International Journal of Advanced Technology and Engineering Exploration, Vol 8(82)
ISSN (Print): 2394-5443 ISSN (Online): 2394-7454
http://dx.doi.org/10.19101/IJATEE.2021.874313

Review Article

Factors affecting cloud data-center efficiency: a scheduling algorithm-based analysis

Arif Ahmad Shehloo[1*], Muheet Ahmed Butt[2] and Majid Zaman[3]
Research Scholar, Mewar University, Chittorgarh, Rajasthan, India[1]
P.G Department of Computer Science, University of Kashmir, Srinagar, J & K, India[2]
Directorate of IT & Support System, University of Kashmir, Srinagar, J & K, India[3]

Received: 27-June-2021; Revised: 12-September-2021; Accepted: 16-September-2021

Abstract
Cloud computing encompasses two massively scalable services: computing capability and data storage space, which are provided by a massive number of machines and clusters. The increased use of big data has resulted in adopting a wide range of analytics engines, such as Hadoop. As a result, Hadoop has gained widespread acceptance as a data analytics platform. Over the past decade, Hadoop's ability to schedule tasks has become a critical aspect of system performance. Numerous researchers have presented various scheduling methods in their work to address the complex issue of performance degradation. However, few studies have been conducted to date to evaluate the effectiveness of these methods. By employing the PRISMA approach for searching and selecting papers, we examine the design choices that went into various Hadoop scheduling techniques proposed between 2008 and 2021. We present a taxonomy for succinctly categorising these scheduling techniques. Additionally, we evaluate methodologies based on a variety of performance metrics. Our search identified 82 studies relevant to this domain, all of which came from high-quality conferences, journals, symposiums, and workshops. This systematic study discusses various dynamic, constrained, and adaptive scheduling methods and their primary motivations, including makespan, data control, deadline, resource utilisation, load balancing, fairness, energy efficiency, and failure recovery. There is also a discussion of some unresolved issues and potential future directions for modifying existing studies. This study conducts a systematic review of the literature to identify and discuss the most critical factors affecting Hadoop scheduler performance and provide a roadmap for researchers working in this field. Finally, we intend to expand on the qualitative analysis conducted thus far and give the experts additional recommendations to conduct future cloud scheduling research.

Keywords
Big data, Cloud computing, Apache Hadoop, MapReduce, Task scheduling.

1.Introduction
Nowadays, users are required to cache, scrutinise, and process massive datasets from various fields, including science, business, and research. As a result, they require data-intensive platforms with ample storage and processing power. In addition, many of these kinds of platforms must-have features like parallel processing, fault tolerance, data dissemination, scalability, availability, and load balancing. Google developed the MapReduce programming paradigm to counter this problem, which served as the foundation for Apache's open-source Hadoop project.

Hadoop relies upon a particular file system designated as Hadoop Distributed File System (HDFS), analogous to Google's File-System (GFS). It splits the massive data into equally sized segments and then places them across multiple nodes in a Hadoop cluster [1]. As a result, Hadoop is now widely accepted as a data analytics model [2]. Hadoop's fundamental operating principle is that "moving computation to data is less expensive than moving data to computation." As a result, Hadoop tries to schedule tasks on local data nodes to minimise network traffic [3]. Task scheduling is critical in Hadoop because it significantly impacts the framework's computation time and, thus, its overall performance [4]. However, given the dynamic nature of the cloud environment, proposing an effective task scheduling strategy is a constant challenge. Nevertheless, only a few studies have analyzed the proposed techniques and their

*Author for correspondence

overall effect on the Hadoop framework's performance. Furthermore, no studies succinctly discussed the Hadoop scheduling problem and provide a research taxonomy for classifying existing scheduling techniques. Early efforts [5–7] to conduct a detailed study of Hadoop platform scheduling problems were limited in scope.

Additionally, they excluded a significant number of papers published during the study periods. Likewise, these studies discuss general depictions of the architectures and objectives of Hadoop schedulers without discussing their shortcomings. They also make no mention of future research directions for improving the effectiveness of existing scheduling strategies. Thus, the paper's primary contributions are as follows:

1. We systematically review Hadoop task scheduling strategies using a PRISMA-style paper selection approach [8] to identify and classify task scheduling problems.
2. Examine the various techniques and methodologies proposed for Hadoop's task scheduling over the last decade, which have piqued considerable interest in scientific and commercial communities. The paper examines the scheduling approaches for makespan, data control, resource usage/deadline, load balancing, fairness, energy efficiency, and failure recovery. Our search identified 82 relevant studies between 2008-2021, all of which came from high-quality conferences, journals, symposiums, and workshops.
3. Dividing the systematic review into three parts:
   - Identification of scheduling problems in the Hadoop system.
   - Discussion of various scheduling strategies in response to the problems identified in part-1.
   - Identify some future research directions for modifying current scheduling strategies.

2.Background

2.1Hadoop architecture

Five different daemons are included in Apache Hadoop architecture, and each of them runs in their JVMs; (i).DataNode, (ii).NameNode, (iii).Secondary NameNode, (iv).JobTracker, and (v).TaskTracker. These background processes (daemons) are typically classified into the HDFS Layer and the MapReduce Processing Layer. NameNode and DataNode (which store data and metadata) are located in the HDFS Layer, while JobTracker and TaskTracker (which execute and track jobs) are located in MapReduce Layer [3].

As illustrated in *Figure 1*, Master daemons, such as NameNode (for HDFS) and JobTracker (for MapReduce), perform on the Master Node. In contrast, Slave daemons, such as Data Node (for HDFS) and TaskTracker (for MapReduce), execute on the remaining cluster nodes, termed Slave Nodes [3].

HDFS implements a master / slave architecture by running both the Primary & Secondary NameNode daemons on the cluster's Master-Node and the DataNode daemons on the cluster's Slave-Nodes, as depicted in *Figure 1*.

Below are the three HDFS Layer daemons:

1) *NameNode:* The NameNode daemon stores data in a file called fsimage and manages file system metadata. NameNode stores this metadata in the main memory to facilitate read/write requests from clients. Apart from maintaining the HDFS's fitness, NameNode manages the partitioning of files into blocks (and then identifying the slave nodes that store these blocks). In addition, NameNode is responsible for I/O and memory-intensive tasks.
2) *Secondary NameNode*: Secondary NameNode goes through the file system at regular intervals, logs changes to a log file, and then apply those changes to the fsimage file, assisting the NameNode in starting up faster the next time.
3) *DataNode*: DataNode daemons (controlled by the NameNode) run on the cluster's slave nodes and are critical components of the HDFS environment's data storage. Additionally, the configuration specifies that data blocks stored within the DataNodes are replicated and distributed across the cluster's nodes to ensure availability and rapid computation.

2.2Hadoop-MapReduce

In the Hadoop cloud-based environment, Hadoop's MapReduce model has evolved into the dominant data processing paradigm. Map/reduce computing model processes data by performing multiple map/reduce operations on partitioned data across the Hadoop cluster's nodes [3]. As with HDFS, MapReduce implements the master/slave architecture by running JobTracker on the master node and TaskTracker on the slave nodes, as illustrated in *Figure 1*.

2.2.1MapReduce architecture

MapReduce is a collection of daemons that includes the following:

1) *JobTracker:* The JobTracker daemon is the master component of MR1 and is in charge of efficiently managing jobs and resources within

the Hadoop cluster. As illustrated in *Figure 1*, when clients submit their MR jobs to the JobTracker, the JobTracker employs an effective scheduling strategy to distribute the various tasks associated with the job across multiple TaskTrackers based on the location of a data-block as determined by the NameNode.

2) *TaskTracker:* The TaskTracker daemon is responsible for running Map & Reduce tasks on slave nodes. Each TaskTracker includes "task slots" determined by the node's capability and used to execute tasks. JobTracker uses the Heartbeat protocol to track the number of available "task slots" on the slave node and confirm that the TaskTracker is still alive. While each slave node has its own TaskTracker, each TaskTracker starts multiple JVMs for parallel MapReduce operations. As illustrated in *Figure 2*, the MapReduce programming framework consists of two stages: Map and Reduce. At the map stage, map tasks read the data input subsets and then generate intermediary outputs in <key-value> pairs using map functions. Initially, these <key-value> pairs are sorted and then shuffled. While the reduce operation combines data based on the same key generated in a map task, each reduce task does so independently.

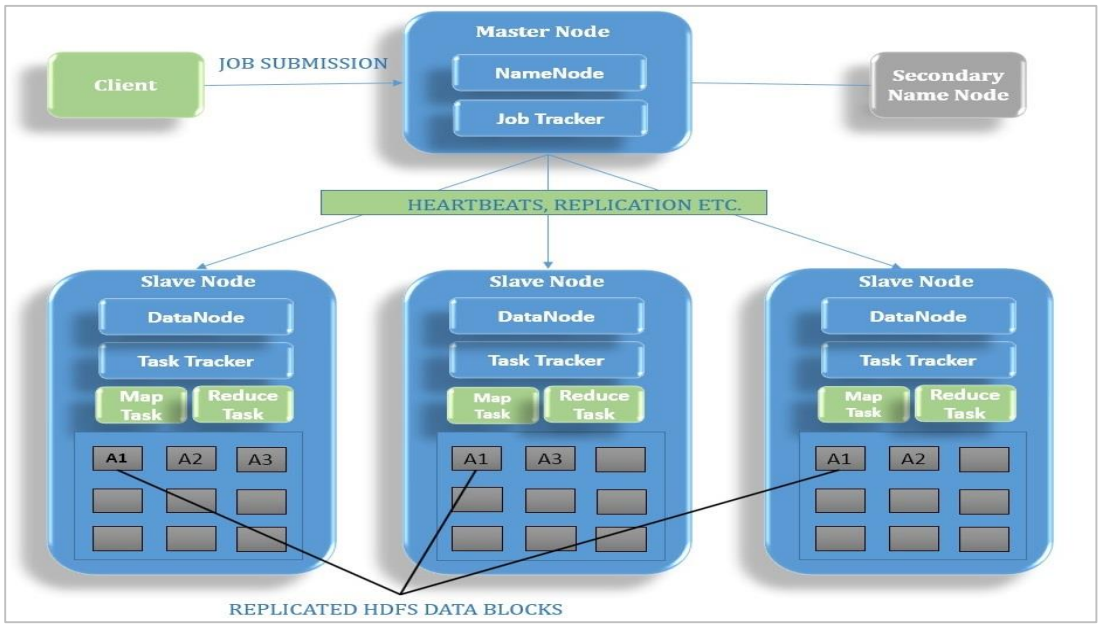

Figure 1 Hadoop framework architecture

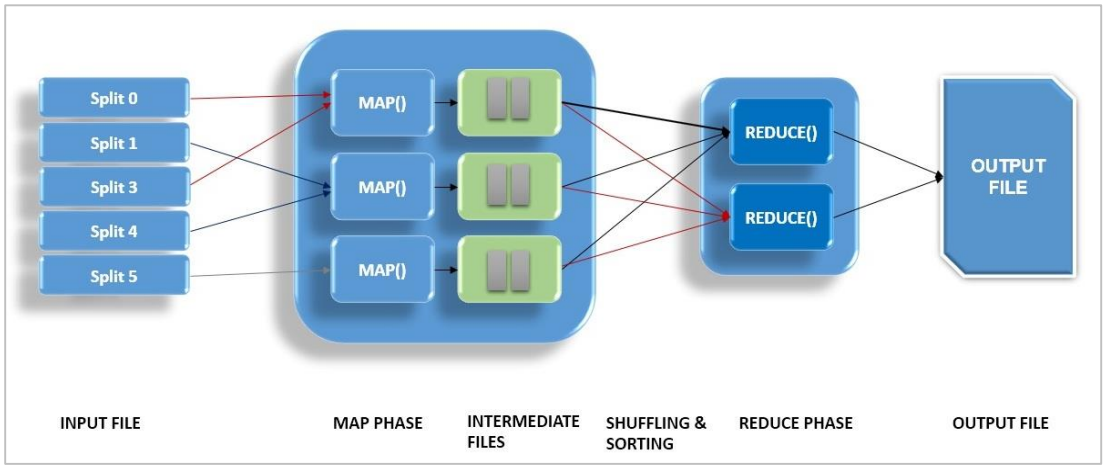

Figure 2 MapReduce model

2.3Classification of Hadoop-MapReduce schedulers

Apart from the default Hadoop-MapReduce schedulers, there are additional pluggable schedulers classified according to various factors such as run-time behaviour (Adaptive/Non-Adaptive), scheduling strategy (Static/ Dynamic), amount of available resources, time, scheduling entities, and so on [9].

2.3.1 Default Hadoop schedulers

The three default MapReduce schedulers offered in Hadoop v2.6 [3] are:

1) *First In First Out (FIFO) Scheduler* is Hadoop's default scheduling mechanism, which prioritises jobs, according to the order in which they are submitted.
2) *Fair Scheduler* handles the scheduling based on pool hierarchy. The goal is to assign each Hadoop user a fair division of cluster resources over job/task execution time. For example, if resources are underutilized, a given application may receive an extra share of its resources. Moreover, suppose a given Hadoop cluster is found to be overloaded. In that case, a given application is guaranteed to receive its fare-share of resources, as the Resource Manager uses resource pre-emption to recompense its resources back.
3) *Capacity Scheduler* handles queue hierarchy instead of pools, assigning resources among the separate queue organisations. In general, the administrator defines min and max number of resources that an application may use.

2.3.2 Based on the runtime behavior (Adaptive/ Non-Adaptive)

The two primary categories of run-time behaviour-centred scheduling algorithms are adaptive and non-adaptive. A scheduling approach is considered adaptive if it adapts to changes in its environment (including changes in the characteristics of the data, physical resources, and workload) while making scheduling decisions at run-time. On the other hand, a non-adaptive algorithm is unaware of environmental changes and thus performs tasks under previously defined policies [9, 10].

2.3.3 Based on a scheduling strategy (Static/ Dynamic)

Scheduling strategies can be static or dynamic. Static scheduling (such as FIFO) forwards jobs/tasks to processors before the start of program execution, based on pre-compile knowledge of processing resources and task execution time. Thus, static scheduling is generally concerned with reducing the execution time of currently running programs. In contrast, dynamic scheduling forwards jobs/tasks to processors during program execution with little prior knowledge of the resources required to complete the task [9, 11].

2.3.4 Based on availability of resources

This scheduling method is intended to be based on the requisites of the task (like CPU usage, disk storage, time, memory etc.). Therefore, it is primarily concerned with performance optimization by maximising resource utilisation [12].

2.3.5 Based on Time

This scheduling policy employs the user-specified deadline, which determines whether or not the job is completed within a specified timeframe [13].

2.3.6 Based on entities scheduled in MapReduce

In a shared environment, the scheduling scheme must take the hierarchical classification shown in *Figure 3* into account when making scheduling decisions [9]. This system of classification is divided into three distinct levels:

1) *User Level:* The scheduling approach, which operates at this level, first selects the user from the user queue and then schedules the job within the cluster.
2) *Job Level:* The scheduling approach functioning at this level focuses first on picking up a job from the queue and then scheduling tasks. There may be various job queues, or a single job queue, where all jobs go.
3) *Task Level:* The scheduling approach operating at this level schedules the job's map/reduce tasks in their respective slots, besides scheduling speculative tasks to assist with delayed map/reduce tasks.

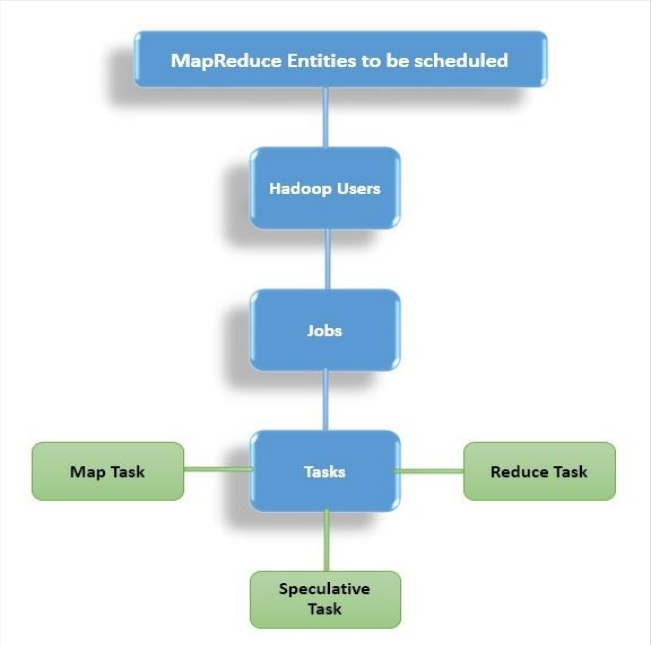

Figure 3 Scheduling entities

3.Methodology

We conducted a systematic review of the literature in accordance with the guidelines in [8], utilising the IEEE, ACM, Springer, and ScienceDirect database sources. We executed an electronic search and considered the following key terms: "Cloud scheduling", "Hadoop task scheduling", "Hadoop scheduler", and "Hadoop MapReduce". Between 2008 & 2021, we searched for published research literature on task scheduling in Hadoop. Then, we focused our research on a selected number of high-quality journals, conferences, workshops, and symposiums widely regarded as the essential resources in this field. Additionally, we consider the number of times the studies have been cited to determine their significance in this field. Other research works are discarded due to a deficiency in their quality (e.g., the research is a minor incremental improvement over previous studies).

The process of searching and selecting published studies from database sources is divided into four levels, as illustrated in *Figure 4.*

1. **Identification:** We identify the most relevant studies by conducting a title and abstract search of databases and other sources. Any study that is deemed irrelevant is omitted. If any confusion exists regarding the relevance of a study conducted at this level, the study is retained.
2. **Screening:** Along with duplicate studies, any studies deemed irrelevant by the selection criteria are also removed.
3. **Eligibility:** After the screening, full-text papers are carefully read and assessed for eligibility based on the motivation in the area of study.
4. **Inclusion:** Finally, we included the high-quality papers widely regarded as the most objective sources in the domain while rejecting all other papers due to their poor quality.

As shown in *Table 1*, the 82 papers included in the review were published by the following publishers: 51 papers (62.21%) in IEEE, 12 papers (14.64%) in ACM, 8 papers (9.72%) in Springer, and 11 papers (13.43%) in ScienceDirect.

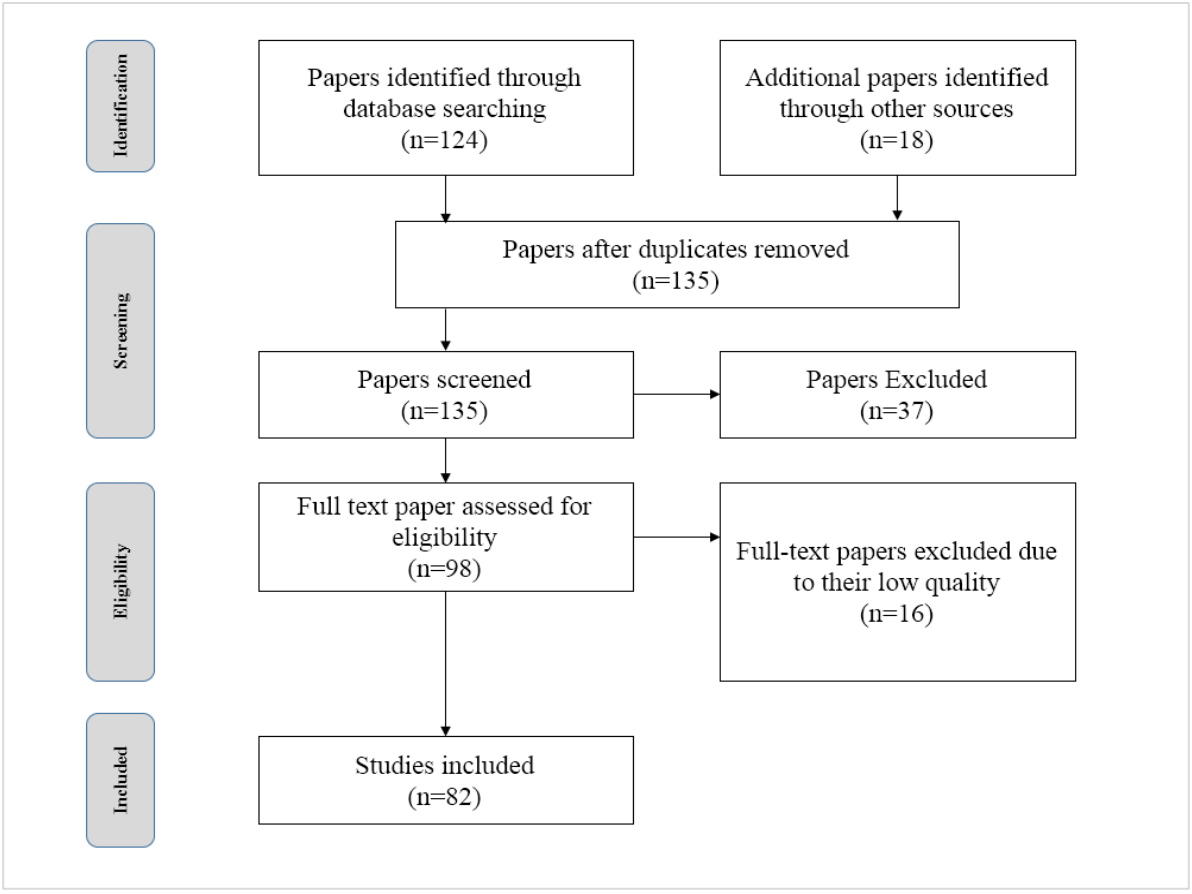

Figure 4 PRISMA flow diagram for searching and selecting papers

Table 1 Paper distribution concerning digital repositories

Publishers database	Number of papers chosen initially for analysis	Number of papers for final analysis	Percentage
IEEE	86	51	62.21
ACM	21	12	14.64
SPRINGER	16	8	9.72
SCIENCEDIRECT	19	11	13.43
Total	142	82	100.00

4.Literature review

The complete block diagram of the literature review is shown in *Figure 5*. First, we examine the numerous Hadoop task scheduling strategies that have garnered considerable attention from scientific and industrial communities in recent years. Additionally, we evaluate each strategy's design options against various performance metrics, including job's makespan, data control (locality, placement, and replication), deadline, resource usage, load balancing, fairness, energy efficiency, and failure recovery. Our analysis of the literature review is divided into 3-parts.

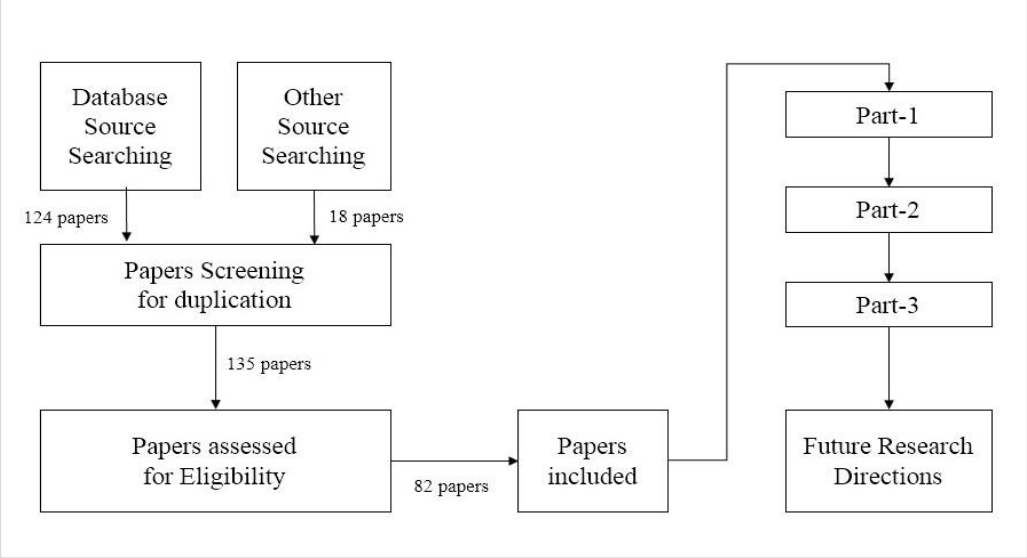

Figure 5 Complete block diagram of systematic review

1) Problems related to job/task schedule

This section discusses the critical problems associated with task scheduling in the Hadoop system. The problems are classified into the following core categories: deadline, makespan, data controlling policy (involving data locality, placement, and replication policy), resource usage, load balancing, fairness, energy efficiency, and failure recovery.

2) *Solution to job/task schedule problems*

This section discusses in detail the possible solutions to the problems discussed in Part-1. Again, the in-depth analysis of the literature helped us comprehensively picture the proposed solution's architecture, characteristics, and objectives. Finally, each category concludes with a summary of each proposed solution's central idea, strengths, and weaknesses.

3) *Directions for future research*

This section of the review identifies a specific amount of research that researchers can conduct to address the shortcomings of the studies discussed in Part 2 and thus provides a prospective road map for researchers interested in updating existing schedulers.

4.1Problems related to job/task schedule

After thoroughly examining the most critical sections of numerous research papers, we discover that the papers we reviewed fall into eight broad categories based on the problems they address. As such, it is appropriate to discuss the various groups in the following manner:

1) Makespan: Also known as scheduling, length, this is the amount of time required to complete a job/task from the time it is started to the time it is

finished. Therefore, in Hadoop-MapReduce, minimising the makespan of the task/job submitted is of greater importance.

2) Data Control: We claim that the data control can be further sub-divided into three more categories:

Locality control policy: In distributed computing platforms like Hadoop-MapReduce, computations are pushed towards the input data; because it cost less to push the computation process to the data than to push the large sets of data to the processing nodes. Following data locality, task scheduling is a significant issue that adversely affects the cluster's overall effectiveness. As a result, it is necessary to have a more efficient resource allocation strategy for tasks distributed across a given set of functional nodes. This way, the task's efficiency has been increased from the amount of non-local data tasks decreases.

Placement control: While data-locality control strategies can help achieve optimal run-time, moving the intermediary result from one cluster node to another and performing non-local tasks add overhead, which may eventually cause a cluster's overall performance to struggle. In general, the data placement strategy used determines the processing of scheduled tasks. This study discovered placement strategies that improve Hadoop cluster performance.

Replication control: Problems with data locality and placement are reducing system performance and data availability. Interestingly, some slow nodes may shut down, preventing data from being available and resulting in a temporary decrease in cluster performance. Numerous techniques have been recommended thus far for replicating data across multiple nodes (in terms of racks/clusters) to maximise performance. The number of available data replicas on the nodes must be increased to allow the scheduler to complete tasks by running slow task replicas on data replica nodes.

3) *Resource usage:* In general, resource allocation approaches aim to link cluster resources to different scheduled tasks, resulting in improved service delivery to end-users and lower cost for cloud service providers. The various cluster resources like CPU, Memory and others, are assigned using two basic MapReduce computing units: Map slot & Reduce slot. A slot refers to the worker's (e.g., TaskTracker) capacity, and each slot only needs to be linked to its respective task, i.e. map slots to map tasks & reduce slots to reduce tasks only. However, the slot assignment must be carried out when the map tasks finish processing before launching their analogous reduce tasks. Since the job's map & reduce tasks is tightly dependent on one another. Therefore, this interdependence can lead to an indefinite delay in task completion (i.e. starvation). Additionally, both the map and reduce tasks may require changing resource demands over time, making it challenging for the scheduler to efficiently utilise diverse computing resources. As a result, certain map/reduce operations may be slow, resulting in unexpected delays in the computation of map/reduce tasks.

4) *Deadline:* If the given algorithm is sufficiently competent to complete the assigned task within the specified time frame (time-limit constraint), Hadoop will proceed with the job/task execution. If this is not the case, the end-user must modify the task's time frame to complete it on time. As a result, it is critical to minimise the time required to complete the user-submitted task.

5) *Load balancing:* The Hadoop scheduler receives incoming job and task requests over a specified period, each with its own set of characteristics and resource requirements, resulting in uneven workloads. This imbalanced workload across the cluster nodes may result in significant delays and other issues affecting the system's reliability. As a result, load distribution across cluster nodes is critical for maximum efficiency. Additionally, workload balancing was completely reliant on an equal and fair distribution of available slots among scheduled tasks. Indeed, inefficient slot allocation is a primary cause of imbalanced workload distribution, as it results in straggling tasks competing for available slots. Thus, proper slot allocation results in a balanced workload.

6) *Fairness:* Fairness is critical for successfully implementing Hadoop, even more so when multiple jobs are running concurrently. Hadoop enables jobs to be partitioned into distinct Map/Reduce tasks that run concurrently and consume an appropriate amount of resources. There may also be differences in the fundamental design of nodes. However, jobs in Hadoop may have varying resource requirements over time. For example, some jobs in Hadoop may catch up on all of their allocated resources for extended periods, resulting in the temporary suspension of other jobs unless a fair plan for allocating slots to different tasks is in place. As a result, the scheduler must distribute slots evenly or fairly among the various tasks in the workload received to minimize the makespan of scheduled tasks.

7) Energy efficiency: Data-intensive applications operating on Hadoop clusters may have significant energy efficiency issues, as these applications consume additional energy to handle the workload

and conduct I/O operations. Additionally, the scheduler design has a substantial impact on the energy consumption of the Hadoop system. For instance, when tasks are executing on data containing nodes, a node may be assigned additional tasks, which consumes extra resources and thus increases the energy level. Besides that, various cluster nodes may experience failures, increasing the number of straggling tasks and thus increasing energy consumption. It is therefore critical to minimise the energy required to process the workload received.

8) *Failure recovery:* Cloud environments are typically heterogeneous, requiring a range of hardware and software configurations. As a result, a problem with software running on massive clusters that causes numerous errors (such as state corruption, memory leaks) could cause the cluster to crash or cause a temporary halt in workload processing. Even though Hadoop has some built-in fault-tolerant mechanisms, the failure of one task may result in significant deferrals due to the map and its corresponding reduce task's strong dependence. Furthermore, because resource contention occurs on an irregular basis, it may result in inefficient resource utilisation. As a result, numerous researchers have discussed this issue and proposed strategies for reducing failure rates and thus increasing the overall efficiency of a given Hadoop cluster.

4.2 Solution to job/task schedule problems

After careful observation of the contents of the reviewed papers, they can be assembled with their scope/objective. In the following sections, we will discuss each solution proposed for each category in greater detail. Finally, after each category, we present a summary of the proposed solutions' core idea, along with their strengths and weaknesses in *Table 2, 3, 4, 5, 6, 7 & 8*, respectively.

4.2.1Makespan-aware schedule

Minimizing job makespan on the MapReduce platform is another cloud computing challenge that has garnered considerable attention from both the industrial and research communities. A good percentage of algorithms have been introduced to address this problem, including [11], an efficient technique to reduce the makespan of a list of MapReduce jobs, called the Utilizing All Available Slots (UAAS) technique. The presented scheme fits the classic Johnson approach criterion and is thus ideal for minimising the makespan. In the real Hadoop environment, the author has also carried out rigorous tests on the optimality of the proposed scheme.

Zaharia et al. [14] introduced a fairly versatile heterogeneous scheduling scheme called Longest Approximate Time to End (LATE). The proposed approach uses the expected completion time to carry out the tasks most likely to affect the response time speculatively. As a result, LATE increases the response time/makespan time of jobs by a factor of two in Amazon's Elastic Compute Cloud (EC2) clusters with 200 VMS.

Tan et al. [15] proposed the Coupling scheduling algorithm, which consists of two critical components: (a) a wait-scheduling component optimised for reduce operations and (b) a random-peeking component optimised for map operations. Compared to the Fair scheduler, the proposed technique reduces the makespan of jobs with an average of 21.3%. However, specific jobs experience lengthy wait times because the other jobs running on the cluster consume the entire set of reduce slots.

Tian et al. [16] introduced a multi-MapReduce job scheduling algorithm with the goal of minimising the job's duration. The author leverages the standard Johnson model and introduces a new HScheduler technique that combines the classic Johnson and MapReduce functionality to significantly reduce job makespan time in on-line and off-line scheduling modes. Additionally, we discovered that, on average, HScheduler outperforms the most commonly used algorithm by 10.6 to 11.7% when scheduling a job in offline mode and by 8 to 10% when scheduling a job in online mode. Thus, the proposed algorithm can optimise throughput, response time, and power consumption in cloud computing.

Jiang et al. [17] proposed a MapReduce scheduling technique for reducing the time required to complete a job on parallel computers of varying speeds. The makespan minimisation schedules are classified into two types in this technique: off-line and on-line schedules. Off-line scheduling is primarily intended for non-preemptive reduce tasks, whereas on-line scheduling is intended for pre-emptive and non-preemptive reduce tasks. Pre-emptive and non-pre-emptive schedules outperform other algorithms by 10%-40% and 5%-20%, respectively, in terms of makespan.

Gandomi et al. [18] developed a new heuristic approach that significantly reduces the jobs' makespan. In this approach, the author first obtains information on implementing the Map/Reduce stages by reviewing the job description log. After that, the

author proposes a similar model for predicting running time using data mining and data analysis techniques. Finally, the job submission and monitoring method is used to calculate the job makespan. The experimental results show that the overall makespan has improved when compared to the un-optimized one.

Xu et al. [19] suggested a collaborative scheduler that takes into account the task-application relationship to minimize the makespan of the multi-task applications (having no deadline). To address this issue, the author introduced a network-based task selection strategy for multiple nodes that could be used to pick up the tasks from various applications. Thus, we transfer most tasks nearer to data based on a task selection strategy while balancing the number of nearby data tasks across a wide range of applications. As a result, the multi-application makespan is reduced after optimal task delegation to nodes is completed. Furthermore, compared to the basic scheduler, the makespan is reduced by 61.5%, and network traffic is reduced by more than 50%.

Table 2 Summary of makespan-aware strategies

Article	Core idea	Strengths	Weaknesses
[11]	Dynamic Job Ordering and Slot Configurations	• An efficient technique for reducing the makespan of a set of MapReduce jobs	• Failed to minimise the makespan of on-line MapReduce jobs
[14]	Longest Approximate Time to End	• Improves job response/makespan time	• It launches a speculative map task with no data locality.
[15]	Coupling Scheduling	• Minimizes the job makespan by making efficient use of resource slots	• There are still some jobs with lengthy waiting times.
[16]	Multiple MapReduce Job Scheduling	• Reduces job makepan for both online and offline scheduling modes	• It applies only to non-preemptive job cases.
[17]	Preemptive/ Non-preemptive off-line and on-line schedule	• Reduces the time it takes to complete a job on computing devices of varying speeds.	• Assumes parallelism for map tasks and nonparallelism for reduce tasks
[18]	MapReduce performance model based on benchmarking	• Reducing the time it takes to complete a job	• In heterogeneous clusters, there are only two modes of operation: slow and fast.
[19]	Multi-application Makespan minimisation	• Reduces the makespan of multi-task applications	• Limiting the number of applications and tasks in complex situations is difficult to set.

4.2.2 Data control-policy aware schedule

Data control is a critical issue that many researchers have considered in their work due to its significant impact on cluster performance. For example, the performance of the Hadoop scheduler is highly dependent on the data control policies that have been implemented. This section discusses the various methodologies researchers propose for optimising the data locality, placement, and replication within a given Hadoop cluster.

Locality-aware schedule

Although MapReduce is frequently used in systems that require frequent changes, it is less adaptable to system-wide data changes. To circumvent this constraint, researchers have proposed a variety of methodologies, including the following:

Zaharia et al. [20] proposed an effective delay scheduling strategy to resolve the inconsistency amid data-locality and fairness. We observe that delay scheduling achieves nearly optimal data locality in certain workloads and can increase efficiency by twofold while maintaining fairness. Moreover, in the sense of a large array of scheduling techniques, the flexibility of delay scheduling makes it more suitable beyond the fair share of resources.

Naik et al. [21] developed a novel scheduling scheme for assigning data blocks based on node processing power. The proposed scheme schedules and manages map/reduce tasks across the heterogeneous cluster's nodes according to their computational capabilities. The author uses the HiBench benchmarking tool to compare the proposed scheduler to other high-

performance schedulers. The experimental results demonstrate the proposed scheduler's effectiveness advantages in heterogeneous environments. Additionally, it improves efficiency compared to existing schedulers by reducing job execution time and increasing overall data locality.

Chen et al. [22] proposed a Locality-aware Scheduling Algorithm (LaSA) scheme to ensure efficient resource allocation and improved data locality. To begin, the author estimates the data-interference weight using a weight estimation model. The data-interference weight is determined by the volume of data on each node with available slots. The author then presented the LaSA algorithm to perform locality-aware resource allocation by executing a set of received map tasks on a machine having comparatively small weight & improved data-locality. Finally, the author evaluated the LaSA's performance using three clusters and 35 virtual machines. However, LaSA does not guarantee fair workload distribution among the cluster's nodes.

Althebyan et al. [23] proposed a new MapReduce job scheduler capable of dealing with massive amounts of data. The given algorithm is known as Multi-Threading Locality (MTL), and it is implemented using a multi-threading strategy with the cluster partitioned into modules. Each module is scheduled in synchronous time by an extraordinary thread. Each thread receives task updates when a job is forwarded to a particular cluster and searches within its modules for data locality. Once a local data-block is identified for a given task, the thread automatically notifies all other threads to terminate the search and begin searching for the next task. The algorithm outperforms FIFO by approximately 47%, Delay by approximately 31%, and Matchmaking by approximately 29%.

Xu and Cai [24] developed the DTSS algorithm to minimise the trade-off between fairness and data-locality during job scheduling. DTSS achieves this by dynamically splitting the task and immediately running the split task on a non-local data node to maximise fairness. Experimental analysis indicates that adjusting the task split percentage is likely to improve both performance and fairness. Additionally, the experimental results demonstrate that the DTSS algorithm improves the makespan of different users by 2% – 11% compared to delay scheduling under conditions where acquiring the cluster's local-data node is difficult. However, experiments have demonstrated that when jobs have easy access to their data blocks, DTSS is overly complicated.

To ensure data locality, Kao and Chen [25] presented a real-time scheduling framework for interactive Hadoop-MapReduce programs. The proposed framework includes a scheduler and a dispatcher. When the necessary resources become available, the scheduler is utilised to assign tasks immediately. Furthermore, the dispatcher considers blocking and data locality. Finally, the author evaluates the proposed scheduling framework's performance using a synthetic workload, demonstrating improved execution time and energy consumption. However, the applications running on the given Hadoop cluster may be the most impacted. When the given scheduling framework schedules a task on its local-data node, priority is ignored.

Dai and Bensaou [26] proposed DPMQS as a new Hadoop task scheduling technique. The proposed method i) encourages data locality and ii) automatically improves the precedence of jobs entering the completion phase of their Map operation, thereby minimising the time difference amid the start of reduce tasks and implementing the reduce function for these jobs. Experiments and simulations have demonstrated that DPMQS significantly reduces response time and that DPMQS is unaffected by cluster geometry changes.

Xie et al. [27] proposed a lightweight acceleration engine called Pandas to process tasks that are resilient to load fluctuations & skewness. Pandas with verifiable stochastic delay-optimality is a reliable locality-aware task-level accelerator. The author used the test-driven approach to check the algorithm's efficiency and found that Pandas increased the data computation by 11 times with hotspots and 2.4 times without hotspots over existing scheduling techniques.

While the preceding studies treat data locality as a single issue, subsequent research delves into it in detail to map and reduce operations.

Map task data-locality

Techniques for data prefetching are critical for optimising data locality and avoiding skewness in map operations. Numerous scientific studies have been conducted on this subject, including [28–30]. The authors of the preceding study suggested using both prefetching and pre-shuffling techniques to improve data localization when performing map operations. All three strategies look for the data block that corresponds to the map task. Additionally, each of the three methods is responsible for selecting a reducer (required for key-value pair shuffles) in order to

minimise network traffic. Despite this, the proposed methods optimise map tasks' data locality. However, they do not alleviate the load on the computation nodes, as the proposed methodologies do not take resource utilisation and load distribution into account.

To optimise the map task's data-locality, Zhang et al. [31] & Polo et al. [32] introduced scheduling policies based on dynamically collected information about the admitted workload. To finish the assigned task within the specified deadline, the authors claimed to manage and control the process of allocating resource slots among accepted jobs dynamically. Experiments confirm that the algorithms improve performance when dealing with large amounts of data transmitted via cluster channels. Additionally, authors may consider admitted tasks of various types (such as long, short, continuous, among others) & determine the scheduled task's remaining execution time to optimise the effectiveness of these scheduling policies.

He et al. [33] proposed a novel MapReduce scheduling scheme that optimises the data locality of the map operation. The proposed method is implemented in the original MapReduce run-time environment. The author analyses the proposed method on two measures, i.e. data-locality & estimated latency (including response-time), by comparing it with the regular FIFO scheduler and delay schedule solution, and found that the proposed scheme resulted in a higher data-locality rate and a lower latency (including processing time) for map tasks.

Ibrahim et al. [34] proposed a new map tasks algorithm called Maestro to increase MapReduce's overall efficiency. Maestro algorithm schedules Map-Tasks in 2 ways (i) It tries to fill the data nodes' available slots based on the percentage of mapped tasks and the replication scheme used to store the input data; (ii) the probability of scheduling a MapTask on a particular machine is defined by the number of copies of the input data. Experiments have shown that on a 100-node cluster, the algorithm provides about 95% of local map-task executions, excludes at least 80% of speculative map-task executions, and cuts their run-time by approximately 34%.

To maximise the data-locality of the map task and reduce the overall length of time it takes to finish the different jobs, Asahara et al. [35] introduced a LoadAtomizer scheduling strategy for MapReduce. LoadAtomizer delegates tasks to slightly loaded data nodes for data locality and evenly distributes the load in cluster nodes. The algorithm also avoids input/output congestion, which effectively reduces the wait time of the CPU I/O for a map operation. The recommended solution minimises the average run-time by up to 18.6%. However, the method is inefficient at minimising the map and reduce tasks' data-skewness because it seeks to balance the input/output load and optimise the admitted task's data locality.

Enhancing the MapReduce scheduler's performance is an ultimate focus, even more so in a heterogeneous virtualized cloud environment. Generally, a map task is allocated with an input split consisting of one or more data blocks. When more than one data block is given to a map task, non-local execution occurs. Traditionally, data blocks are duplicated across the network to a node where the map task is executing. This lengthens the duration of jobs and uses up a significant network channel capacity both within and between cloud data centre racks.

In light of this, Singh et al. [36] proposed a methodology called "improving data locality through ant colony optimization (IDLACO)" for minimising non-local computations and virtual network utilisation when input split is allocated to multiple data blocks. The findings indicate that IDLACO outperforms both the traditional fair scheduler and the holistic scheduler significantly.

Reduce task data-locality

Numerous algorithms are used to improve the data locality in ReduceTasks, including the following:

Hammoud and Sakr [37] introduced the Locality Aware ReduceTask Scheduling (LARTS) approach to ameliorate the reduce task's data locality. LARTS employs the procedure of early shuffling on mappers' intermediary results to minimalize the execution-time by stimulating the ReduceTask, once mappers carry out their characterized percentage (such as 5%). Consequently, it helps to avert the skewness existing in the data & to also minimalize the scheduling delay amid the map & reduce tasks. The method is based on the statistical recognition of the reducer's sweet spot, the duration in which the reducer identifies its entire partitions. However, the dynamic detection of these sweet spots could enhance the LARTS efficiency.

Hammoud et al. [38] presented a locality-aware & skew-aware ReduceTask scheduling approach known as Centre of Gravity Scheduling (CoGRS) to reduce the network congestion in the Hadoop cluster. CoGRS must schedule each ReduceTask at its centre of gravity

node determined after considering the partitioning skew to achieve data locality. The method was implemented in Hadoop-0.20.2 and evaluated on the internal cloud plus the Amazon EC2 cluster. Contrarily to the Hadoop MapReduce, experimental evidence reveals that the CoGRS algorithm decreases off-rack network flow in cloud and Amazon EC2 clusters by an average of 9.6% and 38.6%, respectively.

Tan et al. [39] proposed a stochastic optimised scheduling framework for increasing reducers' data locality while minimising the cost of transitional data fetching. However, because the proposed method utilises a limited number of map & reduce slots, cluster resources can be over or underutilised.

Arslan et al. [40] proposed a novel algorithm called Locality and Network-Aware Reduce Task Scheduling (LONARS), a modern scheduling technique that considers data locality and network traffic in scheduling a reduce task. Data-locality-aware scheduling ensures that tasks are scheduled close to their associated map tasks, minimising network traffic and data access delays. Moreover, Network-Aware seeks to distribute data across the entire network and eliminate the hotspots to minimise network bottlenecks. Research has shown that the proposed scheduling scheme can minimise the time required to "shuffle" the data up to 15%. Furthermore, the algorithm is helping to shorten the overall scheduling time by 3 to 4% relative to other scheduling algorithms. In addition, the proposed scheme reduces traffic on a switch and thereby reduces energy demand.

Selvitopi et al. [41] discusses static scheduling of map and reduce tasks to realise data locality and load balance. Data locality typically results in significant reductions in data transmission during the shuffle stage, and load balance typically resulting in rapid task execution during the map and reduce stages. The proposed model can improve performance by leveraging domain-specific knowledge via graph and hypergraph partitioning models. This information is collected during the preprocessing phase by inspecting the input data to ensure that the map and reduce tasks communicate effectively. MapReduce processes can schedule their tasks effectively by using the output of the models as a hash function. Two essential operational procedures were used to demonstrate the models: Sparse Matrix-Vector Multiplication (SpMV) and Sparse Matrix-Sparse Matrix Multiplication (SpGEMM). These are frequently used procedures in scientific computations and graph algorithms.

Placement-aware schedule

Numerous studies have been made to redress data-placement issues within the Hadoop platform & different optimized schemes have evolved, which finally enhance the scheduled task's data locality. For instance,

Xie et al. [42] developed an algorithm that distributes incoming data among nodes in proportion to their computing speed to support the data placement scheme for load distribution in a Hadoop cluster. Second, the author introduced the data re-distribution method, which effectively addresses the issue of data skewness by reorganising the parts of a file across cluster nodes. While the proposed approaches improve the locality and placement of data within a given Hadoop cluster, they do not include a policy for dealing with superfluous portions of a file or the ability to reorganise data intended for data-intensive programs dynamically.

Anjos et al. [43] proposed the MapReduce with Adapted Algorithms (MRA++) Mapreduce framework to manage large-sized heterogeneous clusters & process the data-intensive programs efficiently. To effectively maintain data-placement schemas, the framework assembles data dissemination information through the use of training tasks. MRA++ is composed of several modules. The data-division module is in charge of dividing the data among the admitted tasks; the scheduling module allocates available slots among the admitted tasks; the clustering module is in charge of task execution control, and the measuring-task module is in charge of data dissemination. Experimental results confirm that the MR++ framework enhances the cluster performance by 66.73\% and minimizes the cluster traffic at least by 70.0\% in networks having a 10Mbps data rate. However, additional delays are possible in the task's computing times. This is because they assemble the extra information & have to hold on to the processing until the measuring-task module schedules them on suitable nodes.

Replication-aware schedule

In order to address the issue of data replication in Hadoop, several studies have been presented so far, including:

Abad et al. [44] proposed Distributed Adaptive Data Replication (DARE), a reactive technique that copes to changes once data is populated at small-scale

intervals. The proposed technique uses stochastic sampling and competitive-ageing to determine the percentage of replicas to list and control their placement in a cluster. Experimental results demonstrate that the designed approach increases data locality by a factor of 85%, respectively, compared to the FIFO and Fair schedulers. However, DARE does not consider data with a low replication factor.

Jin et al. [45] proposed the Availability-Aware MapReduce Data Placement (ADAPT) technique. To minimalize the cluster traffic, the method enables optimum data transmission between cluster nodes in line with their availability without stepping up the data replica's number. However, while the proposed technique improves cluster traffic, it may result in increased disc usage.

John and Mirnalinee [46] proposed a novel dynamic data replication method with an intelligent water drop approach that considers parameters such as bandwidth, user access, storage space availability, and traffic. This paper makes two significant contributions. The first is a dynamic data replication method that replicates data depending on the number of accesses or popularity of the data. Second, the intelligent water drop algorithm assists in finding the optimum replica following the bandwidth, traffic, and several user requests. It is also used to handle cloud storage by determining which replicas should be removed based on the number of accesses and the replica's storage space. The author evaluated the proposed algorithm using Hadoop-like storage and analysed the access time for data storage and retrieval. As a result, there is a nearly 40% increase in available storage space. Additionally, the proposed algorithm has been compared to standard optimization algorithms such as Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) on replication strategy and discovered a 16% improvement over PSO and a 20% improvement over GA.

Table 3 Summary of data control-policy aware scheduling strategies

Article	Core Idea	Strengths	Weaknesses
		Locality-aware schedule	
[20]	Delay Scheduling	• Optimal data-locality combined with increased efficiency.	• Not feasible if the number of tasks exceeds the number of jobs or if slots per node are limited.
[21]	Locality-Aware Scheduling Scheme	• Improving job execution time and overall data-locality	• Only works in a small heterogeneous cluster
[22]	Locality-Aware Scheduling Scheme	• It ensures efficient resource allocation and data locality.	• Does not ensure a balanced workload on cluster nodes
[23]	Multithreading Locality Scheme	• Improved data locality	• Management overhead
[24]	Dynamic Task Split Scheduling Scheme	• Reduces the tradeoff amid fairness and data-locality.	• Not effective in situations where jobs can easily obtain their local-data nodes
[25]	Real-time Scheduling Framework	• Provides improved data locality • Reduces job execution time and energy usage	• Do not give preference when assigning tasks to local data nodes.
[26]	Dynamic priority multi-queue scheduling scheme	• Optimises data locality • Prioritizes the value of near-completed jobs	• In the event of system failure, all lower-priority tasks get lost.
[27]	Robust locality aware task-level Accelerator	• Promotes data locality and accelerates job processing	• Space overhead
[28]	Pre-fetching & pre-shuffling optimization Technique	• Improved data access during map operation • Minimises network overhead incurred during shuffling phase	• Performance is worst when there are fewer nodes than map tasks.
[29]	Locality-Based Partitioning Scheme	• Increases efficiency by executing more local map tasks.	• Longer job completion time
[30]	Map-Task Scheduling	• Improved data-locality	• Reduce tasks hamper efficient

Article	Core Idea	Strengths	Weaknesses
		Locality-aware schedule	
	Scheme	• Traffic-optimal	execution
[31]	Locality-Aware Map-Task Scheduling Scheme	• Optimises data-locality	• It makes no distinction between tasks received (such as long, short, continuous)
[32]	Adaptive scheduling scheme	• Increases data locality • Decreases overall network traffic rate • Meeting time completion goals	• A job needs more nodes to finish faster on non-accelerated maps.
[33]	Matchmaking Scheduling Technique	• Optimizes data-locality and processing time.	• A large cluster of small jobs may result in a low data locality rate
[34]	Replication Aware Scheduling Scheme	• Provides improved data-locality	• Lacks the ability to work in a dynamic environment on the public cloud.
[35]	Locality-aware & I/O load-aware scheduling scheme	• Provides improved data-locality • Reducing job completion-time	• Not capable of minimizing data-skewness
[36]	Enhancing data-locality through ant colony optimisation	• Reduced non-local executions and thus job latency	• No rack awareness
[37]	Locality-Aware Reduce-Task Scheduling scheme	• Optimizes data-locality while reducing network traffic	• No dynamic detection of sweet-spots
[38]	Locality-aware & skew-aware reduce task scheduling	• Optimizes data-locality while reducing network traffic	• It shows the worst completion time when the partition skew is negligible • Static detection of sweet spots
[39]	Stochastic optimization Scheduling Scheme	• Optimises the reduce task's data-locality • Reduces transitional data retrieval costs	• Under or overuse of cluster resources
[40]	Data-locality and Network Aware scheduler	• Optimises data locality & minimizes network traffic • Reduces energy demand	• Maximum Overhead • There is no discernible improvement in total job time
[41]	Locality-Aware and Load Balanced Static Task Scheduling	• Provides improved data locality and reduces data transfer during the shuffle phase.	• No application-specific or fault-tolerance consideration.
		Placement-aware schedule	
[42]	Data-Placement Scheduling Scheme	• It improves data placement & locality	• Duplicate file fragments are unmanageable.
[43]	Adaptive scheme for heterogeneous infrastructure	• It optimises data placement while also reducing network traffic.	• Extra information collection may cause delays.
		Replication-aware schedule	
[44]	Distributed data-replication methodology	• Ensures optimum data locality and replication	• Data with a low replication factor is ignored.
[45]	Availability aware Data Placement Scheme	• Reduces network traffic • Improve data placement	• Extra disk usage
[46]	Dynamic data replication based on an intelligent water drop method	• Increased efficiency while maintaining cloud accessibility, performance, and load balancing.	• Retrieving data from storage remains a difficult task.

4.2.3 Deadline/Resource-Aware Schedule

Even though map and reduce tasks are highly interdependent, existing Hadoop schedulers schedule them independently. According to studies, resource utilisation varies significantly between the two tasks, increasing the time required to complete the user-submitted task. Numerous studies have been published to address this issue, including [47–50], demonstrating how to coordinate the growth of map/reduce tasks, allocating resource slots to them, and scheduling them in a way that maximises the overall performance.

Polo et al. [47] developed a resource-aware MapReduce scheduler that maximises the utilisation of computing resources across multiple devices while meeting time completion goals. Current MapReduce scheduling approaches define a fixed number of slots to represent a Hadoop cluster's capacity. While this abstraction is well-suited for homogeneous systems, it does not account for the varying resource requirements of a multi-user environment. However, the proposed solution generates a job profile that can systematically manage the number of slots to optimise the cluster's resources. Additionally, this technique is motivated by the user-defined goal of completing the task on time.

He et al. [48] proposed a modern Real-Time MapReduce (RTMR) scheduling scheme that resolves the shortcomings of the current schedulers. Experimental results suggest that the proposed scheduler eliminates the jobs leading to the missed deadline and thus guarantees the promised deadline & efficient use of the cluster.

Liang et al. [49] proposed the Preemptive ReduceTask (Predoop) for Job execution accelerations algorithm. The proposed method preempts the reduce tasks in an idle state & allocates their resource slots to the scheduled map tasks to increase the processing power of the map phase. However, the reduce tasks often detain their resources while waiting for the intermediary data/results from map tasks. Experimental results illustrate that Predoop can minimize a job's execution time by at most 66.57\%. Furthermore, reduce tasks' pre-emption can delay the copy and merge phase, putting extra delays in job completion.

Pastorelli et al. [50] introduced a novel scheduling technique called Hadoop Fair Sojourn Protocol (HFSP), which instantaneously adapts to changes in the cluster's resources and workload while maximising resource efficiency and response time. The proposed method assigns resources to accepted tasks based on job size and processing information. To ensure that jobs with the highest priority receive the most resources, HFSP employs an ageing function. As a result, by utilising the pre-emption technique, HFSP can lower the processing time of admitted tasks while maintaining their failure rate. Additionally, MapTasks are delayed for a fixed period on non-local data nodes in HFSP to ensure that map tasks are executed locally. However, delaying MapTask execution for a more extended period increases the overall execution time and thus causes them to miss their deadlines.

There is an increasing need to provide predictable services to users in a competitive environment where job completion times are strictly enforced.

Cheng et al. [12] proposed Resource & Deadline-aware Scheduler (RDS), a scheduling algorithm minimises deadline misses by taking imminent resource availability into account. RDS determines the optimal resource allocation using a robust on-line Receding Horizon Control (RHC) approach. The author evaluates the RDS algorithm's effectiveness using several standard Hadoop benchmarks. The results imply that a larger task size implies more control over the map and reduce overhead. Additionally, the findings suggest that the processing of specific tasks is dependent on two factors, namely the block size and the percentage of reduce task.

In most cases, the block size outlines the estimated running time of a map operation. The longer a task takes to complete, the greater the overhead. This is because the RDS scheduling strategy does not preclude jobs from being allocated to the slot. For longer tasks, the RDS retains the slot for an extended period before releasing it. Additionally, the author conducted experiments to determine the scalability of the RDS algorithm by modifying the cluster resources. The findings indicate that several variables are bottlenecks for a larger cluster, indicating that additional research is necessary.

In order to meet the required Service Level Objective (SLO), Verma et al. [51] developed the Automatic Resource Inference and Allocation (ARIA) framework to define a clear deadline for job completion while utilising the cluster resources. It consists of three interdependent functions. First, a job description is generated that contains the critical performance characteristics of an application for a job that is constantly running on a new data set. Second, the Output Model, MapReduce-optimized, is produced, providing an estimate of the resources required to

complete the job within the specified timeframe. Finally, an SLO-based scheduler is developed that determines the job order and resource requirements for meeting the deadline.

The Hadoop System's default scheduling policies are committed to different constraints. Each standard scheduler used in Hadoop identifies a need as soon as it appears on the market.

Voicu et al. [52] propose a multi-objective and multi-constrained scheduling (MOMC) approach for various MapReduce tasks. The MOMC's introduction focuses on two objectives: (i) avoiding resource conflict, (ii) optimal cluster workload, and two constraints: (i) the deadline and (ii) the budget. The Scheduling Load Simulator (which is embedded in the Hadoop system and allows designers to spend as little time as possible testing) compares the various techniques based on a variety of metrics. This form of the scheduler can be used when there are cost and time constraints.

As latency-sensitive applications (i.e. applications requiring a rapid response) become more prevalent, the Hadoop system shows its inadequacy in completing jobs on time. Encouraged by this, Han et al. [53] proposed the CP-Scheduler algorithm, which utilises an optimizer to determine the optimal schedule for minimising the percentage of delayed jobs. Additionally, the CP-Scheduler algorithm adapts to different remote machines, which is not always the case with the Hadoop system. These characteristics enable it to outperform other Hadoop scheduling algorithms. Three phases comprise the proposed solution. First, the algorithm dynamically interprets the Constraint Model (CM) into a programming language for job and resource optimization. Then, the following Optimizer (OP) is used to obtain a list of job schedules. Finally, send the scheduling information used to calculate Hadoop efficiency to MRSG (Hadoop Simulator). The author implemented the Hadoop System's Earliest Deadline First (EDF) scheduling algorithm along with the CP-Scheduler. Following that, a comparative analysis is conducted using a computer equipped with a 3.3 GHz Intel i7 processor and 12GB RAM (with Linux4.4 as the operating system). The execution results are captured, indicating that the proposed solution reduces missed deadline jobs by 60% compared to EDF. On the contrary, the CP-Scheduler incurs a cost by taking 50% longer to solve and execute.

To implement the scheduler, Dong et al. [54] propose a two-tier scheduling scheme. First, the author proposes a sampling-based method known as Task Forward Scheduling (TFS) for determining the Map/Reduce task's processing time. In TFS, specific tasks are scheduled first to determine the processing time of subsequent tasks. Second, the Approximately Uniform Minimum Degree of parallelism (AUMD) model is being developed to construct a time-limited scheduler where real-time jobs are processed at a minimum level of parallelism at a given point. Thirdly, a two-tiered scheduler schedules both real-time and non-real-time jobs and assists in resource pre-emption.

MapReduce is the dominant data processing model in cloud computing, and it is used extensively in Hadoop. However, since Hadoop's default scheduling strategy, FIFO does not guarantee that a job will be completed within a specified timeframe. Therefore, research has focused on developing timeframe-based MapReduce algorithms using non-preemptive scheduling techniques. Nonetheless, preemptive scheduling is advantageous in comparison to non-preemptive scheduling.

Liu et al. [55] define the pre-emptive scheduling problem and propose a pre-emptive scheduling algorithm. The author introduced the first genuine preemptive scheduler to ensure that the job is completed within the specified time limit. The experimental results indicate that preemptive scheduling is more motivating and capable of managing jobs within a specified time frame than non-preemptive scheduling.

Cho et al. [56] proposed the following objectives in their paper: (i) to run an integrated MapReduce cluster that supports all types of jobs, regardless of their priority or deadline; (ii) to reduce the time required to complete high-priority jobs; and (iii) to improve the time required to complete low-priority jobs. To meet these objectives, the author proposed the Natjam algorithm. Natjam's initial purpose is to create an integrated scheduler that accommodates a wide variety of job priorities and deadlines and manages resources flexibly across a diverse set of jobs. However, the authors' proposed solution increased the running times of low-priority tasks, as they must repeatedly repeat the eviction process.

MapReduce jobs demand an immediate response from worker nodes in order to accomplish them on time. However, current scheduling techniques such as FIFO, fair, and capacity schedulers cannot ensure the timely response required to meet a prior deadline. As a result, Hadoop's response time and processing times for

various MapReduce jobs must be improved. Ullah et al. [57] proposed an effective preemptive deadline constraint scheduling technique based on the least slack-time and data-locality. The author conducts a task scheduling analysis on the Hadoop platform to realise task allocation and load balancing. As a result, a novel preemptive approach was proposed that takes the remaining processing time of the currently running job into account when determining pre-emption. The experimental results demonstrate that the proposed technique significantly decreases job processing time and queue waiting time compared to previous approaches.

Table 4 Summary of deadline/resource-aware scheduling strategies

Article	Core idea	Strengths	Weaknesses
[47]	Resource-aware scheduler for multi-job workloads	•Optimised resource utilisation	•Network bottlenecks
[48]	Real-time MapReduce Policy	•Promised deadline & optimum cluster-usage	•No job Fairness
[49]	Preemptive Reduce Task Scheme	•Minimize job execution time by effectively allocating resources	•Frequent pre-emption results in increased startup/step-down costs for map/reduce tasks.
[50]	Size-Based Schedule	•Reduces job execution time and improves resource utilisation	•Delaying map tasks will cause the response time to exceed the time allotted.
[12]	Resource/ Deadline-aware Scheduling Scheme	•Considers the relative availability of resources when addressing time constraints.	•Longer run-time increases overhead when allocating map-slots •In a larger cluster, multiple variables become bottlenecks.
[51]	Completion time Prediction model	•Jobs have enough resources to meet their SLO needs.	•Not designed to survive failures
[52]	Multi-objective & multi-constrained scheduling	•Avoids resource contention •Having optimum cluster workload •Useful under time & cost limitations	•Assuming no node fails prior to or after scheduling
[53]	Adaptive Scheduling Scheme for heterogeneous environment	•Reduces the chances of missing a job deadline	•Takes a long time to solve and execute
[54]	Deadline Aware/Resource Allocation model	•Real-time jobs are expected to be done on time •Ensures the quality of service for real-time and non-real-time jobs	•No consideration of failures (i.e., no backup tasks) •Based on homogeneous cluster •Based on the uniform key distribution of input data
[55]	Preemptive scheduling under a time-limit	•More effective at meeting deadlines than non-preemptive strategies	•Based on the machines' homogeneity
[56]	Priority & Deadline-Aware Scheduling Policy	•Performs best-effort scheduling based on deadlines •Priority inversion is not observed	•It lacks fairness, resulting in inconsistencies in job completion times.
[57]	Preemptive deadline constraint scheduler	•Significantly reduce job processing and queue wait times	•There are no run-time predictions for the Map or Reduce tasks •There is no dynamic data replication or distribution

4.2.4 Load balancing-aware schedule

Load distribution among cluster nodes is a significant concern. Thus, an efficient load distribution strategy can help in resource optimization and task distribution equity, resulting in improved cluster performance. For example, Mao et al. [58] proposed a load-driven Dynamic Slot Controller (DSC) technique that modifies the map and reduces task slots in response to the slave nodes' workload. When executing the 10GB volume of data, experimental evidence indicates that the proposed algorithm improves the system's CPU utilisation by 34% and response time by 17%. However, the proposed solution fails to address the task's data locality issue when the load is distributed across the computing nodes.

Teng et al. [59] proposed a Shortest Period Scheduler (SPS) scheduling scheme to ensure that a large percentage of jobs are completed before their projected timeframes. The proposed method allows job pre-emption and makes scheduling decisions on the fly while receiving the new workload plan. However, the technique should consider tasks that are heavily reliant on one another and the effect of data exchange between admitted tasks on their estimated deadlines and resource consumption.

Cheng et al. [60] utilizes the workflow set-up to construct a self-adaptive method for task scheduling. The author's solution comprises the Ant algorithm, which allows an effective load distribution amid the cluster resources by considering the tasks' characteristics. Consequently, the proposed technique shows an average improvement of 11% in the task's completion time. However, the author claimed that the method is more appropriate for large-sized jobs having several rounds in their map-task execution. Additionally, the author provides no details regarding the Ant algorithm's optimization to minimize its execution overhead.

The sharing demand of several Hadoop clusters amid the steady growth of many users increases the complexity of the systems. However, most Hadoop schedulers are neglecting this problem.

Rasooli and Down [61] proposed the design & implementation of a novel scheduling approach titled Classification and Optimization Based Scheduler for Heterogeneous Hadoop Systems (COSHH), which addresses the complexity at both the application and cluster stages. The method is sufficiently versatile for any inconsistencies in system parameters to emerge. COSHH is the fusion of FIFO scheduler, Fair scheduler & COSHH. Moreover, the COSHH employs the LP model to make the classification of received workload & to identify the efficient resource allocation scheme. The proposed hybrid solution improves the job's fairness, average completion-time, data-locality & scheduling time. The author defines the three usage configurations: FIFO scheduler for underloaded systems, Fair Scheduler for balanced systems, and COSHH for overloaded systems. However, the author does not specify a threshold that determines which type of scheduler must be used.

Tang et al. [62] proposed a workflow scheduling strategy characterizing jobs as Directed Acyclic Graph (DAG) and classifying them into CPU-bound & I/O-bound categories. The proposed algorithm prioritises jobs according to their types and allots them free resource slots while maintaining data locality and workload balance. Additionally, while the proposed algorithm is efficient for large jobs, it may have a detrimental effect on the performance of small jobs.

When MapReduce is used with unusual data, such as unequally distributed data, it may experience an unbalanced load and a lengthy run-time.

Li et al. [63] introduced the Map-Balance-Reduce (MBR) distributed programming model, which runs on an optimised Hadoop platform and is capable of cost-effectively processing data that is not evenly distributed. According to test results, the proposed programming model can increase efficiency by 9.7% to 17.6% when testing data with an uneven distribution of data in the actual MapReduce programming model.

Table 5 Summary of Load balancing-aware scheduling strategies

Article	Core Idea	Strengths	Weaknesses
[58]	Load driven scheduling scheme	• Optimizes CPU usage & response time	• Does not address the issue of data locality when the load is distributed.
[59]	Real-time scheduling scheme	• Maximises the percentage of jobs completed on time	• No consideration of dependency & data exchange.

Article	Core Idea	Strengths	Weaknesses
[60]	Adaptive-task tuning scheme	• Improves workload distribution across cluster resources • Improves task completion time	• Interferences between jobs can significantly reduce overall performance.
[61]	Classification & Optimisation based scheduling scheme	• Improves the average job completion time	• Scheduling complexity adds to scheduling time and management overhead.
[62]	MapReduce Optimized Job scheduling scheme	• Improved data-locality & load balancing	• It may have a detrimental effect on the efficiency of small jobs in the workplace.
[63]	MapReduce Load Balancing Programming Model	• Effectively balances the load when data is distributed unevenly	• Increased Computation

4.2.5 Fairness-Aware Schedule

Data-locality and fairness are mutually exclusive performance criteria within the Hadoop framework. A significant proportion of tasks must be located close to their computational data nodes to maximise data locality. However, resources must be provided for as long as tasks require in order to minimise delays and ensure fair task execution. Several studies, including [64–70], have specifically addressed this issue.

Ibrahim et al. [64] introduced a key partitioning strategy called Locality-Aware and Fairness-Aware (LEEN). According to the LEEN, the map and reduce mechanism is asynchronous. All possible intermediary keys are considered for separation based on the probability and dispersion of the expected data set after shuffling. The results indicate that LEEN is more effective at improving data locality and fairness. Additionally, it can minimise the total volume of shuffled data. LEEN appears to increase efficiency by 40% for the majority of workloads.

Nguyen et al. [65] proposed an approach called HyBrid Scheduling (HyBS). The proposed method processes CPU-intensive workloads by considering the admitted task's dynamic priority and data locality. In other words, the algorithm utilises dynamic priority information, expected execution times for map tasks, and user-defined SLVs to alleviate delays associated with tasks of varying lengths. The proposed approach ensures that map & reduce tasks receive a fair and equal share of the workload. Additionally, it optimises the map and reduce task’s wait time by determining the CPU-intensive data dependencies. To minimise job latency, the tasks obtained are assigned the Hadoop logger's dynamic priority.

Li et al. [66] discovered that in a multi-user Hadoop environment, processing-based schedulers such as Fair scheduler result in a decrease in efficiency in terms of execution time. As a result, the author proposed Hybrid Parallel pessimistic Fair Schedule Protocol (H-PFSP), which can finish the job before the fair scheduler and improves the average flow-time and the task's fairness. At periodical interims, H-PFSP estimates the remaining processing time of scheduled jobs and performs additive estimation updates based on completed tasks. The recommended solution can reduce the total execution time of the tasks, but it cannot guarantee that the clusters' resources are used efficiently.

Wang et al. [67] proposed the Fair and Efficient Slot Configuration and Scheduling for Hadoop Clusters (FRESH) method for comparing submitted tasks to free slots. The proposed solution shortens the makespan and ensures a fair and equal distribution of available resources. Each node in a Hadoop cluster has a fixed number of resource slots. However, the scheduler continuously accepts concurrent jobs that have slots configured differently. As a result, the author enhances the FRESH algorithm by incorporating a management technique that dynamically determines the optimal slot configuration. In other words, the proposed solution dynamically modifies the slot allocation in the map and reduce operations based on the slot number and the task specifications. Once the slot has finished its task, FRESH allocates the slot to another task. Even though the FRESH enhances the slot assignment & the fair distribution of resources amid the scheduled tasks, it cannot guarantee efficient memory usage.

Zhao et al. [68] introduced a K% - Fairness (multiple queue-based) task scheduler for allocating tasks to

nodes in a fair and location-aware fashion for a given workload. The proposed scheduler covers two aspects: broad planning and task scheduling. To establish a global standard for task scheduling, the author implements a task queue interface for each processing node. Under K% -Fairness, a dynamic assignment of tasks to different processing nodes is carried out using this strategy. Additionally, the author evaluates the scheduler for data locality and fairness, determining that it significantly improves data locality while maintaining fairness.

Cheng and Lo [69] proposed some enhancements to Fair Scheduler to increase its scheduling efficiency when subjected to stringent resource fairness regulations. The modified fair scheduler is adaptable and flexible, as it allows for changes to certain operational parameters, most notably job precedence and job latency, without affecting resource allocation. According to the evaluation results, the improved Fair Scheduler significantly reduces the job's computation time by more than 20% compared to the original Fair Scheduler.

Job scheduling is exceptionally challenging because it is critical for time optimization in big data. Hussain et al. [70] developed a more time-efficient and precise job scheduling algorithm than the current fair job scheduling technique to address this issue. The author minimised iteration in order to optimise the time cycle associated with fair scheduling. By varying the number of jobs, the author enhanced the existing algorithm and proposed a method for quantitatively reducing execution time. The proposed technique is computationally more efficient than the existing one, requiring fewer iterations and increasing time efficiency by an average of 26.719%.

Table 6 Summary of Fairness-aware scheduling strategies

Article	Core Idea	Strengths	Weaknesses
[64]	Locality-aware and Fairness aware Scheme	• It contributes to more effective data locality and fairness.	• Unimpressive output when task input value varies in size
[65]	Hybrid Scheduling scheme	• Significantly reduces the average response time	• SLV affects overall workload completion time
[66]	Multiple queue-based Scheduling scheme	• Significantly improves data-locality while maintaining fairness	• It is only applicable when a cluster contains a finite number of jobs.
[67]	Hybrid Parallel Fair Scheduling Scheme	• Reduces task execution time while increasing fairness	• Unable to ensure optimal utilisation of cluster resources
[68]	Fair/Effective slot configuration Scheduling Scheme	• Minimises the scheduled tasks' makespan • Ensures fair allocation of resources	• Inefficient use of memory
[69]	Improved Fair Scheduling Scheme	• Shortens computation time while improving data locality marginally.	• No focus on slot allocation/ data-locality in reduce tasks
[70]	Fair Scheduler to improve performance	• Reducing iteration to improve performance	• Requires more execution time

4.2.6Energy usage-aware schedule

In a Hadoop cluster, the type of nodes used, the amount of work performed by each node, and the number of underutilised nodes all impact the cluster's energy consumption. Numerous researchers, including [71–78], have examined this issue to develop an efficient energy-saving scheme.

Chen et al. [71] introduced the Berkeley Energy Efficient MapReduce (BEEMR) scheduling strategy to minimise the energy consumption of Hadoop jobs. Once the proposed technique receives the given workloads, it splits them into the time-sensitive & less time-sensitive sets. The earlier set is executed on the nodes with sufficient resources, whereas the latter is executed on the leftover nodes of the given cluster. As a result, BEEMR, under strict design constraints, can reduce the energy intake of the Hadoop cluster by 40.0

to 50.0%. However, even though the BEEMR achieves high energy savings on workloads involving significant actions, it cannot reduce the run-time of long-term jobs with a significantly low level of parallelism, even though all cluster resources are available. The Dynamic Voltage and Frequency Scaling (DVFS) strategy has been acclaimed as a highly successful method of achieving low power consumption by manipulating the CPU's speed during processing.

Wang et al. [72] attempt to reduce the amount of energy consumed by parallel, precedent-restricted tasks. Two distinct scheduling strategies for parallel tasks in DVFS-supported clusters were proposed for this purpose: PATC and PALS. The method discussed here analyses slack time for a given sample of noncritical jobs without expanding the schedule. Additionally, the author introduces a green SLA-based mechanism for reducing energy consumption. The algorithm is subjected to simulation analysis. Furthermore, the test results indicate that the proposed solution significantly reduces the power consumption of the DVFS enabled cluster.

Hadoop is widely regarded as the most advanced open-source framework for Big Data Analytics (BDA) processing, as it implements the MapReduce model. However, Hadoop's current scheduling policies are inefficient and result in high power consumption during sequential job processing.

Lu et al. [73] proposed a scheduling scheme based on GA to increase the efficiency of BDA. Additionally, the author presents an estimation module for implementing the scheduling model. This module forecasts the cluster's performance during job processing. Following an evaluation of the model, it was determined that the approach is feasible and produces acceptable results.

Mashayekhy et al. [74] modelled the energy-saving problem as an Integer Programming Problem (IPP) and proposed the Energy-aware MapReduce Scheduling Algorithms (EMRSA)-I and EMRSA-II Energy-MapReduce strategies. The proposed schemes consider the dependencies between the given set of map/reduce operations to ensure that tasks are completed on time. However, the proposed approach's primary objective is to decrease the map/reduce energy consumption. The author compares the performance of EMRSA-I and II to that of the default Hadoop schedulers using application programs such as PageRank, TeraSort, and K-means clustering and discovers that the proposed algorithm consumes approximately 40% less energy on average. Additionally, the proposed approach has the potential to reduce the duration of jobs. However, the author's assumption that the class of map/reduce tasks associated with the job must receive their resource slots prior to the job's execution may cause the job to be delayed in processing.

Wen [75] proposed a scheme for dynamically assigning host tasks to minimise total energy consumption. The method aims to have distinct thresholds to account for migration overhead, constant and intermittent cluster traffic, and compute power among Cloud Hosts (CHs). Based on the authors' suggested thresholds, the scheduler can adaptively designate tasks, satisfy constraints, and reduce energy consumption. The experimental evaluation confirms that setting a threshold between Cloud Hosts enables the cluster jobs to consume the least amount of energy possible and process efficiently. However, powering on or suspending the CHs adds overhead, disrupting Hadoop's network traffic, particularly when the likelihood of operations increases.

Energy consumption reduction efforts within the Hadoop framework are becoming increasingly important in the world of big data and green computing.

Pandey and Saini [76] defined the scheduling of map and reduce tasks within a single Hadoop Yet Another Resource Negotiator (YARN) job as an integer problem. The author proposed Energy-efficient MR Scheduling YARN (EMRSY), a heuristic methodology for creating suboptimal schedules in polynomial time because the formulated problem is NP-hard. Experiments demonstrate that the stated heuristic algorithm significantly reduces energy consumption for a variety of different benchmarks.

Energy costs are the primary consideration in cloud computing today. As a result, it is critical to implement energy-aware task scheduling methods.

Wang et al. [77] proposed a task scheduling strategy that incorporates deadlines and data locality in order to reduce energy usage in MapReduce clusters with a varying number of slots. Each heartbeat generates a fresh job sequence optimised for meeting deadline constraints and a fresh allocation of tasks and slots optimised for data locality. Jobs are prioritised based on the minimum number of required slots, determined by deadline constraints, available job slots, and

individual job processing times. Experimental results demonstrate that the stated heuristic consumes less energy than existing methodologies with a differential number of slots.

MapReduce job scheduling is a hot research topic, particularly in heterogeneous datacenters. Major challenges include massive energy usage and operational costs. Prior work has primarily focused on optimising the scheduling of a single task.

Chen and Liu [78] proposed a method for cooperatively optimising scheduling time, job costs, and energy consumption by using multiple MapReduce jobs as research objects. This is done by developing a strategy for scheduling MapReduce jobs in heterogeneous data centres that are energy and location efficient. Experiments demonstrate the energy-saving benefits of the proposed algorithm.

Table 7 Summary of Energy usage-aware scheduling strategies

Article	Core Idea	Strengths	Weaknesses
[71]	Energy Eefficient Scheduling Scheme	• Low energy consumption without affecting response time	• Unable to reduce the runtime of long-running jobs
[72]	Energy-Aware Scheduling Scheme	• Improves task execution time • Reduces the power consumption of the DVFS-assisted cluster	• Based on Static-task scheduling • Restricted to DVFS supported cluster
[73]	Genetic Algorithm	• It lowers the cost of energy efficiency.	• Simplified evaluation due to performance estimation module inadequacy
[74]	Energy-Aware Scheduling Scheme	• It reduces energy consumption	• Map tasks (including reduce tasks) must be resourced before jobs are executed.
[75]	Lowest-Energy Consumption Scheme	• Reduces energy usage to the minimum possible.	• There is a cost associated with powering and suspending Cloud Hosts (CHs), which disrupts network traffic.
[76]	Deadline-driven and energy-efficient MapReduce scheduler	• Significantly reduces the energy consumption	• It employs a static design and requires job profiling
[77]	Energy-aware task scheduling	• Reduces energy usage	• Not suitable for a varied workload
[78]	Locality and energy effective multi-job scheduling	• Improves data locality and energy efficiency	• No MapReduce execution process has been tested.

4.2.7Failure recovery-aware schedule

Even though Hadoop has built-in failure recovery mechanisms, task failures due to unexpected cloud events continue to occur. To ensure efficient data recovery in the event of a node failure, HDFS maintains multiple copies of data blocks across all Hadoop cluster nodes. Tasks are rescheduled to their original state on the additional nodes of the given Hadoop cluster in the event of a node failure. However, the proposed mechanisms incur high costs because the task must be re-executed from scratch, which significantly impacts the scheduler's performance. Several techniques, including [79–85], have addressed this issue.

Yuan and Wang [79] proposed an algorithm for dynamically identifying and backing up admitted tasks; as a result, the failed task can be restarted on additional nodes in the given cluster without affecting the intermediary data. However, while the proposed approach improves the fault tolerance of the Hadoop cluster, it lacks a mechanism for ensuring the accessibility of checkpoints and applied backups.

Chen et al. [80] proposed a method for correcting speculative task execution strategies. However, the proposed technique was determined to have a detrimental effect on the scheduling of other jobs, particularly batch jobs.

Yildiz et al. [81, 82] proposed Chronos, a failure-aware scheduling scheme that recursively recovers failed tasks via early support measures. To allocate cluster resources to recovered tasks, the proposed strategy makes use of the preemption technique. According to the experimental results, the proposed scheme significantly decreases job completion times by 55%. However, it is still heavily reliant on preemptive wait and kill methodologies, which waste resources and degrade performance in a given cluster.

Guo et al. [83] proposed a fault-tolerant scheduling model for High-Performance Computing clusters called FT-MRMPI. By leveraging existing Message-Passing Interface (MPI) schemes and a novel idea to minimise user-level failures. The proposed model may be used to mitigate the risks associated with job failures in a fault-tolerant system by utilising checkpoint/restart and detect/resume functionality. The checkpoint/restart functionality enables simple fault-tolerance by utilising the latest MPI schemes. Additionally, the detect/resume functionality enables automatic job resumption and reasonably effective job execution. The study demonstrates that the proposed scheme significantly overcomes failures and reduces job completion time by 39%.

Brahmwar et al. [84] introduced a method for determining the optimal nodes in a heterogeneous cluster for executing a speculative replica of stragglers. To dramatically reduce execution time, the suggested method, dubbed "Tolhit," uses information about the cluster's network and resource utilisation to determine the most appropriate choice for sluggish task execution. The proposed technique preserves the historical data for each cluster node. Additionally, each historical data record contains five values, two for the map stage (M1 and M2) and three for the reduce stage (R1, R2, and R3). The author conducts a series of experiments to evaluate the scheduling algorithm's efficiency at re-executing speculative tasks and then compares it to Hadoop's Fair-Scheduler (HFS). The simulation has been conducted on a 5-node heterogeneous cluster. The results indicate that "Tolhit" outperforms Fair-Scheduler by 27.0% in terms of execution time.

Zhu et al. [85] proposed a Fast-Recovering MapReduce (FAR-MR) strategy for big data applications to achieve resume-able fault tolerance. Using check-pointing and distributed storage technology, FAR-MR allows the recovered task to resume computation from the last recorded point, allowing faster recovery times. This paper describes the development and validation of the proposed FAR-MR on a 23-node computing cluster in the contexts of task and node failure recovery. Compared to Hadoop MapReduce, the performance evaluation revealed that FAR-MR can improve performance by up to 62% and 45%, respectively, in the cases of task and node failure recovery. Additionally, the greater the number of task failures, the greater the performance gain that FAR-MR can realize over MapReduce.

Table 8 Summary of Failure recovery-aware scheduling strategies

Article	Core Idea	Strengths	Weaknesses
[79]	Fault Tolerance Scheduling Scheme	• Improved fault-tolerance	• When no nodes fail, the system requires additional resources and time
[80]	Speculative Execution Scheduling Scheme	• Significantly improves the efficiency of speculative execution	• Scheduling overhead
[81, 82]	Failure-aware schedule	• Re-build the failed tasks • Improves the job completion-times	• Performance overhead
[83]	Fault-tolerant framework	• Overcomes failures and accelerates job completion	• Checkpoint overhead
[84]	Genetic Algorithm based clustering technique	• It determines the optimal node for speculative tasks. • It contributes to the overall execution -time improvement	• Overhead associated with the reclassification of historical data
[85]	Failure recover MapReduce	• Improve job computing performance during task	• The checkpoints' distributed nature

Article	Core Idea	Strengths	Weaknesses
		and node recovery	necessitates extra storage

5.Discussion

We discussed the various scheduling policies to resolve the problems like makespan, data control (data locality, placement & replication), deadline/resource usage, load balancing, fairness, energy efficiency and failure recovery. Moreover, we conducted a detailed comparative analysis of scheduling policies using specific performance metrics in *Table 9.* However, we noticed a significant number of papers lacking a formal outline of the problems discussed and their proposed solutions during the discussion. We also noted that the majority of the papers mentioned carried out empirical investigations, and little work suggests analytics models to resolve the various scheduling issues. Consequently, a motivating direction may be to extend empirical investigations by designing systematic models to enhance the efficiency of Hadoop Schedulers. Another challenge is the absence of a dataset that could be used to configure and define parameters of various benchmarks (such as TeraSort, WordCount, and others) to build a solution without any inclination. In addition, we noted that several research studies performed by researchers do not prove to be commercialized and part of the Apache Hadoop project.

"Different jobs have different performance criteria" is a precarious open challenge in the Hadoop job schedule. Many jobs require balancing these quality requirements, and to date, only a limited number of initiatives have been taken to satisfy these requirements. Given the current state of the art, there is a deficiency of strategies that strike the right balance among a variety of quality requirements. A complete list of abbreviations is shown in *Appendix I.*

Table 9 Comparative analysis of scheduling strategies based on different performance metrics

Article	Data locality	Information collection	Response time	Load balancing	Fault tolerance	Heterogeneity	Network traffic
Makepan							
[11]	○	○	●	○	○	○	○
[14]	○	○	●	○	○	●	○
[15]	●	○	●	○	○	○	●
[16]	○	○	●	○	○	○	○
[17]	○	○	●	●	○	○	○
[18]	○	●	●	○	○	●	○
[19]	●	○	●	○	○	●	●
Data Control							
[20]	●	○	●	○	○	○	○
[21]	●	○	●	○	○	●	●
[22]	●	○	○	○	○	○	●
[23]	●	○	○	○	○	○	○
[24]	●	○	●	○	●	○	○
[25]	●	○	○	○	○	○	●
[26]	●	○	●	○	○	●	○
[27]	●	○	○	●	○	○	○
[28]	●	●	○	●	○	○	●
[29]	●	○	○	●	○	○	○
[30]	●	○	○	●	○	●	●
[31]	●	○	○	○	○	○	●
[32]	●	●	○	○	○	●	●
[33]	●	○	●	○	○	○	○
[34]	●	○	●	●	●	●	○
[35]	●	●	○	○	○	○	●
[36]	●	●	○	○	○	●	●
[37]	●	○	○	○	○	○	●
[38]	●	○	●	○	○	●	●
[39]	●	○	○	○	○	●	●

Article	Data locality	Information collection	Response time	Load balancing	Fault tolerance	Heterogeneity	Network traffic
[40]	●	○	○	○	○	●	●
[41]	●	○	○	●	○	○	○
[42]	○	○	○	●	○	●	○
[43]	○	○	○	●	○	●	○
[44]	●	○	○	○	○	○	●
[45]	●	○	○	○	○	○	●
[46]	●	●	●	●	○	○	○
Deadline/Resource Usage							
[47]	○	●	●	○	○	○	○
[48]	○	●	○	○	○	●	○
[49]	●	○	○	○	○	○	○
[50]	●	●	●	○	○	○	○
[12]	○	●	○	○	○	○	○
[51]	○	●	○	○	○	●	○
[52]	○	●	●	●	○	○	○
[53]	○	●	○	○	○	●	○
[54]	○	●	○	○	○	○	○
[55]	○	●	●	○	○	○	○
[56]	●	○	●	○	○	○	○
[57]	●	○	●	●	○	●	○
Load-Balancing							
[58]	○	●	○	●	○	●	○
[59]	●	○	●	●	○	○	○
[60]	○	●	○	●	○	●	○
[61]	●	●	○	●	○	●	○
[62]	●	○	○	●	○	●	○
[63]	○	●	○	●	○	○	○
Fairness							
[64]	●	○	●	●	○	○	○
[65]	●	●	●	○	○	○	○
[66]	○	●	●	○	○	○	○
[67]	○	●	●	○	○	○	○
[68]	●	○	●	○	○	○	○
[69]	○	●	●	○	○	○	○
[70]	○	●	●	○	○	○	○
Energy-Efficiency							
[71]	●	○	●	○	○	○	○
[72]	○	●	○	○	○	●	○
[73]	○	○	●	●	○	●	○
[74]	○	●	○	○	○	○	○
[75]	○	○	●	●	○	●	○
[76]	○	○	●	○	○	●	○
[77]	●	●	●	○	○	●	○
[78]	●	○	●	○	○	●	●
Failure-Recovery							
[79]	○	○	○	○	●	○	○
[80]	●	○	○	○	●	●	○
[81, 82]	●	●	○	○	●	○	○
[83]	○	○	○	●	●	○	○
[84]	○	●	○	○	●	●	○
[85]	○	●	○	○	●	○	○

6.Directions for future research

Even though there is a substantial body of literature on current scheduling strategies in the Hadoop environment, several critical aspects of this field remain unexplored. Based on the literature review findings, we attempt to suggest some future research directions. These directions serve as a prospective road map for researchers.

Job Makespan

While the existing literature contains a limited number of scheduling strategies for optimising the makespan of a group of MapReduce jobs, there is still much room to improve MapReduce's efficiency by optimising the jobs' makespan. In particular, we found that:

- The earlier scheduling rule shifting computation nearer to the data struggles while minimising the multi-task application's makespan. However, the number of tasks closest to the data is asymmetrical across applications. It is not specified to which the task relates, thus creating an ample space amid the makespan of various applications. Effective scheduling methods should therefore be built to optimise the makespan of multi-task applications.

Data control

- The suggested solutions responsible for data placement & replication over cluster nodes are not designed according to cluster workload. Thus, in order to optimise data localization for assigned tasks, an accurate workload analysis is required for both data placement and replication issues.
- Additionally, spreading multiple local tasks across cluster nodes creates an unbalanced workload. Hierarchical scheduling may be a solution, as it schedules data in layers, with one layer focusing on data locality and the other on load distribution. To cooperate, the layers' local and global data must be shared.
- Furthermore, we discovered that the proposed solutions could not promise a high level of data localization for large jobs because strong data transport is required. As a result, effective scheduling methods for various types of jobs must be developed.

Deadline/resource usage

- While different policies govern the assignment of resource slots to map/reduce tasks, several approaches employ a random selection of map/reduce tasks that satisfy slot requirements. However, we discovered that incorporating data-locality into the scheduling of a given map/reduce task is critical for optimising its processing, thus avoiding data-skewness issues and reducing the task's computation time.
- Additionally, in order to meet the deadline, an efficient algorithm that manages tasks pre-emption activity in a way that results in minimal processing overhead and avoids task starvation is desired.
- Furthermore, analysing factors that contribute to resource efficiency (such as slot accessibility, node count, queue status, and expected workload) can assist the scheduler in making appropriate decisions while allocating the resources in a changing environment.

Load-balancing

- A solution that estimates a task’s processing time based on its growth rate may be introduced to improve the scheduler's efficiency by redistributing the load among computing nodes. However, state-of-the-art techniques estimate the computation time using a simple algorithm, which results in an imbalanced workload and thus has limited application in heterogeneous clusters.
- Additionally, the predictive techniques' effectiveness to determine the permitted workload significantly impacts the cluster's load balancing. As a result, effective models are required to ascertain the properties of the approved workload. Considering these features, the scheduler can be configured to make effective decisions while allocating resource slots to MapReduce tasks with guaranteed data locality.
- Moreover, a model must be developed to modify the scheduler's decision-making process by considering the dependencies between tasks in a given workload, reducing the uncertainty associated with sharing intermediate results.

Fairness

- It is critical to efficiently allocate the available slots between the supplied tasks to avoid the starvation problem. However, we discovered that when slots are assigned, the job addressing the subject fails to account for variations in the map and reduce tasks.
- Additionally, it will significantly improve Hadoop's scheduler performance by developing a constraint solver that takes into account multiple objectives such as data locality, fairness, and resource utilisation. However, while implementing the heuristic approach may solve this problem, it may result in processing overhead.
- Besides that, we noted that more studies ignore the fairness of continuous jobs that require longer-term free resource slots than non-continuous ones. Thus,

developing a method for determining the total number of slots needed to successfully complete continuous and non-continuous jobs while providing a reasonable share of available slots may be advantageous.

- Finally, limiting the amount of shared data used by reliant jobs is critical to minimise the processing overhead associated with intermediate data transfers.

Energy-efficiency

Energy consumption in data centres has increased exponentially, to the point where the server's energy costs now exceed its total capital expenditure. According to the current analysis, existing scheduling algorithms take into account critical factors such as data locality, fairness, resource utilisation, and job deadlines. Still, they have failed to address the energy consumption challenges adequately. As a result, there is an increasing need to improve Hadoop-MapReduce's energy efficiency. We noted the following during our evaluation of the current work:

- Current energy-efficiency optimization research is focused on "workload concentration" & "power-off idle node strategy." Nevertheless, in big data processing models, data that resides on nodes must always be in “ON” state & is very costly to turn off the nodes due to the following reasons: Initially, it is rarely possible for a given cluster node to be idle, as it must provide the data service constantly; Next, it is necessary to store the intermediate results of a given task on a local disc for further processing; Finally, synchronisation between Map & Reduce tasks in the Hadoop environment often requires cluster nodes to be in "ON" state; otherwise, all other nodes will be blocked due of waiting. Therefore, in big-data processing environments like Hadoop, "ON-OFF" state algorithms are difficult to use. As a consequence, the emphasis needs to be on computational resources. If the task's resources are entirely utilised, the energy-efficiency of the task is maximised; and if the energy-efficiency of the task on a node is optimal, the energy-efficiency of that node is optimal.
- To minimise the energy consumption, we should ensure that the appropriate percentage of resources is assigned to each running task, thereby minimising idle resources. This can only be accomplished by striking a balance between "efficiency" and "energy consumption" for a given workload. Furthermore, researchers may run experiments with varying workloads to illustrate the trade-off between efficiency and energy consumption in terms of previously defined parameters. In the future, predictive models based on these known parameters can be developed, resulting in increased energy efficiency.

Failure recovery

In the Hadoop environment, task failure is critical, as it can significantly reduce performance. On the other hand, modern techniques enable the re-execution of failed tasks with little or no information. As a result, an information gap exists between Hadoop components. During our review of the current work, we discovered the following:

- Adaptive methods that gather information about task transitions in the Hadoop environment and adjust scheduling choices accordingly will reduce potential task failure-related issues.
- In addition, redundant (speculative) tasks continue to experience multiple failures and resource wastage due to inaccurate approximations of task growth or resource accessibility. Therefore, it is necessary to monitor the effectiveness of environmental variables in terms of the time required to initiate a task and the number of times a straggler's task can be implemented.
- Finally, distinguishing between the map/reduce task failure is crucial since it strongly impacts task execution.

7.Conclusion

Decision-making on task scheduling is regarded as one of the most significant challenges for high-performance acquisition in the Hadoop Big Data analysis. The paper provided a systematic discussion of various scheduling strategies for Hadoop. In this study, we examined 82 papers to see which factors have the greatest impact on the efficiency of stated schedulers. We discuss these factors in greater detail and the challenges and issues they entail, such as resource consumption, total makespan, and energy efficiency. Furthermore, we identified and analysed existing scheduling strategies in the literature, such as dynamic, constrained, multi-objective and adaptive systems that include both nature-inspired & non-nature inspired strategies, as well as their associated strengths and weaknesses. We classified scheduling problems as makespan, data control, deadline, resource usage, load balancing, fairness, energy efficiency and failure recovery. We go into great detail about the methodologies that have been put in place to address these problems. Eventually, we identified potential future research areas that could be of interest to researchers in job/task scheduling in the Hadoop system.

8.Benefits & limitations

There are currently no studies that succinctly discuss the Hadoop scheduling problem and provide a research taxonomy for classifying existing scheduling techniques. On the other hand, this systematic study examines the various dynamic, constrained, and adaptive scheduling methods and their primary motivations, including makespan, data control, deadline, resource utilisation, load balancing, fairness, energy efficiency, and failure recovery. Additionally, some unresolved issues are discussed, and potential future directions for modifying existing studies are identified. Although the search method was designed to encompass as many studies as possible on the subject, certain studies may have been excluded due to their publication date. This review focused exclusively on publications published between 2008 and 2021. The studies may have been excluded during this period due to a lack of clarity in the title, keywords, and abstract readings. Finally, we intend to expand on the qualitative analysis conducted thus far and provide experts with additional recommendations that can guide future cloud scheduling research.

Acknowledgment

None.

Conflicts of interest

The authors have no conflicts of interest to declare.

References

[1] https://developer.ibm.com/articles/os-hadoop-scheduling/. Accessed 01 June 2021.
[2] Maheshwari A, Bhardwaj A, Chandrasekaran K. Hadoop task scheduling-Improving algorithms using tabular approach. In fifth international conference on communication systems and network technologies 2015 (pp. 1034-8). IEEE.
[3] http://hadoop.apache.org/. Accessed 01 June 2021.
[4] Anagnostopoulos I, Zeadally S, Exposito E. Handling big data: research challenges and future directions. The Journal of Supercomputing. 2016; 72(4):1494-516.
[5] Singh N, Agrawal S. A review of research on MapReduce scheduling algorithms in Hadoop. In international conference on computing, communication & automation 2015 (pp. 637-42). IEEE.
[6] Rao BT, Reddy LS. Survey on improved scheduling in Hadoop MapReduce in cloud environments. arXiv preprint arXiv:1207.0780. 2012.
[7] Patil S, Deshmukh S. Survey on task assignment techniques in Hadoop. International Journal of Computer Applications. 2012; 59(14):15-18.
[8] Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ. 2021:1-9.
[9] Kalia K, Gupta N. A Review on job scheduling for hadoop mapreduce. In international conference on next generation computing and information systems 2017 (pp. 75-9). IEEE.
[10] Rasooli A, Down DG. An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. In proceedings of the conference of the center for advanced studies on collaborative research 2011 (pp. 30-44). IBM Corp.
[11] Tian W, Luo G, Tian L, Chen A. On dynamic job ordering and slot configurations for minimizing the makespan of multiple MapReduce jobs. arXiv preprint arXiv:1604.04471. 2016.
[12] Cheng D, Zhou X, Xu Y, Liu L, Jiang C. Deadline-aware MapReduce job scheduling with dynamic resource availability. IEEE Transactions on Parallel and Distributed Systems. 2018; 30(4):814-26.
[13] Kc K, Anyanwu K. Scheduling hadoop jobs to meet deadlines. In second international conference on cloud computing technology and science 2010 (pp. 388-92). IEEE.
[14] Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I. Improving MapReduce performance in heterogeneous environments. In USENIX symposium on operating systems design and implementation 2008(pp.29-42).
[15] Tan J, Meng X, Zhang L. Coupling task progress for mapreduce resource-aware scheduling. In proceedings IEEE INFOCOM 2013 (pp. 1618-26). IEEE.
[16] Tian W, Li G, Yang W, Buyya R. HScheduler: an optimal approach to minimize the makespan of multiple MapReduce jobs. The Journal of Supercomputing. 2016; 72(6):2376-93.
[17] Jiang Y, Zhu Y, Wu W, Li D. Makespan minimization for MapReduce systems with different servers. Future Generation Computer Systems. 2017; 67:13-21.
[18] Gandomi A, Movaghar A, Reshadi M, Khademzadeh A. Designing a MapReduce performance model in distributed heterogeneous platforms based on benchmarking approach. The Journal of Supercomputing. 2020:1-27.
[19] Xu J, Wang J, Qi Q, Liao J, Sun H, Han Z, Li T. Network-aware task selection to reduce multi-application makespan in cloud. Journal of Network and Computer Applications. 2021;176(15).
[20] Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In proceedings of the European conference on computer systems 2010(pp. 265-78).
[21] Naik NS, Negi A, BR TB, Anitha R. A data locality based scheduler to enhance MapReduce performance in heterogeneous environments. Future Generation Computer Systems. 2019; 90:423-34.
[22] Chen TY, Wei HW, Wei MF, Chen YJ, Hsu TS, Shih WK. LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In international conference on collaboration technologies and systems 2013 (pp. 342-6). IEEE.

[23] Althebyan Q, ALQudah O, Jararweh Y, Yaseen Q. Multi-threading based map reduce tasks scheduling. In international conference on information and communication systems 2014 (pp. 1-6). IEEE.
[24] Xu Y, Cai W. Hadoop job scheduling with dynamic task splitting. In international conference on cloud computing research and innovation 2015 (pp. 120-9). IEEE.
[25] Kao YC, Chen YS. Data-locality-aware mapreduce real-time scheduling framework. Journal of Systems and Software. 2016; 112:65-77.
[26] Dai X, Bensaou B. Scheduling for response time in Hadoop MapReduce. In international conference on communications 2016 (pp. 1-6). IEEE.
[27] Xie Q, Pundir M, Lu Y, Abad CL, Campbell RH. Pandas: robust locality-aware scheduling with stochastic delay optimality. IEEE/ACM Transactions on Networking. 2016; 25(2):662-75.
[28] Seo S, Jang I, Woo K, Kim I, Kim JS, Maeng S. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment. In international conference on cluster computing and workshops 2009 (pp. 1-8). IEEE.
[29] Wang C, Wu Q, Tan Y, Wang W, Wu Q. Locality based data partitioning in MapReduce. In international conference on computational science and engineering 2013 (pp. 1310-7). IEEE.
[30] Wang W, Zhu K, Ying L, Tan J, Zhang L. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. IEEE/ACM Transactions on Networking. 2014; 24(1):190-203.
[31] Zhang X, Zhong Z, Feng S, Tu B, Fan J. Improving data locality of MapReduce by scheduling in homogeneous computing environments. In international symposium on parallel and distributed processing with applications 2011 (pp. 120-6). IEEE.
[32] Polo J, Becerra Y, Carrera D, Steinder M, Whalley I, Torres J, et al. Deadline-based MapReduce workload management. IEEE Transactions on Network and Service Management. 2013; 10(2):231-44.
[33] He C, Lu Y, Swanson D. Matchmaking: A new MapReduce scheduling technique. In IEEE third international conference on cloud computing technology and science 2011 (pp. 40-7). IEEE.
[34] Ibrahim S, Jin H, Lu L, He B, Antoniu G, Wu S. Maestro: Replica-aware map scheduling for mapreduce. In IEEE/ACM international symposium on cluster, cloud and grid computing 2012 (pp. 435-42). IEEE.
[35] Asahara M, Nakadai S, Araki T. LoadAtomizer: a locality and I/O load aware task scheduler for MapReduce. In international conference on cloud computing technology and science proceedings 2012 (pp. 317-24). IEEE.
[36] Singh G, Sharma A, Jeyaraj R, Paul A. Handling non-local executions to improve MapReduce performance using ant colony optimization. IEEE Access. 2021; 9:96176-88.
[37] Hammoud M, Sakr MF. Locality-aware reduce task scheduling for MapReduce. In IEEE third international conference on cloud computing technology and science 2011 (pp. 570-6). IEEE.
[38] Hammoud M, Rehman MS, Sakr MF. Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In fifth international conference on cloud computing 2012 (pp. 49-58). IEEE.
[39] Tan J, Meng S, Meng X, Zhang L. Improving reducetask data locality for sequential mapreduce jobs. In Proceedings IEEE INFOCOM 2013 (pp. 1627-35). IEEE.
[40] Arslan E, Shekhar M, Kosar T. Locality and network-aware reduce task scheduling for data-intensive applications. In international workshop on data-intensive computing in the clouds 2014 (pp. 17-24). IEEE.
[41] Selvitopi O, Demirci GV, Turk A, Aykanat C. Locality-aware and load-balanced static task scheduling for MapReduce. Future Generation Computer Systems. 2019; 90:49-61.
[42] Xie J, Meng F, Wang H, Pan H, Cheng J, Qin X. Research on scheduling scheme for Hadoop clusters. Procedia computer science. 2013; 18:2468-71.
[43] Anjos JC, Carrera I, Kolberg W, Tibola AL, Arantes LB, Geyer CR. MRA++: Scheduling and data placement on MapReduce for heterogeneous environments. Future Generation Computer Systems. 2015; 42:22-35.
[44] Abad CL, Lu Y, Campbell RH. DARE: Adaptive data replication for efficient cluster scheduling. In international conference on cluster computing 2011 (pp. 159-68). IEEE.
[45] Jin H, Yang X, Sun XH, Raicu I. Adapt: Availability-aware MapReduce data placement for non-dedicated distributed computing. In international conference on distributed computing systems 2012 (pp. 516-25). IEEE.
[46] John SN, Mirnalinee TT. A novel dynamic data replication strategy to improve access efficiency of cloud storage. Information Systems and e-Business Management. 2020; 18(3):405-26.
[47] Polo J, Castillo C, Carrera D, Becerra Y, Whalley I, Steinder M, et al. Resource-aware adaptive scheduling for mapreduce clusters. In ACM/IFIP/USENIX international conference on distributed systems platforms and open distributed processing 2011(pp. 187-207). Springer, Berlin, Heidelberg.
[48] He C, Lu Y, Swanson D. Real-time scheduling in MapReduce clusters. In international conference on high performance computing and communications & international conference on embedded and ubiquitous computing 2013 (pp. 1536-44). IEEE.
[49] Liang Y, Wang Y, Fan M, Zhang C, Zhu Y. Predoop: preempting reduce task for job execution accelerations. In workshop on big data benchmarks, performance optimization, and emerging hardware 2014 (pp. 167-80). Springer, Cham.
[50] Pastorelli M, Carra D, Dell'Amico M, Michiardi P. HFSP: bringing size-based scheduling to Hadoop. IEEE Transactions on Cloud Computing. 2015; 5(1):43-56.

[51] Verma A, Cherkasova L, Campbell RH. Aria: automatic resource inference and allocation for MapReduce environments. In proceedings of the ACM international conference on Autonomic computing 2011 (pp. 235-44).
[52] Voicu C, Pop F, Dobre C, Xhafa F. MOMC: multi-objective and multi-constrained scheduling algorithm of many tasks in Hadoop. In international conference on P2P, parallel, grid, cloud and internet computing 2014(pp. 89-96). IEEE.
[53] Han J, Yuan Z, Han Y, Peng C, Liu J, Li G. An adaptive scheduling algorithm for heterogeneous Hadoop systems. In international conference on computer and information science 2017 (pp. 845-50). IEEE.
[54] Dong X, Wang Y, Liao H. Scheduling mixed real-time and non-real-time applications in mapreduce environment. In international conference on parallel and distributed systems 2011 (pp. 9-16). IEEE.
[55] Liu L, Zhou Y, Liu M, Xu G, Chen X, Fan D, Wang Q. Preemptive Hadoop jobs scheduling under a deadline. In eighth international conference on semantics, knowledge and grids 2012 (pp. 72-9). IEEE.
[56] Cho B, Rahman M, Chajed T, Gupta I, Abad C, Roberts N, Lin P. Natjam: design and evaluation of eviction policies for supporting priorities and deadlines in MapReduce clusters. In proceedings of the annual symposium on cloud computing 2013 (pp. 1-17).
[57] Ullah I, Khan MS, Amir M, Kim J, Kim SM. LSTPD: least slack time-based preemptive deadline constraint scheduler for Hadoop clusters. IEEE Access. 2020; 8:111751-62.
[58] Mao H, Hu S, Zhang Z, Xiao L, Ruan L. A load-driven task scheduler with adaptive DSC for MapReduce. In international conference on green computing and communications 2011 (pp. 28-33). IEEE.
[59] Teng F, Yang H, Li T, Yang Y, Li Z. Scheduling real-time workflow on MapReduce-based cloud. In international conference on innovative computing technology 2013 (pp. 117-22). IEEE.
[60] Cheng D, Rao J, Guo Y, Zhou X. Improving MapReduce performance in heterogeneous environments with adaptive task tuning. In proceedings of the international middleware conference 2014(pp. 97-108).
[61] Rasooli A, Down DG. COSHH: a classification and optimization based scheduler for heterogeneous Hadoop systems. Future Generation Computer Systems. 2014; 36:1-5.
[62] Tang Z, Liu M, Ammar A, Li K, Li K. An optimized MapReduce workflow scheduling algorithm for heterogeneous computing. The Journal of Supercomputing. 2016; 72(6):2059-79.
[63] Li J, Liu Y, Pan J, Zhang P, Chen W, Wang L. Map-balance-reduce: an improved parallel programming model for load balancing of MapReduce. Future Generation Computer Systems. 2020; 105:993-1001.
[64] Ibrahim S, Jin H, Lu L, Wu S, He B, Qi L. Leen: locality/fairness-aware key partitioning for mapreduce in the cloud. In second international conference on cloud computing technology and science 2010 (pp. 17-24). IEEE.
[65] Nguyen P, Simon T, Halem M, Chapman D, Le Q. A hybrid scheduling algorithm for data intensive workloads in a MapReduce environment. In international conference on utility and cloud computing 2012(pp. 161-7). IEEE.
[66] Li Y, Lin C, Ren F, Geng Y. H-pfsp: Efficient hybrid parallel pfsp protected scheduling for mapreduce system. In international conference on trust, security and privacy in computing and communications 2013 (pp. 1099-106). IEEE.
[67] Wang J, Yao Y, Mao Y, Sheng B, Mi N. Fresh: fair and efficient slot configuration and scheduling for hadoop clusters. In international conference on cloud computing 2014 (pp. 761-8). IEEE.
[68] Zhao H, Yang S, Chen Z, Fan H, Xu J. K%-fair scheduling: a flexible task scheduling strategy for balancing fairness and efficiency in MapReduce systems. In proceedings of international conference on computer science and network technology 2012 (pp. 629-633). IEEE.
[69] Cheng YW, Lo SC. Improving fair scheduling performance on Hadoop. In international conference on platform technology and service (PlatCon) 2017 (pp. 1-6). IEEE.
[70] Hussain R, Rahman M, Masud KI, Roky SM, Akhtar MN, Tarin TA. A novel approach of fair scheduling to enhance performance of Hadoop distributed file system. In international conference on electrical, computer and communication engineering 2019 (pp. 1-6). IEEE.
[71] Chen Y, Alspaugh S, Borthakur D, Katz R. Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In proceedings of the ACM European conference on computer systems 2012 (pp. 43-56).
[72] Wang L, Khan SU, Chen D, Kołodziej J, Ranjan R, Xu CZ, Zomaya A. Energy-aware parallel task scheduling in a cluster. Future Generation Computer Systems. 2013; 29(7):1661-70.
[73] Lu Q, Li S, Zhang W. Genetic algorithm based job scheduling for big data analytics. In international conference on identification, information, and knowledge in the internet of things 2015(pp. 33-8). IEEE.
[74] Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W. Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Transactions on Parallel and Distributed Systems. 2014; 26(10):2720-33.
[75] Wen YF. Energy-aware dynamical hosts and tasks assignment for cloud computing. Journal of Systems and Software. 2016; 115:144-56.
[76] Pandey V, Saini P. A heuristic method towards deadline-aware energy-efficient mapreduce scheduling problem in Hadoop YARN. Cluster Computing. 2021; 24(2):683-99.
[77] Wang J, Li X, Ruiz R, Yang J, Chu D. Energy utilization task scheduling for MapReduce in

heterogeneous clusters. IEEE Transactions on Services Computing. 2020.
[78] Chen L, Liu ZH. Energy-and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenter. Service Oriented Computing and Applications. 2019; 13(4):297-308.
[79] Yuan Z, Wang J. Research of scheduling strategy based on fault tolerance in Hadoop platform. In international conference on geo-informatics in resource management and sustainable ecosystem (pp. 509-17). Springer, Berlin, Heidelberg.
[80] Chen Q, Liu C, Xiao Z. Improving MapReduce performance using smart speculative execution strategy. IEEE Transactions on Computers. 2013; 63(4):954-67.
[81] Yildiz O, Ibrahim S, Phuong TA, Antoniu G. Chronos: failure-aware scheduling in shared Hadoop clusters. In international conference on big data (Big Data) 2015 (pp. 313-8). IEEE.
[82] Yildiz O, Ibrahim S, Antoniu G. Enabling fast failure recovery in shared Hadoop clusters: towards failure-aware scheduling. Future Generation Computer Systems. 2017; 74:208-19.
[83] Guo Y, Bland W, Balaji P, Zhou X. Fault tolerant MapReduce-MPI for HPC clusters. In proceedings of the international conference for high performance computing, networking, storage and analysis 2015 (pp. 1-12).
[84] Brahmwar M, Kumar M, Sikka G. Tolhit–a scheduling algorithm for Hadoop cluster. Procedia Computer Science. 2016; 89:203-8.
[85] Zhu Y, Samsudin J, Kanagavelu R, Zhang W, Wang L, Aye TT, et al. Fast recovery MapReduce (FAR-MR) to accelerate failure recovery in big data applications. The Journal of Supercomputing. 2020; 76(5):3572-88.

Arif Ahmad Shehloo is a Research Scholar in the Department of Computer Science and Applications at Mewar University, Rajasthan, India. He received his Master's degree in Computer Science & Applications (MCA) & Bachelor's Degree in Science from the University of Kashmir, J&K India.
Email: arif4aziz@gmail.com

Muheet Ahmed Butt is a Scientist "D" in the Post Graduate Department of Computer Science, University of Kashmir, Srinagar, India. He received his PhD degree from the University of Kashmir, J&K India and M.Tech in Communications and Information Technology from the National Institute of Technology [NIT] Srinagar. Additionally, he holds a Bachelor of Science in Computer Science Engineering from Bangalore University, India.
Email: ermuheet@gmail.com

Majid Zaman is Scientist "D" in the Directorate of Information Technology & Support System, University of Kashmir, J&K India. He holds a PhD in Computer Science from the University of Kashmir, Srinagar, J&K, and an M.S. in Software Systems from the Birla Institute of Technology and Science (BITS) Pilani, India. In addition, he received a Bachelor's in Computer Science Engineering from BAMU Mumbai, India.
Email: zamanmajid@gmail.com

Appendix I

S. No.	Abbreviation	Description
1	ADAPT	Availability-Aware MapReduce Data Placement
2	ARIA	Automatic Resource Inference and Allocation
3	AUMD	Approximately Uniform Minimum Degree of parallelism
4	BDA	Big Data Analytics
5	BEEMR	Berkeley Energy Efficient MapReduce
6	CH	Cloud Hosts
7	CM	Constraint Model
8	CoGRS	Centre of Gravity Scheduling
9	COSHH	Classification and Optimization Based Scheduler for Heterogeneous Hadoop Systems
10	CP	Constraint programming
11	DAG	Directed Acyclic Graph
12	DARE	Distributed Adaptive Data Replication
13	DPMQS	Dynamic Priority Multiqueue Scheduler
14	DSC	Dynamic Slot Controller
15	DTSS	Dynamic Task Split Scheduling
16	DVFS	Dynamic Voltage and Frequency Scaling
17	EC2	Elastic Compute Cloud
18	EDF	Earliest Deadline First
19	EMRSA	Energy-aware MapReduce Scheduling Algorithms
20	EMRSY	Energy-efficient MR Scheduling YARN
21	FAR-MR	Fast Recovering MapReduce
22	FIFO	First In First Out
23	FRESH	Fair and Efficient Slot Configuration and Scheduling for Hadoop Clusters
24	GA	Genetic Algorithm
25	GFS	Google's File-System
26	HDFS	Hadoop Distributed File System
27	HFS	Hadoop Fair Scheduler
28	HFSP	Hadoop Fair Sojourn Protocol
29	H-PFSP	Hybrid Parallel pessimistic Fair Schedule Protocol
30	HPC	High-Performance Computing
31	HyBS	Hybrid Scheduling
32	IDLACO	Improving Data Locality through Ant Colony Optimization
33	IPP	Integer Programing problem
34	JVM	Java Virtual Machine

35	LARTS	Locality Aware ReduceTask Scheduling
36	LaSA	Locality-aware Scheduling Algorithm
37	LATE	Longest Approximate Time to End
38	LEEN	Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud
39	LONARS	Locality and Network-Aware Reduce Task Scheduling
40	MBR	Map-Balance-Reduce
41	MOMC	Multi-Objective & Multi-Constrained
42	MPI	Message-Passing Interface
43	MR	MapReduce
44	MRA++	MapReduce with Adapted Algorithms
45	MTL	Multi-Threading Locality
46	OP	Optimizer
47	PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
48	PSO	Particle Swarm Optimization
49	RDS	Resource & Deadline-aware Scheduler
50	RHC	Receding Horizon Control
51	RTMR	Real-Time MapReduce
52	SLO	Service Level Objective
53	SLV	Service level value
54	SPS	Shortest Period Scheduler
55	TFS	Task Forward Scheduling
56	UAAS	Utilizing All Available Slots
57	VM	Virtual Machine
58	YARN	Yet Another Resource Negotiator