution, and secure data flow in the presence of threats and attacks are the critical concerns of MapReduce.

- The cases in which the high workload causes a high demand for resources, researches provide solutions such as provisioning the resources in run-time, i.e., they consider elasticity parameter.
- The studies in which scheduling is the main topic, solutions like adaptive schedulers, efficient resource allocation, and data locality-aware schedulers play vital roles. The adaptive scheduler as the first solution could be employed to schedule user jobs with various SLAs using job run-time information to improve performance metrics including job execution time, makespan, CPU utilization, etc. Resource allocation as the second solution is used to allocate resources to user jobs efficiently. Data locality-aware scheduler is another effective solution to optimize one or a set of performance metrics including data locality, energy consumption, makespan, and so on.

3 Review of studies

In this section, we review primary studies and regular surveys separately.

3.1 Regular surveys

In [31], the authors divided the existing deficiencies in MapReduce into three categories in terms of improvement goals: (1) the native variants which are the studies done by Google as the creator of MapReduce; (2) the Apache variants studies focused on Hadoop; and (3) the third-party extensions wherein most of them have investigated the Hadoop platform improvements such as the I/O access in Hadoop platform, enhancement of database operations in Hadoop, and scheduling scheme of Hadoop map and reduce tasks. This survey also compares the parallel DBMSs with MapReduce in terms of scalability and efficiency. The authors also mention the reason for different parallel processing technologies specifically MapReduce attracting attention. Furthermore, they reviewed some hybrid systems which integrate traditional RDBMS alongside MapReduce. However, there is no comparison between these studies' pitfalls and advantages.

Derbeko et al. [32] have studied the security and privacy aspects of the MapReduce framework in a cloud environment. On the one hand, there is a close relationship of the cloud and MapReduce such that the deployment of the MapReduce in public clouds enables users to process large-scale data in a cost-effective manner and provide the ease of processing and management. But, on the other hand, this deployment causes security and privacy challenges since it does not guarantee the rigorous security and privacy of computations as well as stored data. The authors also investigated the security-related projects in the context of MapReduce such as authentication of users, users' authorization, auditing–confidentiality–integrity–availability (ACIA) of both the data computation pair and verification of outputs. Additionally, they considered the privacy aspects besides security such as the ability of each participating party to prevent adversarial parties from observing data, codes, computations, and outputs. However, the authors did not address some security issues such as authorization frameworks and trust domains of MapReduce requiring different MapReduce algorithms for data encryption and privacy policies.

Hashem et al. [2] have reviewed the application of MapReduce, as a promising technology, in various domains such as telecommunication, manufacturing, pharmaceutical, and governmental organizations. It also considers the algorithms and solutions for improvement and reduction in its challenges between the years 2006 and 2015. This paper has conducted a basic bibliometric study using keywords, abstracts, titles, affiliations, citations, countries, and authorship. Moreover, this paper has investigated the most influential articles of Scopus platform in the MapReduce improvement domain such as declarative interfaces, data access, data processing, data transfer, iteration, resource allocation, and communication in MapReduce as well as their pros and cons.

Li et al. [33] have studied the basic concept of the MapReduce framework, its limitations, and the proposed optimization methods. These optimization methods are classified into several topics such as job scheduling optimization, improvement in MapReduce programming model, real-time computation support for stream data, speeding up the hardware of the system, performance tuning like configuration parameters, energy saving as a major cost, and its security through stronger authentication and authorization mechanisms. Moreover, some open-source implementation frameworks of MapReduce are presented in Table 4. Although this is a comprehensive study, it still needs more research on the mentioned aspects.

Iyer et al. [34] considered the data-intensive processing and its various approaches along with their advantages and disadvantages, MapReduce programming model, and the application of MapReduce in diverse fields. Some platforms which compete with Hadoop for processing large data are as follows: (1) sector and Sphere in terms of processing speed in TeraSort benchmark; (2) DryadLINQ which is a sequential programming model combined with LINQ expressions making the programming easy; (3) integration of Kepler and Hadoop for workflow applications which provide an easy-to-use architecture and impressive performance. The investigated studies have been compared in terms of scalability, efficiency, file system type, and cost. The number of comparison criteria is adequate; however, the number of considered papers is not enough.

Liu et al. [35] have investigated the fault tolerance aspect of MapReduce. In the distributed commodity Hadoop, due to the failure probability in each of the levels of the system such as node level, rack level, and cluster level it causes the emerging of the slow tasks (also known as straggler task), the speculative execution of these tasks is urgent. Hadoop supports this method by the execution of the copy of the slow task on another node which will process the task faster and make the Hadoop throughput better. Additionally, some other speculative methods such as LATE, MCP, Ex-MCP, and ERUL in a heterogeneous environment of Hadoop have been considered.

3.2 Primary studies

In the following sections, we thoroughly consider and analyze individually each topic, presented in Fig. 6. Our observations are summarized in a table in each subsection. The studies are compared in terms of main idea, advantages, weakness, investigated parameters, their tool and method, benchmarks, dataset and jobs

Table 4 Open-source implementation of MapReduce [33]

Framework	Development language	Programming language	Deployment Environment	Operating System
QT Concurrent	C++	C++	Shared memory system	Linux, Windows, Mac OS
Phoenix/Phoenix++	Java, C++	Java, C++	Shared memory system	Linux
Disco	Erlang	Python	Master–Slave clusters	Linux, Mac OS
GridGain	Java	Java	Master–Slave clusters	Linux, Windows, Mac OS
Skynet	Ruby	Ruby	Peer-to-peer clusters	Linux
Twister	Java	Java	Shared memory system	Linux
Misco	Python	Python	Master–Slave mobile systems	Linux
Hadoop	Java	Java, C++	Master–Slave clusters	Linux
Apache Pig	Java	Pig Latin	Master–Slave clusters	Linux
Cascading	Java	Java	Master–Slave clusters	Linux
Scalding	Scala	Scala	Master–Slave clusters	Linux

(workload), and the experimental platform to find whether the study contribution has been implemented, simulated, or both.

3.2.1 Energy efficiency studies

Mashayekhy et al. [36] have proposed a framework to improve the energy efficiency of deadline-assigned MapReduce jobs. The authors have modeled the performance of individual tasks of a job as an Integer Program. To solve the problem, they have provided two heuristic scheduling algorithms which quickly find the near-optimal solutions. Therefore, the schedulers are also suitable for real-time systems. The model was designed to fulfill the service-level agreement in terms of meeting the deadline of jobs. Since there are multiple jobs with different functionalities in a Hadoop cluster, how to model an efficient and distributed scheduler to solve the energy problem has not been considered by the authors.

Ibrahim et al. [37] have investigated the impact of dynamic voltage and frequency scaling (DVFS) on the performance and energy consumption of the Hadoop cluster and trade-off between performance and power. There are several modes to mitigate power usage using DVFS and Turbo including power save, conservative, and on-demand modes. However, these governors result in sub-optimal solutions even in different phases of Hadoop and do not reflect their design goal. Furthermore, the jobs consume different power in these modes and have not the same execution time which impacts the performance of the entire cluster. Therefore, the authors have provided the insights for efficiently deploying and executing MapReduce jobs by determining the job type including CPU-intensive, I/O-intensive, and network-intensive, and then dynamically tuning the suitable governor according to CPU load.

Song et al. [38] have proposed a modulo-based mapping function in which the data blocks are mapped to the nodes of a cluster in order to mitigate the data shuffling and saving energy. The insight behind of such mapping is that by fairly distributing the data blocks across a heterogeneous cluster and by considering the data characteristics, each task can locally access its data and all the tasks can be completed simultaneously. To achieve this goal, the authors considered three factors: “fairness of size,” “fairness of range,” and “best adaptability.” However, the proposed algorithm is deprived of considering the replacement strategy of the blocks when a node failure happens or employing a data replication method in the presence of node failure.

Cai et al. [39] have proposed a network traffic-aware and DVFS-enabled resource allocation scheduler. Based on job profiling, the scheduler allocates the resources to deadline-assigned jobs while considering rack-level data locality. Furthermore, the authors improve energy efficiency based on the slack time in which the CPU frequency is adjusted for upcoming tasks. By considering worst-case completion time, the proposed solution achieved a better SLA than stock Hadoop. However, the study has not considered the heterogeneity of the system. The authors need to employ a modified version of job profiling technique in which the job execution is measured either on a small portion of input dataset or using an online estimation of job execution time when running on servers with different speeds.

Teng et al. [40] have proposed co-optimized energy-aware solutions including (1) Tight Recipe Packing (TRP) is employed to consolidate the reserved virtual Hadoop clusters into the physical servers to save energy and (2) online time-balancing (OTB) is used for on-demand virtual machines placement to mitigate the mode switching through balancing server performance and utilization. The study only considered the off-line and online batch jobs, while a general platform should be able to handle running various workloads with different SLAs to enhance the energy efficiency of a Hadoop-based cloud datacenter. Besides, the proposed power model should consider as well other system resources such as memory and I/O power to reach better performance.

Phan et al. [41] have provided two energy-aware speculative execution techniques while considering system performance. First, a hierarchical slow jobs detection technique is employed for reducing the number of killed speculative copies. Then, the hierarchical method eliminates the non-critical straggles to reduce the energy waste on unsuccessful speculative copies. Second, based on a performance–energy model, an energy-efficient speculative copy allocation mechanism is used to allocate the speculative copies. The hierarchical solution can dramatically reduce energy wasted on removed speculative copies while maintaining a good performance compared to the most recent straggler's detection mechanisms. However, rather than eliminating non-critical slow jobs, a reserved resource-based allocation approach can be applied to reach better performance.

Arjona et al. [42] have provided a comprehensive empirical analysis of the power and energy consumption in the heterogeneous Hadoop cluster. The authors measured the power consumed by the server resources such as CPU, network I/O, and storage under different configurations to find the optimal operational levels. They found that the system is not energy proportional and all the server resources efficiency can be maximized if the number of active CPU cores, their frequency, and I/O block size are tuned based on the system and network load. Moreover, the authors defined that the jobs energy consumption depends on CPU load, storage, and network activity. However, the only one considered application is not representative to justify the accuracy of the energy model. In addition, the RAM energy consumption and the dynamicity of CPU load have not been considered.

Table 5 shows an overview of the studies in energy efficiency topic.

3.2.2 Fault tolerance studies

In Hadoop, the minimal unit of scheduling is "task." Therefore, when a task fails, the whole task should be re-executed from scratch which results in poor performance. Wang et al. [20], have presented a finer-grained fault tolerance strategy in which the map tasks generate checkpoints per spill instead of a map output file. Therefore, a retrying task can start from the last spill point and saves a lot of time. The proposed fault tolerance strategy which comes with little overhead is not static, i.e., it allows the failed task resumes its execution from a checkpoint at an arbitrary point on demand. Some parameters such as task id, task attempt id, input range, host location, and size are used to implement this strategy.

Table 5 An overview of existing primary studies focusing on "energy efficiency"

Nos.	References	Experimental platform	Parameters	Job/dataset/workload	Main idea	Advantages	Disadvantages
1	Mashayekhy et al. [36]	Simulation Hadoop cluster (4 nodes)	Energy consumption Makespan SLA Number of the map and reduce tasks	HiBench benchmark (TeraSort, PageRank, K-means)	EMRSA: energy-aware scheduling of MapReduce jobs for Big data applications	Finding near-optimal job scheduler in terms of energy Very fast and practical Load balancing of all machines. Significant energy saving	No pipelining between map and reduce phases
2	Ibrahim et al. [37]	Grid'5000 test bed (40 nodes)	Job type Power Job response time CPU usage	PUMA benchmark (WordCount, K-means), Pi Grep, Sort	Governing energy consumption in Hadoop through CPU frequency scaling	Improving power consumption Reduction in job response time	
3	Song et al. [38]	Simulation (100 nodes) Hadoop cluster (6 nodes)	Energy consumption Scalability Fault tolerance Disk I/O CPU workload Network I/O Job execution time	MRBench (Sort, WordCount, Grep)	DPA: an energy optimization algorithm based on fair data placement.	Energy optimization No data loading delay No additional cost Improvement in job execution time	Not considering the techniques in the presence of the node failure Not adaptable
4	Cai et al. [39]	Simulation using CloudSim (30 PMs)	Energy consumption Data locality SLA Cluster utilization	Job: sort, matrix, multiplication	SLA-aware energy-efficient scheduling scheme for Hadoop	Energy improvement Low resource cost Low network traffic	Not considering the heterogeneity of the environment

Table 5 (continued)

Nos.	References	Experimental platform	Parameters	Job/dataset/workload	Main idea	Advantages	Disadvantages
5	Teng et al. [40]	Simulation Hadoop cluster	Energy consumption Jobs SLA	Job: Sort, Terasort, WordCount	The energy efficiency of VM consolidation in MapReduce style IaaS clouds	Save energy Performance improvement A co-optimization solution Guarantee SLA DVFS-enabled servers	Only considering MRV1 Simple power model
6	Phan et al. [41]	Grid'5000 test bed (21 nodes)	Energy consumption Number of concurrent map tasks Throughput Number of speculative copies	PUMA benchmark (WordCount, K-means, Sort)	Energy-driven straggler mitigation in MapReduce	Higher energy saving due to the lower number of killed speculative copies Higher throughput	Low-performance gain Not considering the DVFS technique
7	Arjona et al. [42]	Hadoop cluster (nemesis, survivor, erdos servers)	Energy consumption Block size Throughput Number of CPU cores CPU frequency	Job: PageRank	A measurement-based analysis of the energy consumption of data center servers	Lower power consumption Higher throughput Lower disk I/O power usage Lower network I/O power usage Maximum efficiency The highly accurate energy model	Not considering the Ram power usage Not considering the CPU load dynamicity

Fu et al. [43] have conducted their work in three parts: (1) considering the issues of Hadoop speculation mechanism; (2) classifying the faults and failures in a cluster in two groups: (a) hardware failure, i.e., a node failure and (b) software failure i.e., a task failure, and simulating the hardware failure condition for small and large jobs; and (3) manipulating and adjusting the Hadoop failure timeout and testing the different scenarios. The authors have implemented their strategy in three phases: (1) they have used a central information collector which detects faults and failure in run-time; (2) in spite of the Hadoop speculator, the authors' speculator knows the corresponding nodes of each task. Therefore, when a failed node is detected, all the affected tasks are speculated in an exponential order; and (3) they used a dynamic threshold to determine whether a failure should be speculated or not. If the node has been unavailable for a time interval longer than the threshold, the tasks on that node are speculated.

Tang et al. [44] have investigated the node availability and network distance to overcome the node failure and low-speed network bandwidth in a large cluster. This work which is called ANRDF is a part of the authors' previous work, entitled "BitDew-MapReduce." BitDew-MapReduce contains nine components as follows: (1) replicator; (2) fault-tolerant mechanism; (3) data lifetime; (4) data locality; (5) distributor; (6) BitDew core service; (7) heartbeat collector; (8) data message; and (9) ANRDF. They have predicted each node availability in a cluster using the feature-weighted naïve Bayes which is more accurate than the naïve Bayes. In addition, for estimating the network distance, a bin-based strategy has been employed such that any node in a cluster which is called "application node" measures its distance from the "Landmark nodes" and partitions itself into a bin in which the nodes have the minimum latency from each other.

Encountering omission failures which are caused by straggler tasks, there are two aspects: (1) copying the slow task and (2) duplicating the resources. Memishi et al. [45] have presented a failure detection and solving aspect through service timeout adjustment. The authors have employed three levels of the strictness of failure detection using three different algorithms so that the deadline-assigned jobs have more accurate failure detector mechanism. The lenient level of detection is suitable for small workloads whose completion time is less than the default timeout of Hadoop. This level adjusts the timeout by estimating the workload completion time. The two other detectors outperform the default timeout of Hadoop under any workload type and failure injection time and they adjust the timeout dynamically based on the progress score of the user workload.

The reliability of Hadoop is entrusted to its core and is fulfilled by re-executing the tasks on a failed node or by input data replication. Yildiz et al. [46] have presented a smart failure-aware scheduler which can act immediately when a failure recovery is needed. To mitigate the job execution time, the scheduler uses the preemption technique rather than waiting approach in which the tasks should wait an uncertain time until the resources are freed. For obtaining the required resources, one way is to kill the primitive running tasks on the other nodes and allocate their resources to the tasks on the failed machine. This method will waste both the resources on which the tasks were running and all the computations which are already done by these tasks. Therefore, the proposed scheduler benefits from a work-conserving task preemption technique with only a little overhead. The map task preemption is done by "splitting approach" through a preemption signal. For example, upon receiving the signal by a map task, the task is split into two sub-tasks in which the first one consists of all the processed key-value pairs up to preemption and it is reported to the JobTracker as a completed task, while the second one which consists the unprocessed key-value pairs is added to a pool to be executed later when there is available slot. The reduce task preemption is done by "pause and resume approach" in which the reduce task is paused upon receiving a preemption signal and its data are stored on the local node for being restored back upon resume. To choose a task for preemption, the tasks of the low-priority jobs are selected. Priority is based on the data locality. Namely, the scheduler selects the tasks to be preempted that belong to nodes where the input data of the failed tasks reside.

Lin et al. [47] proposed a method to satisfy the Hadoop reliability through intermediate data replication. The authors have measured two parameters: (1) the probability metric in which a job can be completed by the cluster and (2) the energy consumed by the cluster is measured to finish the job under two different intermediate data replication policies which are employed in Hadoop. The first policy is the Hadoop default policy in which the map outputs are stored in their host nodes and is called locally stored (LS). The second policy is imitated the reduce task in which the reduce outputs are replicated in the HDFS and is called a distributed file system (DFS). The authors have conducted the experiments by considering two scales of jobs, i.e., small and large jobs under two levels of parallelism including: (1) full parallelization of a job, i.e., all the tasks of a job can be executed in parallel and (2) full serialization of a job, i.e., none of the tasks of a job can be executed in parallel. Therefore, the authors have considered four scenarios: (1) LS/small jobs; (2) LS/large jobs; (3) DFS/small jobs; and (4) DFS/large jobs that can help Hadoop administrators to choose the best replication configuration for a cluster setting.

Table 6 shows an overview of the fault tolerance in MapReduce cluster-related papers.

3.2.3 Job/task scheduling studies

Xu et al. [48] have provided a dynamic scheduler in which each TaskTracker can automatically adjust its number of tasks based on both its processing capacity and workload changes. The scheduler hinders the overloaded and under-loaded nodes using dynamic slots-to-tasks allocation strategy in preference to static slot allocation of Hadoop. The dynamic strategy is based on that in each heartbeat, the full capacity of a TaskTracker would not be at the disposal of the tasks and the TaskTracker makes the decision to accept either more tasks or not by considering its workload. Two monitoring and task executing modules are used for detecting TaskTracker load condition and for executing the accepted tasks, respectively. To achieve the desired results, the monitoring module considers the CPU load, i.e., the number of tasks in the queue of the CPU which are ready to run, CPU utilization, and memory as the load parameters.

Lim et al. [49] have formulated the matchmaking and scheduling problem for an open stream of multistage deadline-assigned jobs using constraint programming.

Table 6 An overview of existing primary studies focusing on "fault tolerance"

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
1	Wang et al. [20]	Hadoop cluster (16 nodes) on Azure Cloud	Overhead of method Scalability Input data size Map buffer size Map and reduce tasks execution time CPU, disk, and network usage of reduce tasks Performance of merging in reduce side Impact of map side's combiner Reliability	HiBench benchmark (WordCount, Hive query)	BETL: creating checkpoints by allowing map tasks to output their intermediate data in multiple files on multiple nodes	A finer-grained level fault-tolerant mechanism The faster execution time of map tasks due to not merging spill files The benefit of speculative execution Less JVM overhead No master overhead More resilient in terms of creating checkpoints on demand	Reduce task has to shuffle and sort more files Network traffic More disk access No pipelining between map and reduce phases
2	Fu et al. [43]	Hadoop cluster (two private clusters with 21 nodes)	Job size Job execution time Network delay Network stability Failure occurrence time	HiBench benchmark (Aggregation, Join, K-means, PageRank, Scan, Sort, TeraSort), Built-in YARN benchmark (WordCount, TeraSort, SecondarySort)	FARMS: proposing a hybrid solution which includes a speculation mechanism scheme and a task scheduling policy to enhance failure awareness and recovery	Dramatic performance improvement in failure Adaptable Adaptive and available centralized fault analyzer	Not considering the failure in reduce phase

Table 6 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
3	Tang et al. [44]	Simulation using TDEMR (1 PM, 1015 simulated data nodes) Hadoop Cluster with 36 nodes (16 PMs on Hadoop cluster, 20 VMs on Cloud)	Job response time Prediction error Node availability	Dataset: ETI@home traces, BRITE generator Job: WordCount, Grep	ANRDF: Proposing Resource availability and network distance-aware MapReduce over the Internet using weighted naïve Bayes classifier-based availability prediction and landmark-based network estimation	Decreasing job response time High estimation accuracy of nodes' availability Low shuffle transfer	Trusting to nodes which have the potential of failing Not considering the heterogeneity of the environment Lack of new network estimator
4	Memishi et al. [45]	Simulation (25 containers)	Task completion time Workload size Reliability	Job: Sort	Solving the omission failures using timeout service adjustment	More accurate failure detection Performance improvement	Lack of enhancement of behavior of framework in terms of failure detection and its relation with the Hadoop timeout

Table 6 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
5	Yildiz et al. [46]	Grid'5000 test bed Parapluie cluster on Rennes (Hadoop cluster with 9 nodes)	Data locality Task completion time Job completion time Reliability Scalability Type of job Failure detection timeout Workload size Network traffic Job execution time	Dataset: Facebook workload Job: Map-heavy (WordCount), Reduce-heavy (Sort), PUMA benchmark	Chronos: proposing a failure-aware scheduling strategy in which the recovery tasks with a higher priority would preempt the tasks with less priority	Data locality improvement Reduction in job completion time Failure improvement in reduce phase due to the reduction in waiting time Throughput improvement by reducing the waiting time of launching recovery tasks Reduction in network traffic Effectiveness of reduce task preemption technique	Overhead of profiling and preemption technique
6	Lin et al. [47]	Hadoop cluster	Input data size Number of reduce tasks Number of jobs Network traffic	–	Improving job completion reliability and job energy consumption using replicating intermediate data	Improving job completion reliability Reducing job energy consumption	Not considering the heterogeneity of the environment Not considering the network failure

Each job's SLA is characterized by the earliest start time, the execution time, and the end-to-end deadline such that the jobs which miss their deadline are minimum. MRCP-RM is only applicable to jobs with two phases of execution such as MapReduce jobs. The objective of MRCP-RM is to minimize the number of jobs that miss their deadlines.

Kao et al. [15] have investigated the trade-off between data locality and performance for deadline-assigned real-time jobs in a homogeneous server system. Three modules are employed in each node to provide deadline guarantees: (1) dispatcher; (2) power controller; and (3) scheduler. To meet the deadline of the jobs, the authors consider the map task deadline of a job which is called "local deadline." For this purpose, two separate queues for each map and reduce tasks are considered in each data node. Then, the dispatcher first assigns a local deadline to map tasks of a job, and according to this local deadline, the task with the shortest deadline is executed first. Using a partition value estimation, the proposed method partitions tasks to data locality-aware nodes for less data transmission and less blocking. Furthermore, to mitigate energy consumption, some nodes are switched to the sleep state. In this work, because of the considerable penalty of data migration, the proposed framework does not consider the precedence of tasks to satisfy the data locality. Therefore, the shorter jobs are blocked by the non-preemptive execution of larger jobs which mitigates the Hadoop performance.

Sun et al. [50] have provided a data locality-aware scheduler in which the expected data of future launching map tasks are prefetched earlier in memory on the intended nodes. The intended nodes are determined based on current pending tasks which their remaining time is less than a threshold and greater than the data block transmission time. According to the consumer–producer model and to manage effectively the memory buffer, two prefetching buffer units each with the same size as the HDFS block are considered per each map slot. Therefore, by using the prefetching technique, the map tasks with rack locality and rack-off locality would not be delayed and consequently, jobs will be completed rather.

Tang et al. [51] have presented an optimized and self-adaptive scheduler to reduce tasks. The scheduler can decide dynamically and according to the job content, including the completion time of the task and the size of the map output. In fact, this method prevents the wasting of reduce slot during the copy phase by delaying the start time of the reduce task of the current job and provides idle reduce slots for other jobs. Thus, at a certain time, when some tasks of the job have completed, the scheduler schedules and assigns the reduce slots to the reduce tasks of that job. This method mitigates the completion time of the reduce task, decreases the average system response time, and utilizes the resources efficiently.

Bok et al. [52] have considered data locality and I/O load for deadline-assigned jobs which process multimedia and images. Plus, it minimizes job deadline, miss, using two queues, called "urgent" and "delay" queues. The paper minimizes the deadline miss ratio which is caused by I/O load using "urgent" queue and maximizes deadline hit ratio using hot block repetition. Delay queue has the same functionality as Delay scheduling [53] job queue in which the task whose data are located on the other nodes should currently be executed, but its data do not exist on the host node. Therefore, it will wait for a short time (D) expecting at that time a slot on the other nodes is freed and can be executed on them. If in the waiting time, there will be any slots, then after finishing the waiting time the task will be executed on its host node and data locality will not meet. Urgent queue allocates slots to the jobs which are expected to not complete in their deadline because of no data locality or high node workload. When the client submits the job, it first is placed in the delay queue. If the difference of deadline and the predicted completion time is higher than a threshold which is specified by the user, the job is sent to the urgent queue. In the urgent queue, the jobs are arranged to ascend according to the difference amount to execute.

Hashem et al. [54] have proposed a two-objective scheduler for minimizing the cost of cloud services and job completion time. The model is a two-objective model in which the cost of resources from the point of view of resource allocation and the job completion time from the point of view of scheduling is considered as the main objectives. Therefore, the proposed model improves performance when processing Big data using the MapReduce framework. The model applies the earliest finish time algorithm in which both tasks to resources and resources to tasks mapping are done to meet the model objectives. In the algorithm, the earliest finish time is chosen based on the number of tasks of a job which is configured by the job owner. In addition, the service method will return a positive value if there are adequate mappers and reducers to finish a workflow job in the specified budget and deadline.

Nita et al. [55] have presented a multi-objective scheduler which considers both deadline and budget constraints from the user side. To find a best matching between deadline-assigned jobs and available slots, the authors define a service function and a decision variable. The service function returns a positive value if there are enough mappers and reducers to complete a MapReduce job within budget and deadline and the decision variable represents the weight of resource usage. The best assignment between jobs and resources is selected based on the summation of each service result. In addition to the costs for a map and reduce processing time and their resource usage costs, a penalty for the transferred data have been considered due to its non-locality.

Tang et al. [56] have presented a scheduling algorithm for the jobs in the format of a workflow. Since the execution time of the jobs is different due to the job types, i.e., I/O-intensive or CPU-intensive, the algorithm comprises a job prioritizing phase in which the jobs are prioritized with respect to their types, input data size, communication time to other jobs, and type of slots. Moreover, a task assignment phase has been considered to prioritize the tasks for scheduling based on the data locality on their intended node. Therefore, the scheduling length and tasks workflow parallelization have been improved. Table 7 shows an overview of the MapReduce job/task scheduling-related papers.

3.2.4 Load balancing studies

Since the imbalance of keys in a Hadoop cluster is intrinsic, Chen et al. [57] have presented a user-transparent partitioning method to solve data skew in the reducer side. To evenly distribute map output data between the reduce tasks, this paper benefits an integrated sampling in which a small fraction of the produced data

Table 7 An overview of existing primary studies focusing on "job/task scheduling"

Nos.	References	Experimental Platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
1	Xu et al. [48]	Hadoop cluster (7 PMs and 12 VMs using VMWare)	Task response time The execution time of all tasks Load balancing Scalability CPU and memory utilization Number of tasks	CPU-intensive job (TeraSort)	ATSDWA: each task tracker weighs its load in each heartbeat based on the collected load parameters and adjusts the maximum number of slots for tasks dynamically	Load balancing of cluster Simple, reliable, efficient, and applicable algorithm Reduction in tasks' execution time Enhancement of system response ability The same quality of service as Hadoop Higher resource utilization No overloading bottleneck on JobTracker (self-regulatory of TaskTrackers according to workload)	Not scalable Lower system's adaptability when resource utilization is too high

Table 7 (continued)

Nos.	References	Experimental Platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
2	Lim et al. [49]	Simulation Hadoop cluster (11 nodes on Amazon EC2)	Quality of service Data locality Scalability Task execution time Job arrival rate System workload parameters Number of resources Job deadline	Gutenberg project (Synthetic Facebook and WordCount workload)	MRCP-RM: modeling the resource allocation and resource scheduling problem using constraint programming	The low proportion of jobs miss their deadline Lower average job turnaround time Efficiently processing on the open stream of MapReduce jobs with SLAs Flexible in terms of minimization the number of late jobs and small overhead of scheduling Scalable Non-preemption algorithm so, not wasting computation time Implementing a simulator called "SimExec" Reduction in Map and Reduce task execution time	Not effective in the lightly load system Not considering the priority of jobs for modeling resource management No technique to improve performance in error presence caused by user-estimated execution time Not considering the complex workloads

Table 7 (continued)

Nos.	References	Experimental Platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
3	Kao et al. [15]	Simulation using CloudSimRT Cloud cluster (20 PMS, 40 VMs)	Data locality Energy Job response time Meet deadline ratio	Synthetic workload (I/O-bound, CPU-bound) Hive benchmark Facebook workload Yahoo! workload Job: Batch, Select, TextSearch, Aggregation	DamRT: proposing a data locality-aware real-time scheduler for interactive MapReduce jobs	Run-time power saving Improving the quality of service Higher data availability Minimizing response time of tasks	Non-elastic
4	Sun et al. [50]	Hadoop cluster (21 nodes)	Data locality Scalability Job execution time Input data size Block size	PUMA benchmark (Grep, Histogram rating, Classification, WordCount, Inverted-index)	HPSO: proposing an inter-block and inter-block prefetching-based task scheduler to improve data locality	High data locality High scalability Reduction in job execution time	Overhead of memory size due to prefetching buffer Ineffective prediction method Network transmission cost
5	Tang et al. [51]	Hadoop cluster (7 nodes)	System average response time Task execution time Job completion time Slot usage Input data size Number of map tasks	MRBench (WordCount, Pi, TeraSort, GridMix)	SARS: proposing an optimal and self-adaptive reduce scheduling policy.	Reduction in shuffle and sort phases Reduction in task execution time Saving reduce slots Reduction in job completion time	Network I/O bottleneck Not considering the heterogeneity of the environment

Table 7 (continued)

Nos.	References	Experimental Platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
6	Bok et al. [52]	Hadoop cluster (20 nodes)	Data locality I/O load Number of deadline-assigned and total jobs Deadline jobs miss ratio Job completion time Makespan	Dataset: Multimedia data generator Job: I/O-light (WordCount), I/O-heavy (TeraSort)	Minimizing deadline miss ratio of jobs which process large multimedia data using urgent queue scheduling	Reduction in job completion time Reducing deadline miss ratio of jobs using speculation method The increasing deadline success ratio of jobs using hot data block replication Improving makespan	
7	Hashem et al. [54]	Hadoop cluster (10 VMs)	Job completion time Cost Throughput CPU utilization Makespan Input data size	Dataset: Hadoop scheduler load simulator Job: CPU-intensive (WordCount), I/O-intensive (Sort)	Optimizing job scheduling based on the multi-objective mode	Minimizing job completion time High resource utilization rate Reducing the processing time of the workload Low latency	

Table 7 (continued)

Nos.	References	Experimental Platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
8	Nita et al. [55]	MobiWay test bed (12 nodes)	Number of cores Makespan Elasticity Job completion time Memory size Time of node updating	Dataset: scheduling load simulator (SLS million song)	MOMTH: proposing a multi-objective scheduler to fulfills the constraints such as deadline and budget	Development and deployment of cost–time continuing SLS simulator Improvement in job completion time Reduction in task waiting time in the tasks' queue	Not considering the energy consumption Scheduling decision is taken only based on the current knowledge The biggest job cannot be executed due to deadline miss The higher cost of time due to computing dynamically the number of map and reduce slots Not considering the complex job
9	Tang et al. [56]	Hadoop cluster	Type of job Number of jobs Task execution time Number of tasks Input data size	Job: data-intensive (Montage workflow)	MRWS: proposing a two-phase optimized workflow scheduler based on job types and data locality	Parallelism and efficiency speedup under any graph size Practical Makespan improvement Reduction in task execution time	

during the processing of twenty percentage of map tasks is sampled. Afterward, the large keys are split by considering the servers capacity. Reducers can shuffle immediately the data which are already produced. Hence, the job execution time will be dramatically decreased.

Repartitioning intermediate data to preserve load balancing of a heterogeneous Hadoop cluster incurs high overhead. To tackle this problem, Liu et al. [58] have presented a run-time partitioning skew mitigation. The idea is that, rather than controlling data size by its splitting among reducers, resources are allocated dynamically. Namely, the number of allocated resources to the reducers could be increased and decreased in run-time. A resource allocator module is responsible for allocating the number of resources which is demanded by a reduce task. The required resources are allocated based on a statistical model which is constructed by the current partition size and the allocated resources of a reduce task which are enough to estimate the Reduce task execution time. This method is simple and incurs no overhead.

Chen et al. [59] have considered the data placement problem in terms of data locality and remote data access costs for both map and reduce sides. The authors have presented a replication strategy in the map side, in which the data access cost defined as a function of the distance of nodes and the data size to be transferred is minimized. The same way, to mitigate the data access frequency in the reduce side, the block dependencies are detected and the blocks which have strong dependency are merged as a single split for processing. Furthermore, to alleviate network traffic, the authors have defined an optimal matrix to place all data blocks based on a topology-aware replica distribution tree. Thus, the data movement during the map and reduce stages is minimized.

Li et al. [60] have proposed a programming model, called Map-Balance-Reduce, with an extra middle stage to effectively process the unevenly distributed data which is caused by unbalanced keys. In this model, the map outputs are estimated in the Balance stage and the balanced output of this stage is fed into the Reduce stage. This stage is like a mini-reducer stage in which the task that will cause load unbalancing problems is found in advance by preprocessing the map outputs. The stage sums the map outputs of the same key, partitions them to more splits, and feeds them into the reducer nodes. Importantly, how to define that whether the load is unbalanced is based on the workload of reduce task nodes. If the workload of a reduce task node is less than a certain threshold, the adaptive load balancing process is applied in which the current reduce task is stopped and the keys on current reduce nodes will be partitioned and distributed to other n-1 nodes. In this way, the algorithm will mitigate job execution time.

Liroz-Gistau et al. [61] have presented a system which has an extra phase between map and reduces phases, denoted "intermediate reduce." The skewed map outputs take benefit of the reducers of this phase and can be processed in parallel. The intermediate reduce function is like a combiner and can be iterated adaptively based on the partition size, I/O ratio, or partition skew ratio until a given threshold. Once the map buffer verges, their spills are maintained in a table. Hereafter, they are merged as a partition, and based on the greedy or the data locality strategy, they are fed into the intermediate reducers as input splits. Exploiting the spills once they are ready makes the system fault-tolerant and faster, while it incurs master overhead in terms of keeping the spills metadata.

Myung et al. [62] have presented histogram information on a join key attribute method to balance a load of reducers to join jobs. In this paper, the data skew problem is relieved by mapping splits to reducers using a partitioning matrix. By a small number of input samplings, the samples from all relations make the matrix and the join operations are done based on the key range overlapping. Namely, the join candidate cells will provide better performance in the join operations. In the range-based partitioning, the most repeated samples are defined as the cause of imbalanced partitioning, i.e., the most skewed relations. Despite the range-based partitioning, the matrix considers the less skewed relations of join. Furthermore, the proposed partitioning outperforms the random-based partitioning in which the rate of input duplication increases substantially with an increase in the input size.

Liu et al. [63] have presented an architecture in which the workload distribution of reduce tasks is predicted by using an online partition size prediction algorithm. Therefore, in addition to map function and the number of reducers, the partition sizes are dependent on the input dataset. The algorithm uses a small set of random data to profile some characteristics of the whole input data. Based on the predicted workload, the framework can detect the tasks with a large workload using a deviation detection method without any knowledge of statistic distribution of the data in linear time. Before allocating the resources to the overloaded tasks, the framework determines the relation between task duration with two factors, i.e., partition size and resource allocation. Thereupon, the framework speeds up the job completion time by adjusting proactively the allocation of resources to the overloaded tasks.

Zhang et al. [64] have presented a two-objective data skew mitigation model. The model is executed in two independent phases which are called data transmission and data computation. In the data computation phase, the minimum number of nodes which participate in data processing is calculated. In data transmission phase, based on the satisfying upper bound of relative data computation time in computation phase, both the data transmission time and network usage are minimized using a greedy algorithm to find the best network flow. Besides, the method allows users with higher priority to configure their jobs to be processed earlier.

Table 8 shows an overview of the load balancing in MapReduce cluster-related papers.

3.2.5 Performance studies

There are two constraints in MapReduce: (1) executing the Reduce tasks before map tasks for assuring the logic correctness of MapReduce and (2) running the Map tasks in map slots and reduce tasks in reduce slots. Because of the mentioned constraints, Tang et al. [65] have proposed a job ordering algorithm for optimizing the two performance factors including makespan and total completion time. For finding the best ordering of jobs, the authors have defined an optimal slot configuration where jobs are ordered based on this configuration. Furthermore, the authors have considered a condition in which based on every job order, an optimal slot configuration will be found. Since there is a trade-off between makespan and total completion

Table 8 An overview of existing primary studies focusing on "load balancing"

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
1	Chen et al. [57]	Hadoop cluster (30 PMs, 15 VMs using KVM, OpenStack OS)	Job execution time Input data to reduce tasks Coefficient of variation in data size	Dataset: Wikipedia Synthetic workload Job: CPU-intensive (join), I/O-intensive (Sort, Grep, Inverted-index)	LIBRA: solving the data skew problem in reduce side.	Load balancing among reduce tasks Applicable and transparent Job execution speedup Overlap between map and reduce phases Highly accurate approximation for distribution intermediate data No need for a prerun sampling of input data	Occupying more task slots Not able to split and rearrange large keys Not able to improve the shuffle phase by reducing skew mitigation
2	Liu et al. [58]	SAVI test bed (Hadoop cluster with 21 VMs using Xen, OpenStack OS)	Generated partitions' size (intermediate data) Reduce task execution time Job completion time CPU and memory allocation Makespan	Dataset: Wikipedia, Graph generator, RandomText writer, Netflix workload, Synthetic workload PUMA benchmark (text retrieval, web search, machine learning, and database jobs)	DREAMS: providing a run-time partitioning skew mitigation by controlling the number of resources allocated to each reduce task	No repartitioning overhead Simple to implement Improving job completion time Reduction in Reduce task execution time Effective mitigation of the negative impact of partitioning skew Accuracy of partition size prediction	No generality because of job profiling stage overhead No precise estimation of task execution in a highly dynamic environment No network and disk fairness sharing are investigated (only CPU and memory are considered) Not applicable for computational skew applications

Table 8 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
3	Chen et al. [59]	Simulation using TopoSim (1080 simulated data nodes) hadoop cluster (18 data nodes)	Data locality Network traffic Makespan Block size Input data size Replication factor Network scalability	Job: K-means, Word-Count, TeraSort	Proposing an optimal data placement technique by a topology-aware heuristic algorithm	Minimizing global data access costs Maximizing data locality Least computation costs in all kinds of block sizes Least computation and communication costs in all kinds of input data sizes, replication factor, and network sizes Implementing a simulator called "TopoSim"	It is not open source Resource costs of data placement into HDFS is higher than Hadoop-stock
4	Li et al. [60]	Hadoop cluster (5 nodes)	Data Skew Input data size Job execution time	Dataset: NCDC's weather	MBR: proposing the preprocessing scheduling and self-adaption scheduling of Map-Balance-Reduce programming model for effectively processing data with unbalanced keys	Reduction in reduce phase Load balancing in reduce tasks Improvement in job execution time	Time overhead in evenly distributing data

Table 8 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
5	Liroz-Gistau et al. [61]	Grid'5000 platform (20 nodes)	Data Skew Input data size Number of intermediate keys Intermediate data size Block size Job execution time Data locality	Dataset: Wikipedia, Synthetic workload Job: Top k %, SecondarySort, Inverted-index, PageRank, WordCount	Skew handling of reduce side by introducing a new phase called Intermediate Reduce	Reduction in reduce phase Reduction in job execution time Higher fault tolerance	Overhead of computation due to intermediate reducers
6	Myun et al. [62]	Hadoop cluster (13 nodes)	Data skew to reduce tasks Input data size Speedup Scalability Number of samples Number of cores Number of splits Sample size	Dataset: Cloud ships and land stations report, Synthetic workload (Scalar skew, Zipf's skew)	MDRP: proposing a skew handling method by constructing multi-dimensional range partitioning	Compatible and fast Applicable to complex join operations Load balancing of reduce tasks	Additional cost due to creating a histogram Lack of efficient maintenance of a histogram Lack of selection of a join key attribute
7	Liu et al. [63]	SAVI test bed (Hadoop cluster with 11 VMs)	Size of partition Heap size Number of reduce tasks Makespan of reduce tasks Reduce task execution time Job completion time	Dataset: Wikipedia, Synthetic workload Job: Sort, Inverted-index, WordCount, the relative frequency	OPTIMA: proposing an online partitioning skew mitigation technique for MapReduce by prediction the workload distribution of reduce tasks at run-time	Reduction in Reduce task execution time Improvement in job completion time Prediction accuracy of partition size in linear time Reduction in makespan of reduce tasks Applicable Load balancing	

Table 8 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
8	Zhang et al. [64]	Simulation	Input data size Data Skew Resource usage Scalability	–	DTPM: minimizing data transmission time and network bandwidth usage using distributed two-phase mode	Reduction in the number of nodes which participate in data computation phase Less bandwidth usage Reduction in data skew Scalable Reduction in shuffle time	Not considering the energy efficiency Not considering the job priority

time, a greedy job ordering algorithm based on a heuristic slot configuration algorithm have been proposed. Although in the second version of Hadoop, YARN is introduced which benefits "container" model, as there is no controlling of a number of reduce which can run in a container, the network bandwidth will be a bottleneck due to the reduce tasks shuffling.

Verma et al. [66] have presented a two-stage scheduler based on the Johnson algorithm to minimize the makespan of multi-wave batch jobs. According to Johnson algorithm, the jobs are arranged based on the map execution time in an ascending queue. If the reduce execution time is shorter than map execution, the scheduler puts the job at the tail of the queue. Although this method mitigates the makespan, in some scenarios that the number of tasks of a job is less than the available slots, local optimal would cause a problem. To tackle the problem, a heuristic method called "Balanced Pool" is employed in which the jobs are divided into two pools with approximately the same makespan. The paper has not considered a model for the jobs whose data are ready during the other jobs' execution time because the order of the algorithm is almost high due to repetitive divisions.

Since manually configuration of Hadoop is a tedious and time-consuming task, Bei et al. [67] have presented a random forest methodology in which the parameter settings are tuned automatically. In this method, two regression-based models are constructed to accurately predict the performance of each map and reduce stage. Subsequently, in each stage a genetic algorithm is employed which is fed by the aforementioned models outputs and the configuration space is found. The proposed method is suitable and fast for repetitive and long-running applications with large input data in a Hadoop cluster.

Although the Hadoop performance is enhanced using task scheduling or load balancing techniques, the heterogeneity of a cluster deteriorates the performance of the running jobs which are configured homogeneously. Cheng et al. [68] have proposed an ant-based architecture which is model-independent and automatically obtains the optimal configuration for the large job sets with multi-wave tasks. Namely, improvement in task tuning is performed during job execution by starting from random parameter settings and with any job profiling. The proposed architecture consists of two modules: (1) self-tuning optimizer and (2) task analyzer which resides in JobTracker. The first round of tasks is configured randomly by the optimizer module and the tasks are conducted to TaskTrackers to be executed. Once the first wave is finished, for the next round of tasks execution, the task analyzer suggests better settings to the optimizer module using a fitness function which uses task completion time.

Yu et al. [69] have presented an accelerator framework that benefits plug-in components to expedite data movement and merge data without any repetitive disk access. The key idea of the method is to levitate the data on the remote disk nodes until records merge time. The merge time is the time that all the map tasks are finished, i.e., all the map out files are produced and the construction of priority queues from segments (partitions) of map tasks is possible. This mechanism provides a full pipeline between Hadoop map, shuffle, and reduce phases and is more scalable than Hadoop-stock. Moreover, InfiniBand is used as communication hardware rather than Ethernet which is very fast.

In the shuffling phase, all of the data partitions are transmitted from the map side to the reduce side to be aggregated to feed into their related reducers. This yet challenging problem imposes high network traffic and makes the network bandwidth a bottleneck. Guo et al. [70] have proposed in-network aggregation in which the map outputs are collected and routed across the network and processed at the intermediate nodes once the transmission phase is started. To attain the idea, the authors use a tree model and a graph model to minimize each in-cast transmission, i.e., data transmission of all maps to one reducer and shuffling, i.e., data transmissions of all maps to all reducers, respectively. The methodology relieves the reduce side's aggregation load by parallelizing the reducing and shuffling phases and diminishing the job completion time.

Guo et al. [71] have presented the shuffle phase of Hadoop as an independent service from the map and reduce phase, called "iShuffle." The service acquires the intermediate data proactively, i.e., before starting to Reduce task through a "Shuffle-On-Write" operation and make it ready for the reduce task. In the Shuffle-On-Write operation, after the map buffer on a node disk is verged and its data are written on the disk, the dedicated shuffler of the node gets a copy of the data. Afterward, the shuffler places data partitions to nodes where the reduce tasks will be launched according to a placement algorithm. Therefore, using the placement algorithm which is based on the partition size prediction and solving it by linear regression, the even data distribution on the reduce nodes during data transferring is guaranteed. To gain fault tolerance, the data are not sent to the intended node directly, but it is written on the node disk first. In addition, the method uses preemptive scheduling to lessen jobs completion time. The proposed method in [72] is inspired by this paper; however, the placement mechanism is totally different. The type of jobs, CPU-intensive or data-intensive, also has been considered to balance the node workload.

Ke et al. [73] have presented a three-layer model that alleviates network traffic by designing a data partitioning schema. The proposed model defines a graph which has three layers: (1) mapper nodes; (2) intermediate nodes including aggregation nodes and Shadow nodes and (3) reduce nodes. The model is basically based on the default Hadoop placement technique. According to the intermediate data size, if the produced map output size related to a key partition is large, it is processed on the reduce tasks which are closed to the map task node and it is not sent to the reduce tasks which are placed on the other racks. In the second layer, the nodes are potential if it is supposed that the data to be moved to a reducer will be active. Otherwise, they are sent directly to the reducer through shadow nodes which practically do not exist. Therefore, by considering data locality levels, i.e., node locality, rack locality, and cluster locality, this method achieves data locality, while it mitigates the network traffic. The network traffic minimization is done by a distributed algorithm which is solved by a linear programming using Lagrange.

Chen et al. [74] have presented a speculative strategy performed in four steps: (1) detecting the stragglers; (2) predicting the original task remaining time; (3) selecting the stragglers to backup; and (4) placement of the backup tasks on the suitable nodes. First, to detect the straggler tasks, the authors use the task progress rate and the processing bandwidth in each Hadoop phase. Second, to predict process speed and task remaining time, an exponentially weighted moving average method is used. Third, to determine which task to be backed up based on a load of a cluster, a cost–benefit model has been proposed. Finally, to determine suitable nodes to host the backup tasks, data locality and data skew have been considered. The proposed strategy mitigates the job completion time and improves cluster throughput.

Load imbalance, i.e., data skew causes the emerging of straggler tasks. To overcome this problem, Guo et al. [75] have proposed a user-transparent speculative strategy for Hadoop in a cloud environment. When the stragglers are detected, the slots of the cloud are scaled out such that the stragglers benefit more resources to process their input data in less time. The proposed strategy balances the resource usage across the cluster using an adaptive slot number and slot memory size changer method in an online manner. Therefore, both the data skew and job completion time are mitigated in this strategy.

There are two main strategies for speculative execution: (1) cloning and (2) straggler detection-based. In the cloning, if the cost of computing of task is low and there are enough resources, additional replicas of the task are scheduled in parallel with the initial task. In the straggler detection-based, the progress of each task is controlled, and the additional versions are started when a straggler is detected. Xu et al. [76] have divided the cluster into lightly loaded and highly loaded. They have introduced the smart cloning algorithm (SCA) for the lightly loaded cluster and the enhanced speculative execution (ESE) algorithm for the heavily loaded cluster based on the straggler detection approach.

Jiang et al. [77] have presented a heuristic method for online jobs which enter into the system as time goes and an approximate method for off-line jobs to minimize the jobs makespan. Authors' contribution is to employ servers with different speeds. Moreover, the non-parallelizable reduce tasks assumption is another contribution which makes it more difficult to solve the makespan minimization problem. In this method, the reduce tasks are considered once preemptive and once non-preemptive. The main idea is based on the bin packing problem in which the reduce tasks are arranged according to their execution time descending and allocated to servers with higher speed, respectively. Next, the time duration that the reduce tasks will take longer on these servers is calculated and the results are arranged. Using the results, the time in which all of the servers are idle is defined and the related map task to the largest reduce task is scheduled for execution. Therefore, all the map tasks are allocated in the reduce task execution intervals. Once the total idle slots have been occupied in the interval, the rest of the map tasks are allocated after that time. Ultimately, according to MapReduce execution logic which map tasks should be executed prior to reduce tasks, the current scheduler is reversed and in case of available slots, allocation of reduce tasks continues.

Veiga et al. [78] have presented an event-driven architecture in which the phases of map tasks and reduce tasks are executed using the java threads which are called "operations." Rather than a container-based resource allocation in Hadoop, the proposed model integrates the map and reduces resources into a pool and allows the operations to benefit the resources when they need. The operations form the stages of a pipeline and are connected using data structures to reading and write the data. To alleviate the memory copies in each stage, the architecture uses the reference to the data rather than the data itself. Therefore, in this way, there is no need for converting the data to the writable objects. Furthermore, for executing the operations which must be done before the other operations, i.e., the map operation which should be executed before the merge, the system considers a priority method. The architecture is compatible with Hadoop MapReduce jobs, and any changes are required to the source code of the jobs.

According to Hadoop-LATE [79], system load, data locality, and low priority of the tasks are the major factors which should be considered as the performance model metrics. To precisely estimate the remaining execution time of the tasks, Huang et al. [80] have proposed a speculative strategy which is based on the linear relationship between system load and execution time of tasks. To detect the slow nodes a dynamic average threshold is defined and for efficient resource usage, an extended maximum cost performance model is proposed. Unlike [73], different slot values are considered. The strategies mitigate the running time and response time of the job.

Tian et al. [81] have presented a framework based on the Johnson model to minimize the makespan of off-line and online jobs. This paper has improved the paper [66], and the idea is that rather than dividing cluster resources into pools, only one pool, i.e., the cluster is enough, and all jobs can benefit all the available resources. In this way, better makespan would be acquired. In addition, this paper has proved that obtaining minimum makespan can be solved in linear time and it is not an NP problem. The authors have also mentioned that although the makespan of each pool is minimum, and the makespan of all jobs is not minimum.

Wang et al. [82] have presented a speculative execution strategy in which rather than starting the slow tasks from scratch, they start from the leveraged checkpoint of original tasks. The idea is like the checkpoints for the fault-tolerant mechanism which contributes to the granularity of fault tolerance in the spill level rather than the task level. The remaining execution time in each speculative strategy should be well estimated to select rightly the speculative tasks. Therefore, this method benefits two checkpoint types, i.e., input checkpoint an output checkpoint. The speculative task fetches its data from output checkpoint and constructs its memory states and skips the already data processed in the input checkpoint. The authors have also proposed a scheduler to select a speculative task. They have calculated the original task remaining time using the progress score, the progress rate, and the time the task has already taken. For calculating the speculative task completion time, the recovery time of partial map output and the execution time of unprocessed data are used. Based on the two calculated times and by comparing their sum to the remaining time of the original task, the "speculation gain" is calculated. The tasks with the higher gain are selected to be scheduled on the cluster.

Table 9 shows an overview of the MapReduce performance-related papers.

3.2.6 Security studies

Fu et al. [83] have investigated data leakage attacks in two platform layers, i.e., application and operating system layers. They have proposed a framework which is composed of an on-demand data collector and a data analyzer. The data collector collects Hadoop logs, FS-image files, and monitors logs from every node actively or on demand. The collected data are sent to the data analyzer in which the data are analyzed with automatic methods to find the stolen data, find the attacker, and reconstruct the crime scenario. Moreover, the authors have presented a four-dimensional algorithm with Abnormal Directory, Abnormal User, Abnormal Operation, and Block Proportion dimensions for detecting the suspicious data leakage behaviors.

Parmar et al. [84] have identified Hadoop security vulnerabilities and introduced "Kuber" to remove the vulnerabilities. The proposed framework uses three levels of security: (1) secure user authentication; (2) encrypted data in transit; and (3) encrypted data at rest. In the proposed framework, the HDFS encryption zone security mechanism is totally removed and tasks can directly access data by employing encryption on each individual data block. This technique eliminates the requirement of decryption of the complete file. Moreover, the authors benefit Salsa20 and its variant chacha20 rather than AES as a cipher suit because of their speed, safety, and easy implementation. However, the authors have not tested their framework in a distributed environment to consider the performance and scalability of the framework.

Gupta et al. [85] have presented a multilayer access control framework covering Hadoop ecosystem services, data, applications, and system resources to restrict unauthorized users. The authors enhanced the authorization capabilities of Hadoop by employing Apache Ranger and Knox frameworks in services such as HDFS, Hive, and HBase. Moreover, they enforced YARN security policies using a Ranger plug-in to prohibit unauthorized users from submitting jobs into the cluster. However, the authors have not investigated the fine-grained authorization between Hadoop core daemons including NameNode, DataNodes, and ApplicationMaster.

Wang et al. [86] have developed a compromised Hadoop cluster in which an attack is launched and a protective block-based scheme is proposed to deal with that. The authors infected a node of the cluster that delays the job execution. The toxic node cannot be detected to be decommissioned from the cluster. Therefore, the defense scheme monitors the nodes and it blocks the node in which there is any job with more killed tasks, more several slow containers, or more running tasks slower than the average task execution time. Such blocked nodes are recognized as the attacker nodes. This study only focused on the map tasks attack; however, researchers can also consider the reduce tasks attacks scenarios to better simulate toxic real systems.

There are many encrypted communications in Hadoop which leads to sensitive information leakage by means of communication patterns detection. Therefore, Ohrimenko et al. [87] have presented a framework in which secure implementation of jobs is considered and the data traffic between the map and reduce stages are analyzed. They implemented Melbourne shuffle, a secure framework to deals with information leakage which is caused by adversaries at system and application levels by means of interfering or observing of jobs execution.

Ulusoy et al. [88] introduced a fine-grained framework called, GAURDMR which enforces security mechanisms at the key-value level. The proposed framework generates dynamic authorized views of data sources using object constraint language (OCL) specifications. Moreover, it guarantees security at the computation level using a modular reference monitor and provides built-in access control model.

Table 9 An overview of existing primary studies focusing on "performance"

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
1	Tang et al. [65]	Simulation Hadoop cluster (20 nodes) on Amazon EC2	Makespan Total completion time (TCT) The average execution time of map and reduce tasks	Dataset: Synthetic workload, Facebook workload Job: PUMA benchmark (WordCount, Sort, Grep, etc.)	Proposing two classes of algorithms for optimizing job ordering and map/reduce slot configuration	Reduction in makespan and total completion time Accurately designed estimator called "MR Estimator'"	Not considering the data locality, fault tolerance, and straggler tasks Not considering the complex workload with priority Not considering the heterogeneity of the environment
2	Verma et al. [66]	Simulation Hadoop cluster (66 nodes)	Makespan Resource (number of map and reduce slots) utilization	Dataset: Wikipedia article traffic logs, Complex workload (Facebook and Yahoo!) Job: WikiTrends (Select, TextSearch, Aggregation, Join)	BalancedPools: proposing an optimal two-stage map and reduce job scheduler to minimize the makespan based on "Johnson" algorithm	Makespan improvement	Not considering the fairness Not considering the dependent jobs (DAGs) No modeling for measuring jobs' makespan which their input data will be ready during the execution time of other jobs
3	Bei et al. [67]	Hadoop cluster (10 PMs, 10 VMs)	Map execution time Reduce execution time	Job: HiBench benchmark (WordCount, TeraSort, Sort), PUMA benchmark (Adjust, Inverted-index)	RFHOC: proposing a random forest approach that constructs two groups of performance models for both map and reduce stages	The robust and accurate prediction model High scalability Speedup at map phase Lower cost at reduce phase	Large buffer size for sort phase

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
4	Cheng et al. [68]	Hadoop cluster (9 nodes) Virtualized Hadoop cluster on multi-tenant private cloud	Job completion time Task completion time I/O rate CPU steal time The memory size of sorting Job size Workload type	Dataset: Wikipedia Job: PUMA benchmark (WordCount, TeraSort, Grep)	Proposing an ant-based self-adaptive task tuning approach that automatically searches the optimal configurations for individual tasks running on different nodes	Flexible and adaptable Improvement in average job completion time Not a unified and static parameter tuning method, but an online one The effective method especially in a heterogeneous environment High speed of finding a good configuration	Not considering the implementation in a public cloud environment Not suitable for CPU-intensive jobs and small jobs Not performing well in the virtualized cluster

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
5	Yu et al. [69]	Hadoop cluster (26 nodes with InfiniBand software stack of OFED)	Shuffle-merge-reduce delay Disk throughput CPU utilization Transparency JVM overhead Network traffic Memory scalability Memory write	Job: WordCount, TeraSort	Hadoop-A: proposing an acceleration framework which uses plug-in components to fast data movement and to merge data without repetition and disk access	Full pipelining between shuffle, merge and reduce phases More scalable than Hadoop-stock Fast remote disk access through InfiniBand interconnection Elimination of repetitive merge and disk access Fast completion of map tasks due to lightweight fetching and setting up operations Improvement in throughput	Cost of new hardware, i.e., InfiniBand Building overhead of the priority queues Delay in completion of reduce tasks due to waiting for the completion of the last map output file

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
6	Guo et al. [70]	Simulation Hadoop cluster (6 PMs, 61 VMs)	Datacenter size Shuffle transfer size Aggregation ratio Network traffic Number of active links Number of cache servers Intermediate data size	Dataset: Ten input files of 65 MB per each map Job: WordCount	iShuffle: decoupling shuffle from reduce tasks and proactively pushing intermediate data to nodes via a novel shuffle-on-write operation	Skew is tackled by flexible reduce tasks dispatching Load balancing Pipelining between shuffle and reduce phases Significant Reduction in shuffle delay for shuffle-heavy jobs Reduction in job completion time Significant reduction in recovery time of a reduce task Transparent	The speculative method is not enabled Less improvement for shuffle-light jobs Overhead in map phase due to the independent shuffler Unfairness in reduce task scheduling of large jobs

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
7	Guo et al. [71]	Hadoop cluster (32 nodes)	Shuffle delay Job completion time Map phase overhead Load balancing Fault tolerance Disk throughput Locality Fairness Task type Intermediate data size	Facebook workload generated by SWIM Job: PUMA benchmark, HiBench benchmark (PageRank, Bayes), Shuffle-heavy job (SelfJoin, TeraSort, K-means, Inverted-index, Term-vector, WordCount, PageRank, Bayes), Shuffle-light job (Histogram movies, Histogram ratings, Grep)	Reducing network traffic for a MapReduce job by designing a novel intermediate data partition scheme, called "three-layer model"	Reduction in network traffic caused by map tasks Handling MapReduce job in an online manner when some system parameters are not given	No data locality in reduce side
8	Ke et al. [73]	Simulation (5 simulated VMs) Hadoop cluster (20 nodes)	Network traffic Data reduction ratio Size of the time interval Number of the map and reduce tasks Number of aggregators Number of nodes Number of keys	Dataset: Wikimedia Job: WordCount	SRS and IRS-based shuffling: pushing the aggregation computation into the network and parallelizing the shuffle and reduce phases	Reduction in network traffic Usage of less number of resources such as aggregating servers and active links Adaptable to other server-centric method structures Delay reduction during the reduce phase Data locality	

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
9	Chen et al. [74]	Hadoop cluster (small: 30 VMs and 15 PMs, large: 101 VMs and 30 PMs)	Heterogeneity of environment Scalability Job execution time Cluster throughput Data locality Data Skew	Job: Data-intensive and CPU-intensive (WordCount, Sort, Grep, GridMix)	MCP: developing the maximum cost performance strategy which has three phases: finding slow tasks, predicting their remaining time, and selecting the one to backup based on a load of a cluster, to improve the effectiveness of speculative execution	Scalable Small overhead Handling data skew well Stable performance under various kinds of environment	Not considering the straggler tasks Unable to reduce the I/O wait time More straggler tasks due to unawareness of imbalance in resource allocation of VMs
10	Guo et al. [75]	Hadoop cluster (8 PMs and 32 VMs using VMware VSphare)	Load balancing Number of slots Size of the memory slot	Dataset: TeraGen, Wikipedia, Netflix Job: HiBench benchmark (PageRank, Bayes), PUMA benchmark (WordCount, TeraSort, etc.)	FlexSlot: proposing a user-transparent task slot management scheme that automatically identifies map stragglers and resizes their slots accordingly to accelerate task execution	Flexible changing of the number of slots in an online manner Efficient resource utilization Mitigation of data skew Reduction in job completion time Simple implementation	Incurring overhead in slot resizing

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
11	Xu et al. [76]	Simulation (one PM and 11,000 simulated VMs)	Average job flow time Overall computation cost	Dataset generator	SCA: proposing a cloning-based scheme which is based on maximizing the overall system utility for a lightly loaded cluster ESE: a detection-based scheme to mitigate the number of stragglers for a heavily loaded cluster	Reduction in job delay time Reduction in e Reduction in total job flow time	More resource consumption Not considering the complex job
12	Jiang et al. [77]	Simulation (50 nodes)	Makespan Server speed Preemptive or non-preemptive task	Synthetic workload	Minimizing makespan of off-line and online jobs using heuristic and approximation methods	Minimizing makespan Applying different server speeds Load balancing	

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
13	Veiga et al. [78]	Hadoop cluster (DAS-4) Public Cloud (Amazon EC2)	Block Size Replication factor Data buffer size Data pool size Worker heap size	Dataset: RandomTextWriter, BigDataBench suite (Kronecker graph generator) Job: Micro benchmark (Sort, Grep), Application benchmark (PageRank, Connected component)	Flame-MR: proposing an event-driven architecture for improving Hadoop performance by avoiding memory copies and data movement pipelining	Scalable Portable Compatible with MapReduce jobs Flexible in terms of same software interface as Hadoop Pipelining between map and reduce phases Reduction in memory and disk usage Reduction in job execution time Minimization overhead of thread creation/destruction Alleviating memory copy operations	Low fault tolerance Unnecessary disk access
14	Huang et al. [80]	Hadoop cluster (7 PMs and 3 VMs using VirtualBox)	Job response time Data skew in the map task Job completion time Makespan Task execution time System load	Dataset: RandomWriter Job: Sort, Grep, WordCount, GridMix2	ERUL: proposing two speculators for accurate estimating of the task remaining time	Reduction in job completion time Accurate estimation Higher throughput Makespan improvement Task execution time improvement	Failure in evenly reduce task input data assumption

Table 9 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
15	Tian et al. [81]	Hadoop cluster (4 PMs and 32 VMs)	Task response time Makespan Resource utilization	Dataset: Wikipedia article traffic logs, Complex workload (Facebook and Yahoo!) Job: WikiTrends (Select, TextSearch, Aggregation, Join) TeraSort WordCount	HScheduler: proposing a Johnson model-based framework for minimizing the makespan of off-line and online jobs	Makespan improvement Considering of multi-wave jobs	Not considering the preemption Not considering the energy efficiency Higher cost due to additional process such as job setting up, dispatching, and migration Not considering the load balancing
16	Wang et al. [82]	Hadoop cluster (8 PMs and 15 VMs using Xen)	Job completion time The efficiency of speculative execution Network size of shuffling Scalability Waiting timeout of speculative tasks Block size Map selectivity	Analytical workload Job: Micro benchmark (WordCount, Grep), Machine learning (K-means)	PSE: starting speculative tasks from the checkpoint using partial speculative execution approach to reduce operations costs	Reduction in operation costs such as re-reading, re-copying, and re-computing the processed data Reduction in job completion time Higher efficiency of speculative execution Scalable Applicable	

The framework provides a secure environment and does not require hard coding programming to perform policy specification and function assigned to the jobs.

Table 10 shows an overview of the Hadoop security-related papers.

3.2.7 Resource provisioning studies

Khan et al. [89] have presented a job performance model to provision resources for deadline-assigned multi-waves jobs. The model is constructed based on the historical job execution records, allocated map and reduce slots, and size of the input dataset. The model estimates the job execution time using locally weighted linear regression and provisions the required amount of resources based on Langrage multiplier technique. To hinder the resource provisioning bias (over-provisioning or under-provisioning), the average of the best-case and worst-case execution of a job is considered.

Nghiem et al. [90] have addressed the resource provisioning problem while considering the energy consumption and performance trade-off. The authors have defined the optimal number of tasks for a set of jobs using the actual sampled runtime data of the cluster, and there is no need to rely on the rules of thumbs. The optimal number is achieved by considering the trade-off between data locality and resource utilization which is handled by tuning split size for CPU-bound and I/O-bound jobs. The author's approach is based on the accuracy of optimal resource provisioning per application on a particular system. This method saves energy significantly up to several million dollars; however, users should establish a database which is required for jobs profiling.

Application-centric SSD caching for Hadoop applications (AC-SSD), which reduces the job completion time has been proposed by Tang et al. [91]. This approach uses the genetic algorithm to calculate the nearly optimal weights of virtual machines for allocating SSD cache space and controlling the I/O operations per second (IOPS) based on the importance of the VMs. Furthermore, it proposes a closed-loop adaptation to face the rapidly changing workload. Considering the importance of VMs and relationships among VMs inside the application improves the performance. Table 11 shows an overview of the papers.

4 Results and discussion

After synthesizing the data, we answered to our research questions RQ1 to RQ6 in this section.

Answer to Question RQ1 What topics have been considered most in MapReduce field?

Of the 55 studies that provided MapReduce topics, the greatest number of studies ($N = 16$) could be accounted for on the topic performance. We can see that two other subjects, namely scheduling with the number of 9 (16%) articles and load balancing with the number of eight (15%) articles, are the next most investigated research topics. Of the remaining, 7 (13%) articles focused on energy efficiency, 6 (11%) articles on security, 6 (11%) articles on fault tolerance, and 3 (5%) articles on resource provisioning. Figure 7 shows the percentage of studies frequency of each topic on the corresponding slice of the pie chart.

Figure 8 shows the most frequent topics, investigated by each publisher. IEEE has mostly considered performance topic, i.e., eleven articles out of sixteen (69%). Elsevier has mostly investigated fault tolerance topic i.e., four articles out of six (67%). Springer has mostly considered the energy efficiency topic, i.e., four studies out of six (67%), and ACM has mostly considered the security topic i.e., three studies out of six (50%).

Answer to Question RQ2 What are the main parameters, investigated by the studies?

According to Fig. 9, of the 55 studies included in our research, 25% ($N=14$) considered job completion time and makespan as main parameters, 24% ($N=13$) of studies considered scalability and data locality parameters, and 22% (12) considered input data size parameter. Job execution time and network in terms of network traffic overhead, network I/O (transmission cost), network delay, and network stability are the next most investigated parameters, considered by 20% ($N=11$) of studies. 18% ($N=10$) of studies considered a number of map and reduce tasks, while 16% ($N=9$) of the studies considered the size of intermediate data produced by the map tasks. In 15% ($N=8$) of the studies, the execution time of either map or reduce task has been considered, and SLA has been considered by 9% ($N=5$) of the studies.

Answer to Question RQ3 What are the main artifacts produced by the research?

The four main artifacts produced by the study on MapReduce are shown in Fig. 10: algorithms, frameworks, architectures, and topology.

When a paper has a logical view, i.e., like a design pattern, we put it in the architecture category. When a paper implements an architecture, we put it in the framework category. Algorithm category consists of the papers which have introduced a method, an algorithm, an approach, a schema, and a strategy to enhance the MapReduce functionality. Mostly, schedulers belong to this category. Furthermore, a topology is proposed when the shuffling network design is supposed to be considered.

Using this classification, half of the papers have contributed an algorithm to enhance the MapReduce functionality, whereas topology has been less proposed. The number of each artifact investigated by the publishers is shown in Fig. 11. Besides, we show the studies belong to each artifact in Table 12.

By categorizing the papers based on the software and hardware solutions, about 93% ($N=51$) of the studies have improved the MapReduce challenges through software solutions, i.e., algorithm. But only 7% ($N=4$) of the studies [37, 39, 40, 69] have employed hardware technologies as an improvement tool. The reason is that, on the one hand, using new and high-speed hardware for facing challenges

Table 10 An overview of existing primary studies focusing on "Security"

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
1	Fu et al. [83]	Hadoop cluster (16 nodes using VirtualBox)	Data leakage Reliability	–	Proposing a framework for investigating data leakage attacks in Hadoop cluster	Detection of suspicious data leakage behaviors An efficient way to locate attacked nodes Finding attacker and reliable evidence Reconstruction of the entire scenario	–
2	Parmar et al. [84]	Hadoop cluster (single node)	Encryption cost Machine performance	Dataset: synthetic (6 various file sizes)	Kuber: a three-dimensional security technique to remove the Hadoop security vulnerabilities	Cost-effective technique High memory performance in encryption/decryption compared to default Hadoop Encryption Zone More flexible Managing encryption credentials securely on the client side	The slow speed of encryption Not integrated with KMS Hadoop encryption service
3	Gupta et al. [85]	Hadoop cluster	SLA	–	Multilayer authorization framework for a representative Hadoop ecosystem deployment	Meet users SLA Two-layer data access checking Tag-based data access policy using the Atlas framework	Not considering the authorization level in Hadoop daemons

Table 10 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main idea	Advantages	Disadvantages
4	Wang et al. [86]	Hadoop cluster (9 nodes)	Cluster performance Makespan Task execution time	Dataset: Wikipedia, Synthetic (using TeraGen) Jobs: TeraSort, WordCount, WordMean	SEINA: A stealthy and effective internal attack in Hadoop systems	Higher system performance in the presence of an attack Minor overhead Improvement in task execution time	Not considering the reduce tasks attacks
5	Ohrimenko et al. [87]	Hadoop cluster (8 nodes)	System overhead Memory usage	Dataset: Census data sample, New York taxi ride Job: aggregate, aggregate filter	Observing and Preventing leakage in MapReduce	Lower I/O overhead Evaluation of framework on secure implemented of Hadoop, VC3. The lower overhead of framework due to pregrouping values with the same key Implementation in java and C++	–
6	Ulusoy et al. [88]	Hadoop cluster (7 nodes)	Scalability Cluster performance	Dataset: Twitter, Google images	GuardMR: fine-grained security policy enforcement for MapReduce systems	High efficiency Small overhead Scalable High modularity and flexibility User-transparent framework Practical policy specification	Lower performance due to performing reflection operations in the reference monitor

Table 11 An overview of existing primary studies focusing on "resource provisioning"

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
1	Khan et al. [89]	Hadoop cluster (8 VM nodes) Amazon EC2 Cloud (20 instances)	Map, Shuffle and Reduce phase execution time Job execution time Number of reduce tasks Input data size	Dataset: TeraGen Jobs: CPU-intensive (WordCount), IO-intensive (Sort)	Estimating job completion time based on a Hadoop job performance model using "locally weighted linear regression" technique and resource provisioning for deadline-assigned jobs using "Lagrange multipliers" technique	Accuracy of the proposed model in job execution estimation The proposed HP model is economical in terms of resource provisioning Reduction in job execution time	The model over-provisions when there are more virtual machines The method does not consider the jobs with logical dependency
2	Nghie et al. [90]	Hadoop cluster (24 nodes)	Energy efficiency Task execution time Job execution time CPU time Number of Reduce tasks	Dataset: TeraGen Job: TeraSort	A resource provisioning algorithm with a mathematical formula for obtaining the exact optimal number of task resources for any workload	The accurate and optimal number of reduce tasks Improvement in energy consumption Improvement task execution time Usable in other MapReduce implementation frameworks Reduction in job execution time	Not considering the heterogeneity of the environment Not considering the scalability

Table 11 (continued)

Nos.	References	Experimental platform	Parameters	Dataset/workload	Main Idea	Advantages	Disadvantages
3	Tang et al. [91]	Hadoop cluster (4 PMs forms 3 clusters, 20 VMs)	CPU time Job completion time Cache size Network I/O	Micro benchmark, TestDFSIO benchmark, WordCount, TeraSort, Sort, Aggregation, Join Scan, Bayes, PageRank, K-means	Application-centric SSD cache allocation for Hadoop applications	Considering the application-centric instead of VM-centric SSD caching schemes Shortest job completion time Higher performance	The simple and not accurate solution to detect workload changes Performance degradation during provisioning

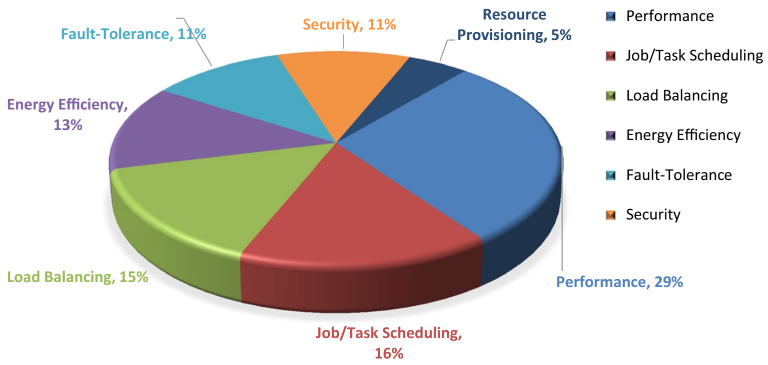


Fig. 7 Research topics ranked by the percentage of publications

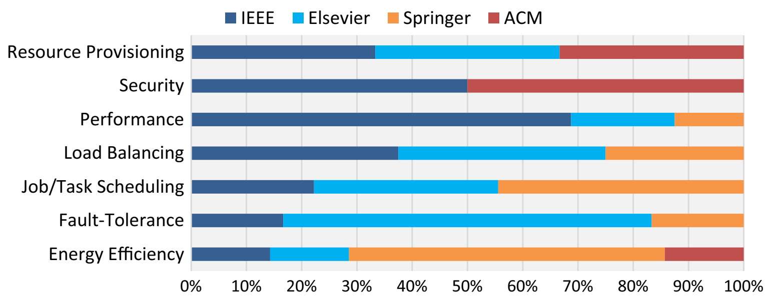


Fig. 8 Percentage of investigated topics per publishers

imposes more costs to the developer, and on the other hand, the researchers who wish to compare their work to these studies are forced to re-extend or spend high cost (if the hardware is accessible!) to simulate the same situations. Hence, the number of citations of these papers will be impacted by this case.

Answer to Question RQ4 What experimental platforms have been used by the researchers for analysis and evaluation?

We classified the experimental platforms into three categories: simulation, implementation using Cloud services, and implementation in the test bed. Therefore, based on these categories, in 71% (N=39) of studies which evaluated the results using implementation, cloud with 7% (N=4), test bed with 13% (N=7), in-house Hadoop cluster with 51% (N=28) [20, 42, 43, 47, 48, 50–52, 54, 56, 57, 60, 62, 67–69, 71, 74, 75, 78, 80–91] have been used. Of the test bed category, Grid'5000 is used in four studies [37, 41, 46, 61], SAVI test bed is used in two studies [58, 63], and MobiWay is used in one study [55], respectively. 9% (N=5) of studies [39, 45, 64, 76, 77] only used simulation to evaluate the results in which one study [39] have used CloudSim and the rest of studies have

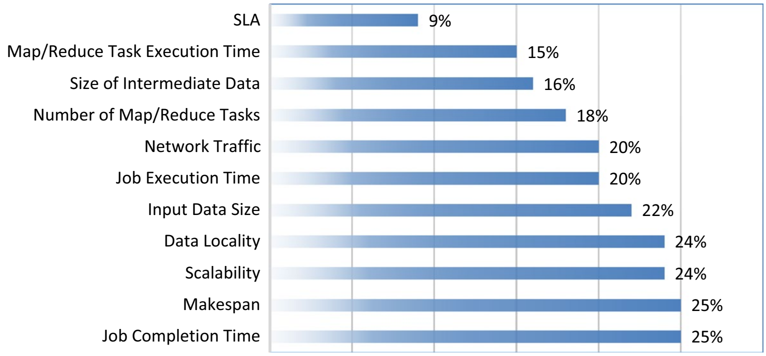


Fig. 9 Investigated percentage of each parameter

used stock simulator. 20% (11) of studies [15, 36, 38, 40, 44, 49, 59, 65, 66, 70, 73] have used both simulation and implementation as the experimental platform in which in terms of implementation, studies [15, 44] have been implemented in cloud and the rest of studies have been implemented in in-house Hadoop cluster. In terms of simulation, studies [15, 44, 59] have used their extended simulator: TDEMR, CloudSimMR, and TopoSim and the others have used stock simulator.

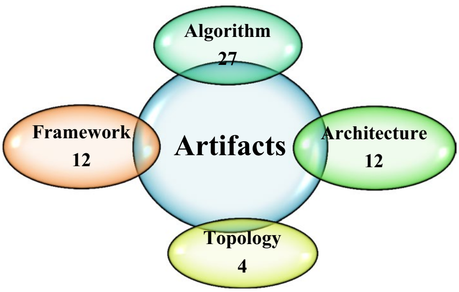
The virtualized tools, used in the studies include Xen, VMWare, KVM, and VirtualBox. The statistics are shown in Fig. 12.

Answer to Question RQ5 What kind of jobs, benchmarks, and dataset have been used in the experiments? And what percentage of each one has been used in the studies?

For answering this question, we have provided the job name and its functionality, job shuffle degree in terms of heavy or light shuffling, dataset, and benchmarks in Table 13.

According to Table 13, jobs are categorized as shuffle-light and shuffle-heavy in terms of produced intermediate data by map tasks. Of the total number of publications included in this study, six benchmarks have been used: PUMA, HiBench, MicroBench, MRBench, TestDFSIO, and Built-in YARN, included

Fig. 10 Four main artifacts of studies



in 42% ($N=23$) of the studies. Among all, PUMA is used frequently by 44% ($N=10$), HiBench is the second most used benchmark by 26% ($N=6$), while MicroBench and MRBench are used by 13% ($N=3$) and 9% ($N=2$) of studies, respectively. TestDFSIO and built-in YARN are used in only 4% ($N=1$) of studies. The remaining studies which are 58% ($N=32$) have used a different combination of common jobs of Table 13. Figure 13 shows these statistics.

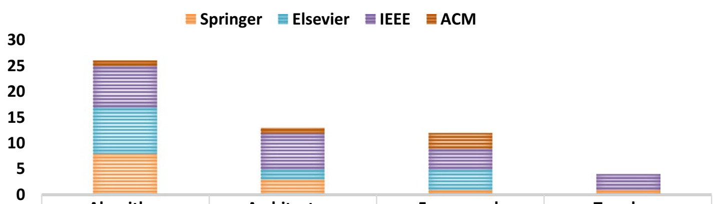
From the 55 existing articles about the MapReduce framework presented in this study, 51 papers have used the jobs which have been shown in Table 14. However, there is any information about the dataset or jobs which have been used in four studies [47, 64, 83, 85]. Figure 14 shows the percentage that each job has been used in the 55 articles (popularity).

Answer to Question RQ6 What are the open challenges and future directions in Hadoop MapReduce?

- Open challenges

To answer this question, some of the challenges presented in the section of reviewed papers have been considered. However, some yet challenging problems in MapReduce can be mentioned as follows:

- Hadoop MapReduce has been widely discussed to improve performance. Some researches try to improve the performance by studying the dependency of the workflow and to reach the data locality. Separating the phases as independent jobs brings better performance. However, most of the jobs have a dependency on them, so how to justify the independency of them is a yet challenging problem.
- By decoupling the phases to accelerate the computations, there would be a dilemma between speed and scalability. The MapReduce model is designed for scalability, so how to maintain the scalability in the decoupled design is another issue.



	Algorithm	Architecture	Framework	Topology
ACM	1	1	3	0
IEEE	8	7	4	3
Elsevier	9	2	4	0
Springer	8	3	1	1

Fig. 11 Number of each artifact investigated by the publishers

Table 12 Classification of studies based on the artifacts

Algorithm								
BeTL [20]	Xu et al. [48].	MRCP-RM [49]	Tang et al. [65]	Balanced Pools [66]	Chen et al. [74]	Xu et al. [76]	Khan et al. [89]	Ibrahim et al. [37]
FARMS [43]	Memishi et al. [45]	Chronos [46]	SARS [51]	MDRP [62]	Jiang et al. [77]	ERUL [80]	Nghiem et al. [90]	Song et al. [38]
Jia-Chun Lin et al. [47]	Bok et al. [52]	Hashem et al. [54]	Momth [55]	Tang et al. [56]	HScheduler [81]	Yaoguang Wang et al. [82]	Wang et al. [86]	Arjona et al. [42]
Architecture								
LIBRA [57]	RFHOC [67]	Cheng et al. [68]	iShuffle [71]	FlexSlot [75]	Fu [83]	HPSO [50]	Flame-MR [78]	OPTIMA [63]
Tang et al. [91]	Phan et al. [41]	Teng et al. [40]						
Framework								
Mashayekhy et al. [36]	DREAMS [58]	Hadoop-A [69]	ANRDF [44]	Kao et al. [15]	FP-Hadoop [61]	MBR [60]	Parmar et al. [84]	Gupta et al. [85]
Ohrimenko et al. [87]	Ulusoy et al. [88]	Cai et al. [39]						
Topology								
Paik [59]	Guo et al. [70]	Guo et al. [73]	Zhang et al. [64]					

- Many production jobs are executed in the cluster of Hadoop using the MapReduce programming model. Therefore, makespan of these jobs is an important issue which should be considered as an effective metric in performance. The order in which jobs are executing has a significant impact on makespan.
- Systematically exploring the Hadoop parameters space and finding a near-optimal configuration are a challenge. Some new intelligent algorithms and techniques which are based on the cluster and workload properties are required to suggest an appropriate parameter setting.
- Network overhead is another serious problem in prolonging execution of jobs. To overcome this issue, designing new algorithms and techniques are required to improve and accelerate the shuffle phase of MapReduce.
- The straggler tasks which are caused by internal and external problems such as resource competition, hardware heterogeneity, hardware failure, and data skew should be considered as the other performance metrics. How to select the straggler tasks and how to define the proper node to host the tasks are the notable challenges in the speculative strategies. Moreover, some energy consumption models are required to prevent waste of energy on killed speculative copies.
- There are many kinds of MapReduce jobs such as production, interactive, and deadline-assigned jobs. On the one hand, we should be able to provide resources at run-time to meet jobs requirements. On the other hand, this provisioning should not cause "Bias" which influences energy efficiency and performance.
- Enterprises and IoT providers use Hadoop Lake to store and process data generated from IoT devices. In this situation, security and privacy requirements are critical challenges for the prominent technology firms and state. Providing protective schemes in terms of authentication, authorization, and data confidentiality are imperative to secure Hadoop system in the presence of attacks. To prevent and confront the attacks, Hortonworks [92] have divided Hadoop security vulnerabilities into three parts: (1) systemic; (2) operational; and (3) architectural. By researching and presenting new solutions on each domain, we can overcome Hadoop security problems. Table 14 shows an overview of the challenges.

- Future directions

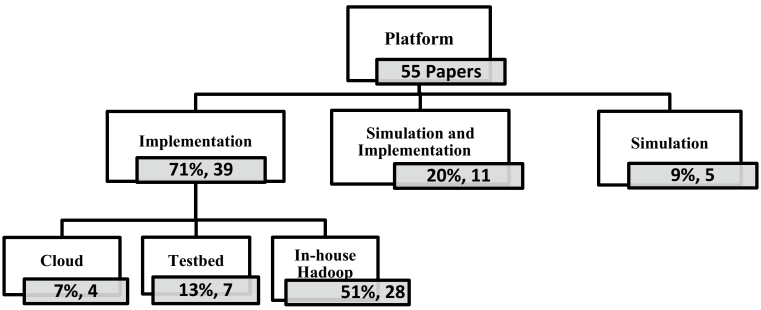


Fig. 12 Percentage of environments which have been used in the studies

Table 13 Benchmarks, dataset, job name, and its functionality, shuffling degree

Dataset	Functionality	Shuffling degree	Job	Benchmark
TeraGen	Counts the occurrence of each distinct word in a text file	Heavy	WordCount	PUMA
Wikipedia Article Traffic Logs Complex (Event Log of Facebook, Yahoo!)	Is used for Google search results: it refers to Web sites counting the number of links and the quality of the links they refer to	Heavy	PageRank	HiBench MicroBench Built-in Yarn
SwimGen RandomTextWriter	Counts the number of occurrences of strings matching the target in a text file	Light	Grep	WikiTrends BigDataBench
DatasetGen	Mining a graph to determine its sub-networks	Heavy	Connected components	ApplicationBench
Synthetic Netflix WikiMedia	A database index storing a mapping from content, such as words or numbers, to its locations in a table, or in a document or a set of documents	Heavy	Inverted-index	MRBench TestDFSIO
Cloud Ships and Land Station Traces BRITEGen Live Journal Graph Data	Returns information and statistics about phrases in the context of a particular document	Heavy	Term-vector	
Google Web Graph Newyork Taxi Ride	Returns number of votes about movies, registered by users	Light	Histogram ratings, Histogram movies	
Census Data Samples Twitter Google Images	A clustering which its purpose is to divide n observation into k-clusters Each observation belongs to the cluster that has the closest average, selected as the prototype	Heavy	K-means	
	Sorts a text file with a volume of a terabyte	Heavy	TeraSort	
	Predict class membership probabilities such as the probability that a given tuple belongs to a particular class	Heavy	Bayes	
	Joins a table with itself—each row in the table is combined with itself and with other rows of the table	Heavy	SelfJoin	

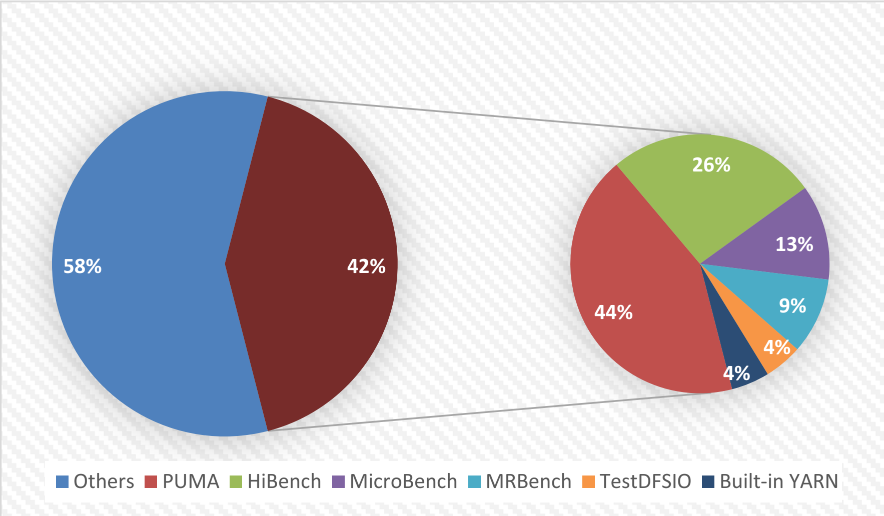


Fig. 13 Percentage of most common used benchmarks in the articles

Table 14 Three-dimensional security of Hadoop cluster [92]

Systemic
Data access and ownership
Data at rest and data in motion protection
Multi-tenancy
Inter-node communication
Client interaction
Distributed nodes
Operational
Authentication and authorization
Administrative data access
Configuration and patch management
Authentication of applications and nodes
Audit and logging
Monitoring, filtering, and blocking
API security
Architectural
Walled garden
Cluster security
Data-centric security
Enterprise security
Embedded security

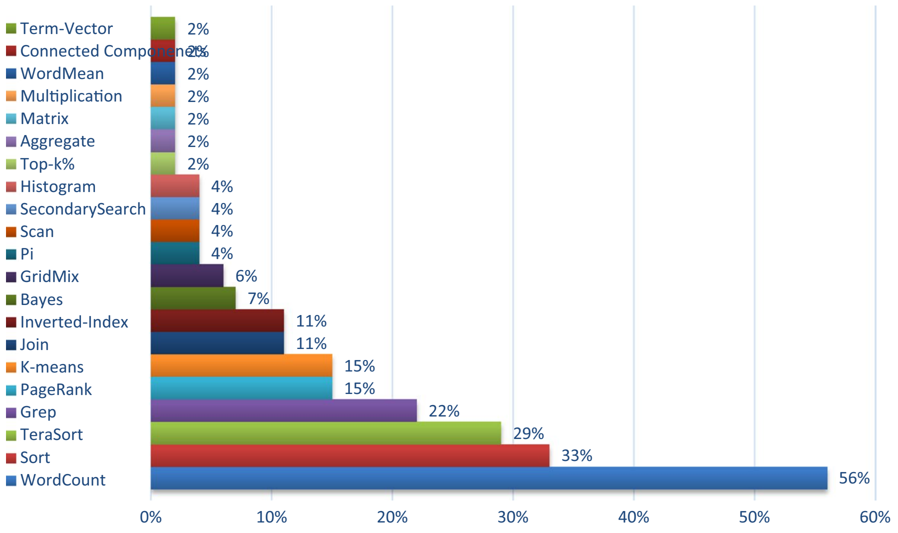


Fig. 14 Percentage that each job has appeared in the articles

Although many signs of progress have been gain, there are still several open issues in the MapReduce at the infrastructure level. Therefore, after studying related papers in MapReduce, we will discuss some unmentioned issues that can be studied and analyzed further. We enumerate some promising future directions in Hadoop MapReduce as follows:

- General platform: by integrating MapReduce and Spark, we can benefit a general platform in which the batch, streaming, iterative, and in-memory applications can be executed simultaneously in a Hadoop cluster. We can employ a dedicated pool for each application type or group of users and reach better performance and power saving.
- Artificial intelligence approaches: we can build accurate and robust performance prediction models based on historical data in each Hadoop phases and feed these models output to algorithms such as genetic, smart hill climbing, and machine learning. Using the qualified search in the Hadoop configuration space, these methods can find optimal or near-optimal configuration with high probability. These methods help developers to not scramble with manually configuration of Hadoop configuration parameters.
- Combination techniques: hardware approaches such as dynamic voltage and frequency scaling, SSD-based in-storage computing, and remote-based data access controllers along with pipelining the map, sort, shuffle, and reduce phases can improve the power consumption of a MapReduce cluster.
- Software-based approaches: we can employ algorithms in which the placement of data, produced by mappers is earlier defined so that the partition, belonged to a specified reducer would be available by and by during completion of map

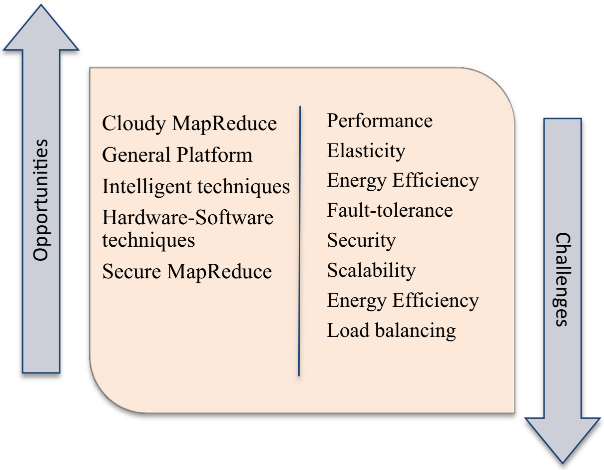


Fig. 15 Challenges and opportunities in MapReduce area

phase. In such way, the heavy shuffling of the shuffle phase is divided into light shuffling and accelerates the job execution time.

- MapReduce Model: by defining an appropriate execution model based on the heterogeneity of systems such as application type, data type and format, server characteristics, topology and communication medium type, and workload prediction, we can reach to higher performance.
- Cloudy MapReduce: since MapReduce programming model accelerates Big data processing, deploying MapReduce in IaaS clouds can maximize the performance of cloud infrastructure service. Furthermore, we can service MapReduce to cloud users for running their MapReduce applications in the cloud. Besides, we can benefit fine-grained cluster security using cloud-based MapReduce.
- Cluster Topology: shuffling is a network-consuming stage in geo-distributed MapReduce-based datacenters. The default network topology of Hadoop is flat, i.e., "tree" [14, 59] which does not support scalability and causes higher data computation and communication costs. Although there are two masters (one as a backup) in a Hadoop cluster, how many nodes can deploy in a sub-cluster and how the masters of the sub-clusters should communicate with each other are already the open issues.
- Secure MapReduce: To secure Hadoop cluster, the robust and efficient algorithms are required in four aspects of security including authentication, authorization, auditing, and data access. To prevent and confront the attacks, some solutions including new user authentication protocols such as Kerberos [93], robust encryption algorithms for data communication between Hadoop daemons, and powerful data-at-rest access control mechanisms can be employed. Further, we can design and develop visualizations tools and intelligent algorithms to predictive models for informing the system administrator of data spillage and destructive attacks using attack patterns detection and provenance logs.

- Cost-effective MapReduce: the mentioned challenges impose costs in terms of energy consumption. To alleviate the costs, we can focus on the solutions which mitigate job execution time. Load skew handling including online solutions (quickly aggregating intermediate data and then estimating the reduce task workload), writing customized partitioners, multi-level partitioners, optimal schedulers such as run-time map task split binding or the run-time reduce task partition binding, powerful speculative mechanisms, and efficient data replication algorithms reduce the job execution time and subsequently the required energy. In this way and with this outlook, we reach "Green MapReduce" since the carbon emissions are controlled. Figure 15 shows a summary of challenges and opportunities.

5 Conclusions and limitations

In this paper, we have conducted a holistic study systematically in Hadoop MapReduce. First, we had an architectural overview of Hadoop main components. After describing our research methodology, we classified the MapReduce studies into seven areas: (1) performance; (2) job/task scheduling; (3) load balancing; (4) resource provisioning; (5) fault tolerance in terms of availability and reliability; (6) security; and (7) energy efficiency. Afterward, we extracted the main idea, discussed strengths and drawbacks and provided our observations by answering the research questions. The chronicle of studies reflects the attention to the challenges of MapReduce as a Big data processing platform among the researches. The majority (16 out of 55 articles) of studies have focused on performance as the most significant topic in MapReduce, while scheduling, load balancing, energy efficiency, security, fault tolerance, and resource provisioning are the next most considered topics, respectively. We defined the future direction and presented several potential solutions to researchers, interested in MapReduce area.

We studied the major investigated challenges of MapReduce framework as well as the best-proposed solutions and tried hard to provide a comprehensive systematic study. But, the study might have some limitations which are our plan to address them in future studies. Searching only digital libraries using search string keywords is just one of the many channels of finding research activity stream about a widely focused topic like MapReduce. Two search approaches for future study are as follows: (1) using other means such as Ph.D. theses, academic blogs, editorial notes, and technical reports and (2) relaxing some of the strict exclusion criteria such as considering the interdisciplinary articles, national journals, and conferences, and non-English articles, it might help us to become familiar with other worthy solutions.

References

1. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

2. Hashem IAT, Anuar NB, Gani A, Yaqoob I, Xia F, Khan SU (2016) MapReduce: review and open challenges. Scientometrics 109(1):389–422
3. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R (2010) Hive—a petabyte scale data warehouse using Hadoop. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)
4. Polato I, Ré R, Goldman A, Kon F (2014) A comprehensive view of Hadoop research—a systematic literature review. J Netw Comput Appl 46:1–25
5. Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. IEEE Access 2:652–687
6. Chen CP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347
7. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mob Netw Appl 19(2):171–209
8. http://spark.apache.org/
9. http://datampi.org/
10. Soualhia M, Khomh F, Tahar S (2017) Task scheduling in big data platforms: a systematic literature review. J Syst Softw 134:170–189
11. Zhang B, Wang X, Zheng Z (2018) The optimization for recurring queries in big data analysis system with MapReduce. Future Gener Comput Syst 87:549–556
12. http://hadoop.apache.org/
13. Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies (MSST)
14. White T (2009) Hadoop: the definitive guide. O'Reilly Media Inc, Sebastopol
15. Kao Y-C, Chen Y-S (2016) Data-locality-aware mapreduce real-time scheduling framework. J Syst Softw 112:65–77
16. Wang F, Qiu J, Yang J, Dong B, Li X, Li Y (2009) Hadoop high availability through metadata replication. In: Proceedings of the first international workshop on cloud data management. ACM, Hong Kong, pp 37–44
17. Li F, Ooi BC, Tamer Ozsu M, Wu S (2014) Distributed data management using MapReduce. ACM Comput Surv 46(3):1–42
18. Singh R, Kaur PJ (2016) Analyzing performance of Apache Tez and MapReduce with Hadoop multinode cluster on Amazon cloud. J Big Data 3(1):19
19. https://www.bogotobogo.com/Hadoop/BigData_hadoop_Ecosystem.php
20. Wang H, Chen H, Du Z, Hu F (2016) BeTL: MapReduce checkpoint tactics beneath the task level. IEEE Trans Serv Comput 9(1):84–95
21. Alapati SR (2016) Expert Hadoop administration: managing, tuning, and securing spark, YARN, and HDFS. Addison-Wesley Professional, Boston
22. Gupta M, Patwa F, Sandhu R (2017) Object-tagged RBAC model for the Hadoop ecosystem. In: IFIP Annual Conference on Data and Applications Security and Privacy. Springer
23. Erraissi A, Belangour A, Tragha A (2017) A big data Hadoop building blocks comparative study. Int J Comput Trends Technol 48(1):36–40
24. Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: an update. Inf Softw Technol 64:1–18
25. Cruz-Benito J (2016) Systematic literature review & mapping. https://doi.org/10.5281/zenodo.165773
26. Lu Q, Zhu L, Zhang H, Wu D, Li Z, Xu X (2015) MapReduce job optimization: a mapping study. In: 2015 International Conference on Cloud Computing and Big Data (CCBD)
27. Charband Y, Navimipour NJ (2016) Online knowledge sharing mechanisms: a systematic review of the state of the art literature and recommendations for future research. Inf Syst Front 18(6):1131–1151
28. Poggi N, Carrera D, Call A, Mendoza S, Becerra Y, Torres J, Ayguadé E, Gagliardi F, Labarta J, Reinauer R, Vujic N, Green D, Blakeley J (2014) ALOJA: a systematic study of Hadoop deployment variables to enable automated characterization of cost-effectiveness. In: 2014 IEEE International Conference on Big Data (Big Data)
29. Sharma M, Hasteer N, Tuli A, Bansal A (2014) Investigating the inclinations of research and practices in Hadoop: a systematic review. In: 2014 5th International Conference—Confluence the Next Generation Information Technology Summit (Confluence)

30. Thakur S, Ramzan M (2016) A systematic review on cardiovascular diseases using big-data by Hadoop. In: 2016 6th International Conference—Cloud System and Big Data Engineering (Confluence)
31. Lu J, Feng J (2014) A survey of mapreduce based parallel processing technologies. China Commun 11(14):146–155
32. Derbeko P, Dolev S, Gudes E, Sharma S (2016) Security and privacy aspects in MapReduce on clouds: a survey. Comput Sci Rev 20:1–28
33. Li R, Hu H, Li H, Wu Y, Yang J (2016) MapReduce parallel programming model: a state-of-the-art survey. Int J Parallel Prog 44(4):832–866
34. Iyer GN, Silas S (2015) a comprehensive survey on data-intensive computing and mapreduce paradigm in cloud computing environments. In: Rajsingh EB, Bhojan A, Peter JD (eds) Informatics and communication technologies for societal development: proceedings of ICICTS 2014. Springer India, New Delhi, pp 85–93
35. Liu Q, Jin D, Liu X, Linge N (2016) a survey of speculative execution strategy in MapReduce. In: Sun X, Liu A, Chao H-C, Bertino E (eds) Cloud Computing and Security: Second International Conference, ICCCS 2016, Nanjing, China, July 29–31, 2016, Revised Selected Papers, Part I. Springer, Cham, pp 296–307
36. Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W (2015) Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Trans Parallel Distrib Syst 26(10):2720–2733
37. Ibrahim S, Phan T-D, Carpen-Amarie A, Chihoub H-E, Moise D, Antoniu G (2016) Governing energy consumption in Hadoop through cpu frequency scaling: an analysis. Future Gener Comput Syst 54:219–232
38. Song J, He H, Wang Z, Yu G, Pierson J-M (2016) Modulo based data placement algorithm for energy consumption optimization of MapReduce system. J Grid Comput 1:1–16
39. Cai X, Li F, Li P, Ju L, Jia Z (2017) SLA-aware energy-efficient scheduling scheme for Hadoop YARN. J Supercomput 73(8):3526–3546
40. Teng F, Yu L, Li T, Deng D, Magoulès F (2017) Energy efficiency of VM consolidation in IaaS clouds. J Supercomput 73(2):782–809
41. Phan T-D, Ibrahim S, Zhou AC, Aupy G, Antoniu G (2017) Energy-driven straggler mitigation in MapReduce. In: European Conference on Parallel Processing. Springer
42. Arjona Aroca J, Chatzipapas A, Fernández Anta A, Mancuso V (2014) A measurement-based analysis of the energy consumption of data center servers. In: Proceedings of the 5th International Conference on Future Energy Systems. ACM
43. Fu H, Chen H, Zhu Y, Yu W (2017) FARMS: efficient mapreduce speculation for failure recovery in short jobs. Parallel Comput 61:68–82
44. Tang B, Tang M, Fedak G, He H (2017) Availability/network-aware MapReduce over the internet. Inf Sci 379:94–111
45. Memishi B, Pérez MS, Antoniu G (2017) Failure detector abstractions for MapReduce-based systems. Inf Sci 379:112–127
46. Yildiz O, Ibrahim S, Antoniu G (2017) Enabling fast failure recovery in shared Hadoop clusters: towards failure-aware scheduling. Future Gener Comput Syst 74:208–219
47. Lin J-C, Leu F-Y, Chen Y-P (2015) Analyzing job completion reliability and job energy consumption for a heterogeneous MapReduce cluster under different intermediate-data replication policies. J Supercomput 71(5):1657–1677
48. Xu X, Cao L, Wang X (2016) Adaptive task scheduling strategy based on dynamic workload adjustment for heterogeneous Hadoop clusters. IEEE Syst J 10(2):471–482
49. Lim N, Majumdar S, Ashwood-Smith P (2017) MRCP-RM: a technique for resource allocation and scheduling of MapReduce jobs with deadlines. IEEE Trans Parallel Distrib Syst 28(5):1375–1389
50. Sun M, Zhuang H, Li C, Lu K, Zhou X (2016) Scheduling algorithm based on prefetching in MapReduce clusters. Appl Soft Comput 38:1109–1118
51. Tang Z, Jiang L, Zhou J, Li K, Li K (2015) A self-adaptive scheduling algorithm for reduce start time. Future Gener Comput Syst 43:51–60
52. Bok K, Hwang J, Lim J, Kim Y, Yoo J (2016) An efficient MapReduce scheduling scheme for processing large multimedia data. Multimed Tools Appl 76(16):1–24
53. Zaharia M, Borthakur D, Sarma JS, Elmeleegy K, Shenker S, Stoica I (2010) Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European Conference on Computer systems. ACM, Paris, pp 265–278

54. Hashem IAT, Anuar NB, Marjani M, Gani A, Sangaiah AK, Sakariyah AK (2017) Multi-objective scheduling of MapReduce jobs in big data processing. Multimed Tools Appl 77(8):1–16
55. Nita M-C, Pop F, Voicu C, Dobre C, Xhafa F (2015) MOMTH: multi-objective scheduling algorithm of many tasks in Hadoop. Cluster Comput 18(3):1011–1024
56. Tang Z, Liu M, Ammar A, Li K, Li K (2016) An optimized MapReduce workflow scheduling algorithm for heterogeneous computing. J Supercomput 72(6):2059–2079
57. Chen Q, Yao J, Xiao Z (2015) LIBRA: lightweight data skew mitigation in MapReduce. IEEE Trans Parallel Distrib Syst 26(9):2520–2533
58. Liu Z, Zhang Q, Ahmed R, Boutaba R, Liu Y, Gong Z (2016) Dynamic resource allocation for MapReduce with partitioning skew. IEEE Trans Comput 65(11):3304–3317
59. Chen W, Paik I, Li Z (2016) Topology-aware optimal data placement algorithm for network traffic optimization. IEEE Trans Comput 65(8):2603–2617
60. Li J, Liu Y, Pan J, Zhang P, Chen W, Wang L (2017) Map-balance-reduce: an improved parallel programming model for load balancing of MapReduce. Future Gener Comput Syst
61. Liroz-Gistau M, Akbarinia R, Agrawal D, Valduriez P (2016) FP-Hadoop: efficient processing of skewed MapReduce jobs. Inf Syst 60:69–84
62. Myung J, Shim J, Yeon J, Lee S-G (2016) Handling data skew in join algorithms using MapReduce. Expert Syst Appl 51:286–299
63. Liu Z, Zhang Q, Boutaba R, Liu Y, Wang B (2016) OPTIMA: on-line partitioning skew mitigation for MapReduce with resource adjustment. J Netw Syst Manag 24(4):859–883
64. Zhang X, Jiang J, Zhang X, Wang X (2015) A data transmission algorithm for distributed computing system based on maximum flow. Cluster Comput 18(3):1157–1169
65. Tang S, Lee BS, He B (2016) Dynamic job ordering and slot configurations for MapReduce workloads. IEEE Trans Serv Comput 9(1):4–17
66. Verma A, Cherkasova L, Campbell RH (2013) Orchestrating an ensemble of MapReduce jobs for minimizing their makespan. IEEE Trans Dependable Secure Comput 10(5):314–327
67. Bei Z, Yu Z, Zhang H, Xiong W, Xu C, Eeckhout L, Feng S (2016) RFHOC: a random-forest approach to auto-tuning Hadoop's configuration. IEEE Trans Parallel Distrib Syst 27(5):1470–1483
68. Cheng D, Rao J, Guo Y, Jiang C, Zhou X (2017) Improving performance of heterogeneous MapReduce clusters with adaptive task tuning. IEEE Trans Parallel Distrib Syst 28(3):774–786
69. Yu W, Wang Y, Que X (2014) Design and evaluation of network-levitated merge for Hadoop acceleration. IEEE Trans Parallel Distrib Syst 25(3):602–611
70. Guo D, Xie J, Zhou X, Zhu X, Wei W, Luo X (2015) Exploiting efficient and scalable shuffle transfers in future data center networks. IEEE Trans Parallel Distrib Syst 26(4):997–1009
71. Guo Y, Rao J, Cheng D, Zhou X (2017) iShuffle: improving Hadoop performance with shuffle-on-write. IEEE Trans Parallel Distrib Syst 28(6):1649–1662
72. Maleki N, Rahmani AM, Conti M (2018) POSTER: an intelligent framework to parallelize Hadoop phases. In: Proceedings of the 27th international symposium on high-performance parallel and distributed computing. ACM
73. Ke H, Li P, Guo S, Guo M (2016) On traffic-aware partition and aggregation in mapreduce for big data applications. IEEE Trans Parallel Distrib Syst 27(3):818–828
74. Chen Q, Liu C, Xiao Z (2014) Improving MapReduce performance using smart speculative execution strategy. IEEE Trans Comput 63(4):954–967
75. Guo Y, Rao J, Jiang C, Zhou X (2017) Moving Hadoop into the cloud with flexible slot management and speculative execution. IEEE Trans Parallel Distrib Syst 28(3):798–812
76. Xu H, Lau WC (2017) Optimization for speculative execution in big data processing clusters. IEEE Trans Parallel Distrib Syst 28(2):530–545
77. Jiang Y, Zhu Y, Wu W, Li D (2017) Makespan minimization for MapReduce systems with different servers. Future Gener Comput Syst 67:13–21
78. Veiga J, Expósito RR, Taboada GL, Tourino J (2016) Flame-MR: an event-driven architecture for MapReduce applications. Future Gener Comput Syst 65:46–56
79. Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I (2008) Improving MapReduce performance in heterogeneous environments. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. USENIX Association, San Diego, pp 29–42
80. Huang X, Zhang L, Li R, Wan L, Li K (2016) Novel heuristic speculative execution strategies in heterogeneous distributed environments. Comput Electr Eng 50:166–179
81. Tian W, Li G, Yang W, Buyya R (2016) HScheduler: an optimal approach to minimize the makespan of multiple MapReduce jobs. J Supercomput 72(6):2376–2393

82. Wang Y, Lu W, Lou R, Wei B (2015) Improving MapReduce performance with partial speculative execution. J Grid Comput 13(4):587–604
83. Fu X, Gao Y, Luo B, Du X, Guizani M (2017) Security threats to Hadoop: data leakage attacks and investigation. IEEE Netw 31(2):67–71
84. Parmar RR, Roy S, Bhattacharyya D, Bandyopadhyay SK, Kim T (2017) Large-Scale Encryption in the Hadoop Environment: challenges and Solutions. IEEE Access 5:7156–7163
85. Gupta M, Patwa F, Benson J, Sandhu R (2017) Multi-layer authorization framework for a representative Hadoop ecosystem deployment. In: Proceedings of the 22nd ACM on symposium on access control models and technologies. ACM
86. Wang J, Wang T, Yang Z, Mao Y, Mi N, Sheng B (2017) Seina: a stealthy and effective internal attack in Hadoop systems. In: 2017 International Conference on Computing, Networking and Communications (ICNC). IEEE
87. Ohrimenko O, Costa M, Fournet C, Gkantsidis C, Kohlweiss M, Sharma D (2015) Observing and preventing leakage in MapReduce. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, Denver, pp 1570–1581
88. Ulusoy H, Colombo P, Ferrari E, Kantarcioglu M, Pattuk E (2015) GuardMR: fine-grained security policy enforcement for MapReduce systems. In: Proceedings of the 10th ACM symposium on information, computer and communications security. ACM, Singapore, pp 285–296
89. Khan M, Jin Y, Li M, Xiang Y, Jiang C (2016) Hadoop performance modeling for job estimation and resource provisioning. IEEE Trans Parallel Distrib Syst 27(2):441–454
90. Nghiem PP, Figueira SM (2016) Towards efficient resource provisioning in MapReduce. J Parallel Distrib Comput 95:29–41
91. Tang Z, Wang W, Huang Y, Wu H, Wei J, Huang T (2017) Application-centric SSD cache allocation for Hadoop applications. In: Proceedings of the 9th Asia-pacific symposium on internetware. ACM
92. Hadoop S (2016) Security recommendations for Hadoop environments. White paper, Securosis
93. Garman J (2003) Kerberos: the definitive guide. O'Reilly Media, Inc