*The Knowledge Engineering Review*, Vol. 34, e4, 1–33. 
doi:10.1017/S0269888918000371

# Jargon of Hadoop MapReduce scheduling techniques: a scientific categorization

MUHAMMAD HANIF and CHOONHWA LEE

*Division of Computer Science and Engineering, Hanyang University, Seoul, Republic of Korea;*
*e-mail: honeykhan@hanyang.ac.kr, lee@hanyang.ac.kr*

**Abstract**

Recently, valuable knowledge that can be retrieved from a huge volume of datasets (called Big Data) set in motion the development of frameworks to process data based on parallel and distributed computing, including Apache Hadoop, Facebook Corona, and Microsoft Dryad. Apache Hadoop is an open source implementation of Google MapReduce that attracted strong attention from the research community both in academia and industry. Hadoop MapReduce scheduling algorithms play a critical role in the management of large commodity clusters, controlling QoS requirements by supervising users, jobs, and tasks execution. Hadoop MapReduce comprises three schedulers: FIFO, Fair, and Capacity. However, the research community has developed new optimizations to consider advances and dynamic changes in hardware and operating environments. Numerous efforts have been made in the literature to address issues of network congestion, straggling, data locality, heterogeneity, resource under-utilization, and skew mitigation in Hadoop scheduling. Recently, the volume of research published in journals and conferences about Hadoop scheduling has consistently increased, which makes it difficult for researchers to grasp the overall view of research and areas that require further investigation. A scientific literature review has been conducted in this study to assess preceding research contributions to the Apache Hadoop scheduling mechanism. We classify and quantify the main issues addressed in the literature based on their jargon and areas addressed. Moreover, we explain and discuss the various challenges and open issue aspects in Hadoop scheduling optimizations.

## 1 Introduction

One of the indispensable qualities of cloud computing is the aggregation of resources and data in data centers over the Internet. Recently, cloud computing (Armbrust *et al.*, 2010) has transmuted the bulky part of the IT industry to make services more affordable by offering a pay-as-you-go package. Different cloud services improve the aggregate applications at different levels, such as servers, OS, and middleware levels. Furthermore, the amount of digital data is increasing daily as Google processes about 20 petabytes of information per day (Gunelius 2015). To make use of this digital information, these tens of petabytes of data must be handled properly, which requires a new type of technology rather than traditional information and communications termed as big data processing.

The term big data (James *et al.*, 2011) refers to datasets of enormous volume, complexity, and growth rate, which makes them difficult to be stored, managed, and processed by conventional technologies and tools such as relational database management systems and other traditional statistics systems within the necessary time to take advantage of the analysis. The information is coming from instrumented, consistent supply chains transmitting real-time data about instabilities in everything from market demand to weather. Furthermore, deliberate information has begun arriving through unstructured digital channels: social

media, smartphone applications, and an ever-increasing stream of emerging Internet-based gadgets. The data management of this large volume of data comes with some challenges: data are 'big'; hence, how should we scale them?; data are fast," how do we keep up?; data are 'unstructured', how do we model them?; data come from 'many sources', how do we ingest and integrate this data?; and data 'contain spatial text, graphs'; hence, how do we analyze them? Cloud computing is a point of interest for industrial and academic research communities because it is scalable, distributed, and has fault tolerant storage servers and applications which handle challenges related to big data. The data processing of big data in the cloud computing environment is a core issue and has been the focus of research for quite a while. Hadoop (Apache! 2015c) has ascended as an open source implementation of a distributed storage and processing framework which supports computational models like MapReduce on large-scale data over distributed infrastructures such as cloud computing.

MapReduce (Dean & Ghemawat 2008), popularized by Google, is an emerging data-intensive programming model for large-scale data parallel applications, such as web indexing, data mining, and scientific simulations. MapReduce offers an easy parallel programming interface in a distributed computing environment. It is primarily used for distributed fault tolerance for managing multiple processing nodes in Hadoop's MapReduce clusters. The most influential feature of MapReduce is its high scalability that allows users to process an enormous amount of data in a brief amount of time. There are numerous fields that benefit from MapReduce, such as bioinformatics (Matsunaga *et al*., 2008); machine learning (Chu *et al*., 2007); scientific analysis (Ekanayake *et al*., 2008); and web data analysis, astrophysics, and security (Mackey *et al*., 2008). Hadoop has been successfully adopted by many companies including AOL, Amazon, Facebook, Yahoo, and the New York Times to run their applications over the cluster. For example, AOL uses it to run an application that analyzes the behavioral pattern of their users to offer targeted services based on location or interest, etc. In such distributed environments, scheduling plays an imperative role in the performance and efficiency of frameworks.

Hadoop framework's default scheduler is based on a First-in First-out (FIFO) scheduling algorithm where jobs are executed in the order of their submission in the system. After a job is partitioned into individual tasks, they are loaded into the queue and assigned to free containers as they become available on nodes in the cluster. Although there is support for the assignment of priorities to jobs, it is not turned on by default. By default, each job would use the entire Hadoop cluster; thus, jobs had to wait for their turn to be processed. Although a shared cluster offers great potential for offering large resources to many users, the problem of sharing resources fairly among users requires a better scheduler, such as Capacity Scheduler and Fair Scheduler. These schedulers have been added in the new releases of Hadoop by Yahoo and Facebook, respectively. Production jobs need to be completed in a timely manner while allowing users who are building smaller ad-hoc queries to acquire results in a reasonable time.

To handle numerous nodes with various MapReduce jobs in a shared cluster environment, an efficient scheduling mechanism is needed to achieve the best performance. There are quite a few key factors that affect scheduling issues, such as locality, synchronization, fairness constraints, etc. Locality is about engaging a task on the node containing the input data. Scheduling in view of locality can save the network I/O cost because network bandwidth is an inadequate resource in a shared cluster. Synchronization overhead is instigated by stragglers, which causes an intact delay in the reduce phase of MapReduce jobs and an under-utilization of resource problem in the cluster. Another important issue is how to allocate jobs with fairness among multiple jobs and multiple users.

Hadoop scheduling considers different factors such as the state of tasks or jobs, node availability, locality, among others. Several efforts have been made to date to optimize Hadoop schedulers for job/task assignment to available resources in clusters of homogeneous/heterogeneous machines with some modest assumptions. The authors in Yoo and Sim (2011) and some others, Rao and Reddy (2012) and Tiwari *et al*. (2015), present surveys on scheduling techniques and issues faced in MapReduce, such as locality, synchronization, fairness, and heterogeneity. Their research approach is to present the strength and weakness of the techniques and compare them on the basis of their pros and cons. In addition, their revisions focus on partial numbers of techniques such as Longest Approximate Time to End (LATE) (Zaharia *et al*., 2008), Dominant Resource Fairness (DRF) (Ghodsi *et al*., 2011), and Capacity Scheduler (Apache! 2015a), etc., and do not cover the development area entirely. These approaches are unable to classify various methods

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

and techniques according to precise topics or issues covered, such as data locality, data replication, speculation, starvation, and fault tolerance. In this paper, we explain the issues related to the scheduling in Hadoop MapReduce and different techniques used in handling or solving these issues. The available scheduling techniques have been categorized and compared on the basis of their unique features, pluses, and affected areas in the Hadoop MapReduce framework. The objectives of this paper include,

- Discern and explain different issues related to the scheduling problem in Hadoop MapReduce, such as locality, synchronization overhead, fairness constraints, etc.
- Create a landscape review of existing scheduling practices in Hadoop.
- Categorize these existing Hadoop MapReduce scheduling techniques.
- Create an extensive analysis view of existing scheduling techniques to help researchers understand the overall view of the area.

The remainder of the paper is organized as follows: Section 2 briefly describes the background knowledge about Hadoop and MapReduce framework while Section 3 explains the different issues affecting scheduling performance in the Hadoop framework. In Section 4, recently improved schedulers for Hadoop MapReduce are discussed. Section 5 creates a comparative analysis view of Hadoop scheduling techniques with state-of-the-art schedulers, and Section 6 concludes the work.

## 2 Background

In this section, we will briefly describe the MapReduce framework, Hadoop implementation, and the role of the scheduling procedure to set the context for our subsequent analysis of various Hadoop MapReduce schedulers.

Hadoop is one of the most popular open source implementations of the Google MapReduce (Apache! 2015e) parallel processing framework. Hadoop hides the details of parallel processing, including data distribution over the cluster nodes, speculative execution for slower tasks, restarting of failed tasks, exchanging of intermediate data over the nodes, and merging results after computation finish. This type of framework permits developers to write parallel programs that place an emphasis on their computational problem, rather than the parallelization issues which are handled by the framework, and it also allows programmers to abstract from other scheduling issues such as parallelization, replication, partitioning, etc. Hadoop follows a master slave architectural design for both data storage and disturbed data processing, the underlying architecture of the Hadoop ecosystem is shown in Figure 1. It has been separated vertically into the Hadoop Distributed File System (HDFS) layer and MapReduce layer, of which each work in parallel manner. The HDFS layer handles data-related information such as meta-data for data distribution and partitioning, whereas the MapReduce layer keeps track of computation and application related information such as scheduling tasks and executing tasks over the cluster. The master node for HDFS layer is NameNode and the master node for MapReduce layer is Application Manager. The slave nodes are the machines in the cluster which store data and perform complex computations. Every slave node has DataNode and Application Master daemons that synchronizes with NameNode and Application Manager accordingly. Hadoop cluster can be setup either in the cloud or on-premise. Apache Hadoop includes HDFS and Hadoop MapReduce, which are discussed in details in the following subsections.

### *2.1 Hadoop distributed file system*

The HDFS is inspired by Google File System (GFS) for storing large amounts of data with high throughput (as high number of bytes read/write) access to data over huge clusters of commodity servers (Shafer *et al*., 2010). HDFS has fault tolerance, high throughput, and server failure handling mechanism similar to GFS. It also maintains by default three replicas (a second copy should be stored in a local rack and a third copy stored in a remote rack's node) of its data split across different machines to provide data locality and reliability. By default, files are divided into 64 MB chunks and are usually once-write-multiple-read chunks. HDFS is designed to run over distributed machines throughout the cluster and to provide high throughput, lower latency, and survive individual server failures. It achieves reliability by replicating data

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

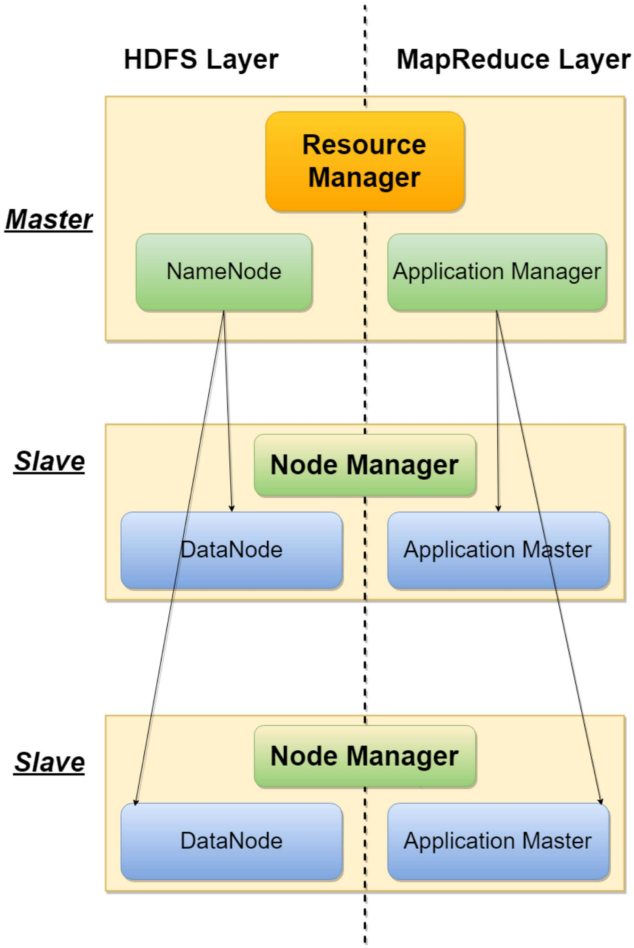


**Figure 1** High level Hadoop system architecture.

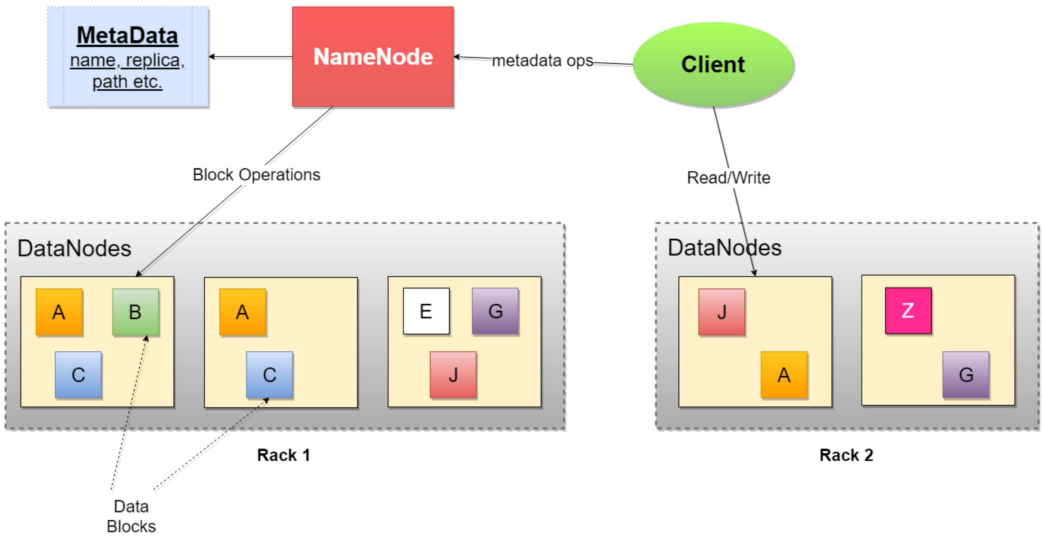


**Figure 2** Hadoop distributed file system architecture

across multiple nodes. As shown in Figure 2, it is built based on the principle of master-slave architecture. It comprises a single NameNode and multiple DataNodes. NameNode is responsible for assigning data blocks to DataNodes, and managing file system operations, such as opening, closing, renaming, among

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

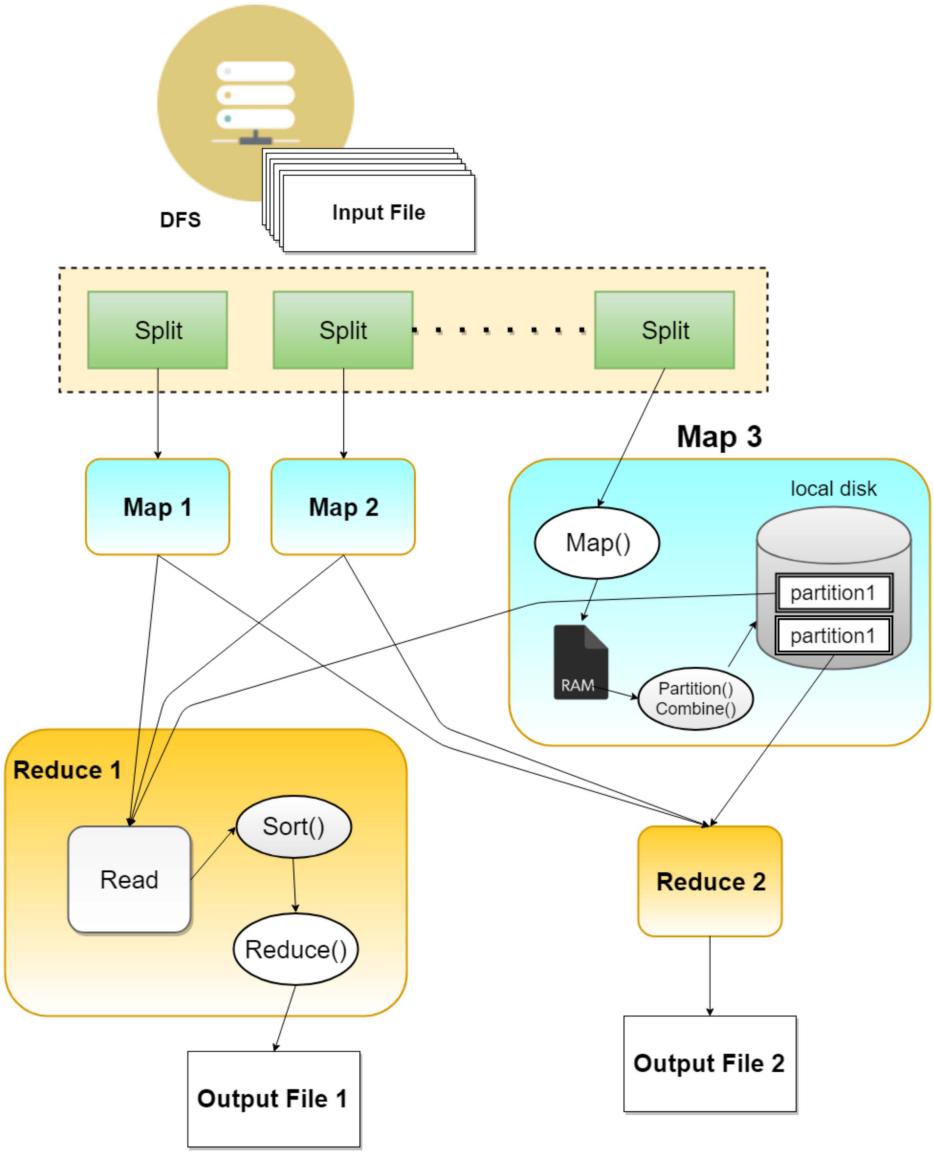


**Figure 3** Hadoop MapReduce architecture.

others, files and directories in the cluster. A DataNode is the actual storage resource for the running jobs of the Hadoop cluster. NameNode instructs the DataNodes to create, delete, and modify blocks throughout the cluster. NameNode also maintains file system namespace which records metadata regarding the creation, deletion, and modification of files by client programs.

### *2.2 Hadoop MapReduce*

The processing spine in the Hadoop system is a parallel processing framework called MapReduce. Hadoop MapReduce is inspired by the originally designed parallel processing framework in Google, named MapReduce. It is a simple and efficient approach for big data processing. It hides low-level programming details, such as partitioning, data distribution, and scheduling, from the developers. It usually divides a bigger problem into smaller pieces and runs in a parallel manner over the Hadoop cluster. MapReduce programs run in two phases, Map and Reduce, as shown in Figure 3. The input data chunks obtained from splitting input files are mapped as <key, value> pairs in the map phase. The output of the map phase is obtained as a collection of key-value pairs called intermediate data which are aggregated in the reduce phase on the basis of the keys throughout the data over the cluster. The middle phase of the map and reduce

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

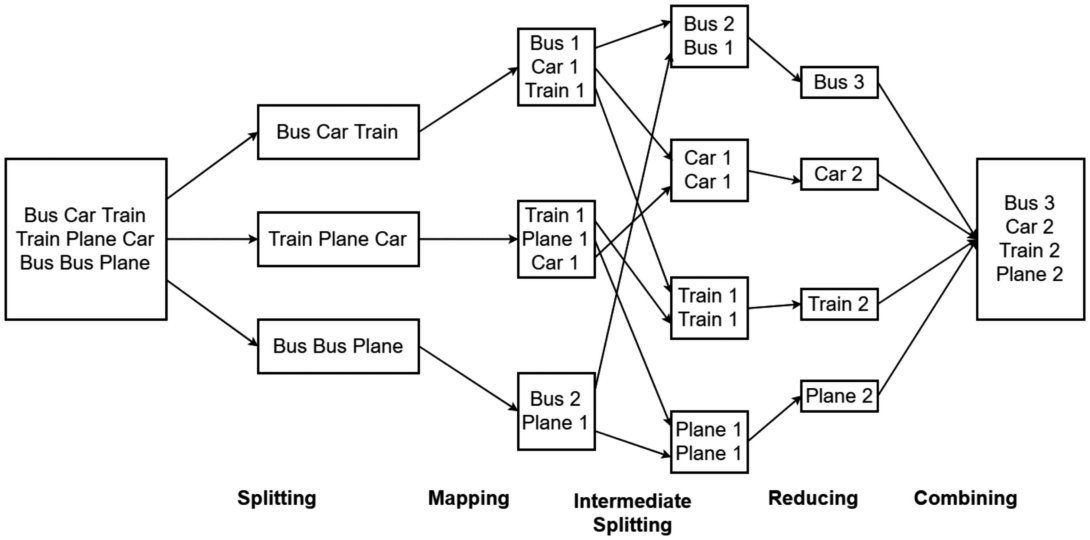


**Figure 4** Workflow of Map-Reducing of a word count example.

phase is called the Shue phase where the sorting, partitioning, and communicating among different mappers and reducers takes place. In Hadoop version 2, the job is submitted to the Resource Manager (RM) which manages the global assignment of computational resources to each application or job. For each job, there is a per-application Application Master which manages the life cycle of the application. The RM schedules tasks to available resources in the cluster using the pluggable Scheduler component. In the Hadoop system, there are different types of schedulers available such as FIFO, Capacity Scheduler, and Fair Scheduler; additionally, different researchers proposed several scheduling techniques to achieve better cluster performance, which will be discussed in later sections.

**Hadoop MapReduce word count example:** Word count reads text files and counts how often each word occurs in the document and write the output result to one or more files accordingly. Each mapper takes a line as an input and breaks it down into words. It then emits a <key, value> pair of the word and value 1 each time it occurs. Each reducer sums the counts for each word and emits a single < key, value> pair with the word and the sum. A combiner is used on the map output as an optimization that combines each word into a single record and reduces the amount of data sent across the network in the shuffle phase. Suppose we have an input file containing the travel arrangement of a Tech Company CEO to quarterly business meeting at different branches of the company. Each line represents different year travel record of each quarter. The workflow of Map-Reducing of a word count example is shown in Figure 4. The whole process consists of five steps. (1). *Splitting:* Splits the input using splitting parameter such as splitting by space, semicolon, comma, or new line characters. The input file is divided by new line and each line is given to different machine to process. (2). *Mapping:* takes a dataset and convert it into a set of <key, value> pairs. Input lines are converted to <key, value> pairs of rides by types of vehicle with the number of time as value. (3). *Intermediate Splitting:* the order to group the same keys, it shuffle the data accordingly. (4). *Reduce:* takes the output from Map as an input and combine those data tuples into a smaller set of tuples. (5). *Combining:* gather the individual results and combine it into an output. The final output file contains results of the CEO trips of the last three years that the CEO takes Bus 3 times, Car 2 times, Train 2 times, and Plane 2 times.

## 3 Scheduling issues in Hadoop

There are diverse scheduling issues such as locality, fairness, starvation, parallelization, replication, partitioning, among others, in Hadoop during the process of scheduling jobs and tasks in the cluster. These issues are not considered in the default Hadoop scheduling technique even though they affect the performance of the Hadoop MapReduce cluster. Significant issues are shown in Figure 5. All the issues are

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

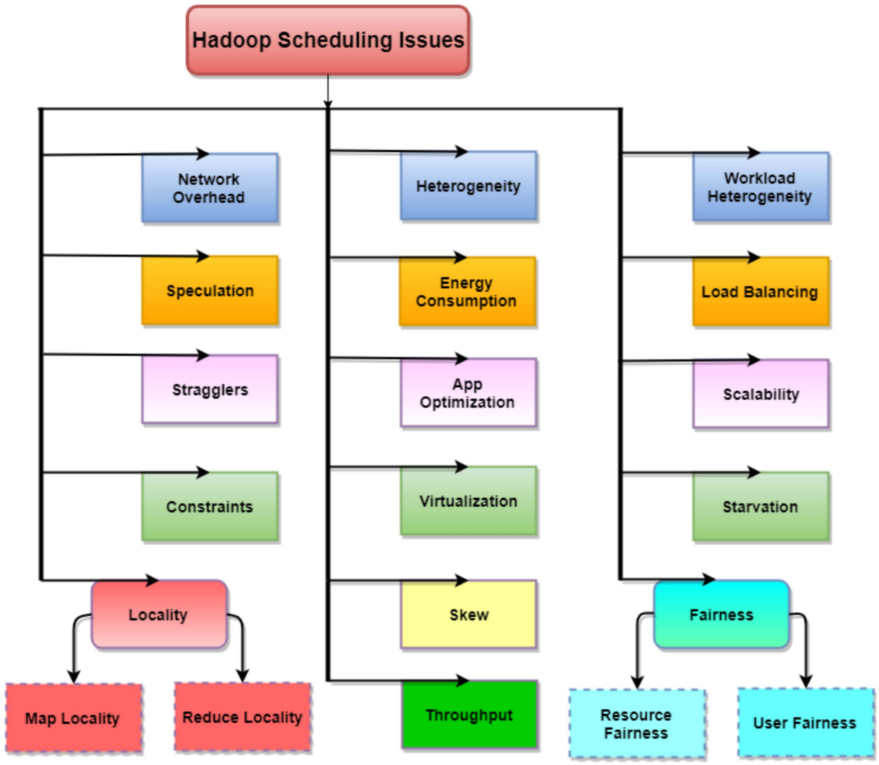


**Figure 5** Hadoop scheduling issues.

categorized as top to bottom and left to right, as more to less significant to the performance and efficient utilization of the cluster. The Locality issue received extensive research courage along with Fairness, therefore, both of the issues has been sub-categorized accordingly. Locality has sub-categories of Map task Locality and Reduce task locality, whereas Fairness has Resource Fairness and User Fairness. Each of these issues is explained in detail in the following subsections.

### *3.1 Heterogeneity*

Heterogeneity is a word that signifies diversity. A Hadoop cluster comprising machines with different processing power, memory, and input/output (I/O) would be considered as having the quality of heterogeneity. The trade-off between faster push and faster map is usually a challenge in heterogeneous environments.

### *3.2 Scalability*

Scalability in the case of Hadoop is the capability of the Hadoop MapReduce system to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. Usually there is performance vs scalability tradeoffs to consider as sometimes both of these are used interchangeably and will be discussed in details in later section.

### *3.3 Locality*

Data locality in Hadoop MapReduce clusters means that the tasks are assigned to those nodes which are holding the data or where the data is stored locally for those tasks. The execution of local data by a map reduce task plays a vital role in the inclusive performance of MapReduce job. There are two kinds of locality (the distance between a node with the input data and the task allocated node) addressed in Hadoop by different researchers: Map locality and Reduce locality. Map locality in Hadoop MapReduce clusters means that the Map tasks are assigned to those nodes which have the input data for those Map tasks.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Reduce locality in MapReduce clusters means that the Reduce tasks are assigned to those nodes which have most of the intermediate data for Reduce tasks. In bulky clusters, data locality is critical because the network bisection bandwidth becomes a bottleneck. Therefore, we categorize scheduling techniques that address the problem of data locality in scheduling.

### *3.4 Network overhead*

Network overhead is any combination of excess or unnecessary bandwidth usage during the execution of a MapReduce job in Hadoop cluster. In Hadoop, the MapReduce application works in two phases: Map phase and Reduce phase. The output data of mappers in map phase and input to the reducers in the reduce phase is known as intermediate data. The exchange of intermediate data and communication between mappers to mappers, reducers to mappers, and reducers to reducers plays an important role in overall job performance and execution time.

### *3.5 Speculation*

Speculative execution is an optimization technique where some nodes in the Hadoop system perform a task that may not necessarily be needed, i.e., executing extra copies of tasks to mitigate the straggling tasks execution problem on stragglers nodes in the system. The speculative execution in the default Hadoop system is based on several false assumptions, such as the cluster contains only homogeneous machines, all tasks progress at the same rate, and all the Reduce tasks process the same amount of data. Speculative execution is directly proportional to network overhead, so as the speculative jobs increases, it also increases the network overhead. So the user has to decide a trade-off between these two.

### *3.6 Straggler*

Nodes that have the lagging out behavior in the cluster slow down execution of other parallel running tasks in the Hadoop MapReduce system. These nodes ultimately prolong the total execution time of the MapReduce job. Stragglers affect the flow of data over the network, especially in the shuffle phase of the process. Hence it is related to the network overhead problem in a direct way.

### *3.7 Fairness*

Fairness is a property of unbounded non-determinism, i.e., the quality of the system to give fair resources to every user or job by making judgments that are free from discrimination. Several MapReduce jobs are executed in a multi-tenant manner or shared data warehouses of cloud computing enterprises like Amazon, Google, Facebook, and Yahoo, to use the available resources efficiently. A MapReduce job with a heavy workload may dominate the utilization of the shared clusters. As a result, some short jobs may not have the desired response time. The demands of the workload can be elastic. Thus, a fair sharing of resources with each job or user in the shared cluster should be considered. Fairness constraints have trade-offs with locality, dependency, and several other factors depending on the environment. In most circumstances, a corporate approach used to ensure fairness is to schedule jobs, such that all users or jobs receive an equal percentage of resources.

### *3.8 Constraints*

Constraints are the conditions or specific settings and configurations which are requisite for the application's survival, such as time constraints for deadline-based applications and resource constraints such as minimum resources to run an application or program smoothly.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

### 3.9 Virtualization

Virtualization is the formation of a virtual (rather than concrete or physical) form of resources, such as an operating system, a server, a storage device, or a network. Hadoop has a different behavior under virtualized resources which is targeted in both academic and industry-based research work.

### 3.10 Starvation

Starvation is a problem encountered in a parallel computing framework like MapReduce where a process is perpetually denied the necessary resources to process its work (Tanenbaum 2009). Starvation may be caused by errors in scheduling or a mutual exclusion algorithm, but can also be caused by resource leaks, and it can be intentionally caused via a denial-of-service attack, etc. In Hadoop scheduling, such a scenario occurs where one type of task starves for a long time while other jobs or tasks hold the resources for long periods of time. For example, in FIFO-based scheduling, long jobs hold the execution queue for long a time while the short jobs are starved until the longer jobs finish execution.

### 3.11 Load balancing

Load balancing distributes workloads across multiple computing resources, such as commodity computers in a cluster, network links, CPUs, or disk drives. Load balancing aims to optimize resource usage, maximize throughput, minimize response time, and avoid overloading any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing helps to increase performance gain and reduce different kinds of congestion in Hadoop's MapReduce systems.

### 3.12 Skew

Data skew principally refers to a non-uniform distribution in a dataset. Skewed distribution can follow common distributions, for instance, Zipfian, Gaussian, and Poisson; however, Zipfian distribution is commonly considered to be a model for skewed datasets. The direct impact of data skew on the parallel execution of complex big data applications results in poor load balancing leading to a higher response time. In Hadoop's MapReduce applications, it leads to a computational skew caused by the load imbalance of the intermediate data in the cluster.

### 3.13 Workload heterogeneity

Workload heterogeneity means that workloads may have heterogeneous resource demands because some workloads may be CPU-intensive, whereas others are I/O intensive. Others might be able to use special hardware like GPUs to achieve dramatic performance gains.

### 3.14 App optimization

Application optimization is the process of modifying an application to make some aspect of it work more efficiently. In general, an application like PageRank, Grep, etc., may be optimized so that it executes more rapidly, or is capable of operating with less memory storage or other resources, or draws less power. In the case of Hadoop, application optimization means that the underlying architecture of Hadoop must not change, and instead use a different application implementation so that it runs more efficiently over the Hadoop cluster.

### 3.15 Energy consumption efficiency

Energy consumption is a major and costly problem in data centers and big clusters such as a 4000 node Hadoop cluster. Big data and the use of commodity nodes in Hadoop clusters have caused a huge increase in power consumption and an increase in the operational cost of data centers. Therefore, increasing the energy efficiency of Hadoop clusters has attracted significant attention in both academia and industry

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

research. The low utilization levels of servers, owing to low load and data duplication, also result in poor energy efficiency in Hadoop clusters. For many workloads, a large fraction of energy goes to powering idle machines that are not performing useful work. There are two causes for this inefficiency: low server utilization and a lack of power proportionality. Therefore, energy consumption is an important issue in Hadoop clusters which has been addressed by numerous researchers.

### *3.16 Throughput*

In the process of analyzing big data and running big data applications, the number of jobs or tasks completed over time is of significant value, because they mirror the overall performance of the cluster. High throughput can be achieved by refining other quality measures, such as data localization, energy efficiency, and optimal resource utilization. In upcoming sections, we describe how several scheduling algorithms achieve throughput improvement and how, sometimes, overemphasis on data locality can actually reduce the throughput of the system.

## 4 Scientific categorization: jargon of scheduling techniques

The Hadoop MapReduce runtime infrastructure attempts to ensure different quality attributes including availability, performance, and fault tolerance. Every worker node sends a periodic heartbeat signal to the master node, and if any of the nodes do not respond within the stipulated amount of time, that node is marked as either dead or failed. When the master detects a failed node, it reschedules its assigned tasks to any other available nodes in the cluster. To handle fault tolerance and other-related problems like stragglers, the Hadoop system uses a number of different strategies, including data replication, starting backup tasks, i.e., speculative execution, and many more.

The scheduling problem in Hadoop is based on multiple aspects which need to be addressed; therefore, the efficient classification of these scheduling techniques needs to be broad enough to cover those aspects. Our classification and categorization process deliberates aspects of the scheduling techniques in the area. Different aspects of Hadoop schedulers include: a scheduling technique that can adapt to different situations and environments like data distribution, resources, and job characteristics; a technique designed to meet multiple QoS requirements, such as fault tolerance, availability, optimal resource utilization, fairness, energy consumption efficiency and reliability; and lastly, a technique that has different scopes, such as job scheduling or task scheduling, or that can cover both entities. These different aspects are dependent on each other in most of the real-world scenarios. Therefore, we use the principle of spiral model to show all these interdependence possibilities based on the situation and underlying platform at which the Hadoop system run, as shown in Figure 6. The Hadoop scheduling techniques are in the middle of the spiral so that to show that each aspect handling ultimately benefits the Hadoop scheduling.

The scope of the schedulers usually varies from system to system. Some systems utilize a job level scope, while others utilize a task level scope, and some even have a hybrid of both. A job level scheduling algorithm chooses a job from the job queue for the execution of its tasks. A cluster can have either a single job queue or multiple job queues containing a number of tasks. Task level scheduling involves the selection of tasks based on node availability. Whereas hybrid tries to schedule both jobs and tasks in efficient ways possible. In MapReduce programming model, there are two types of tasks, map and reduce, that need to be scheduled. There is one other type of task created during the runtime of a MapReduce job called a speculative task, which acts as a backup task in the case of straggling tasks. Each Hadoop application has an Application Master, which keeps track of the application's progress. Hadoop MapReduce scheduling is performed based on different metrics, including data locality on DataNodes, the input of map and reduce tasks, availability of containers, and avoiding factors such as long waiting times and stragglers. In the following sections, we will discuss several scheduling techniques and classify these approaches into various categories and classes based on the areas covered by these techniques, as shown in Figure 7.

Hadoop has two main versions as 1.0 and 2.0(YARN). Hadoop's version 1.0 default scheduler is a FIFO-based scheduler which maintains jobs in the queue which are delivered as they appeared or arrived.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

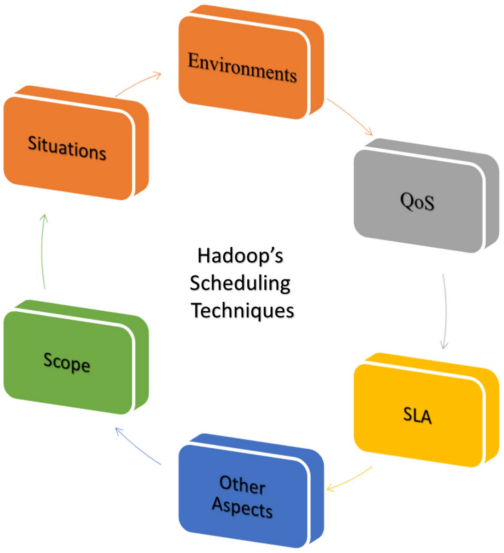


**Figure 6** Aspects of Hadoop scheduling techniques.

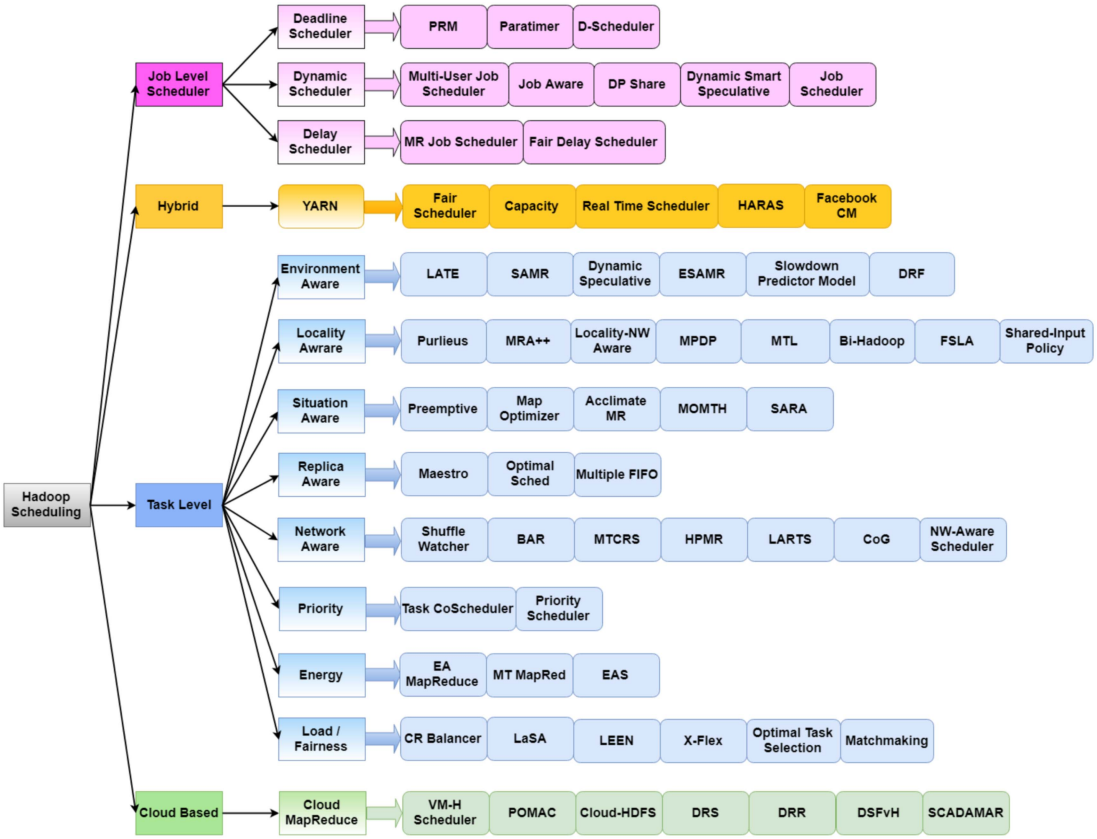


**Figure 7** Scientific categorization of Hadoop schedulers

It also supports prioritizing the tasks which are switched off by default and need to be turned on manually. Different scheduling techniques in Hadoop include deadline-based (Teng *et al.*, 2014), (Morton *et al.*, 2010); locality-aware (Palanisamy *et al.*, 2011), (Arslan *et al.*, 2014), (Wei *et al.*, 2015); situation-aware (Yoo & Sim 2011), delay (Zaharia *et al.*, 2009) (Zaharia & Borthakur *et al.*, 2010), resource-aware (Palanisamy *et al.*, 2011); heterogeneity-aware (Zaharia *et al.*, 2008); network-aware (Ahmad *et al.*, 2014), (Seo *et al.*, 2009); and environment-aware (Zaharia *et al.*, 2008), (Chen *et al.*, 2010) schedulers.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Yet Another Resource Negotiator (YARN) (Douglas *et al.*, n.d.) raises the default Hadoop to version 2.0 architecture to a new level by splitting the resource allocation and job scheduling duties from a JobTracker to a Resource Manager and Application Master, respectively. In YARN, the Fair scheduler (Apache! 2015b) divides and assigns a fair share of resources based on both memory and CPU capacity to all applications in the cluster, unlike the previous default fair scheduling which considered only memory capacity. In addition, in the new architecture, newly submitted applications do not have to wait and will be assigned resources immediately. The speculation of tasks depends upon the ratio of progress of tasks compared to the average progress of the job.

Although this paper does not discuss the high performance of computing schedulers and their optimization for schedulers other than Hadoop, a distributed framework scheduler such as Apache Mesos will be compared with the Hadoop version 2.0 i.e., YARN. Apache Mesos (Hindman *et al.*, 2011) is a distributed framework that provides abstraction over the storage, processing, and memory of machines (both physical and virtual). It is based on the Linux kernel, even though it has a different level of abstraction. Compared to YARN, Mesos supports several distributed frameworks, including Spark, Hadoop, and Kafka. It can be utilized for different purposes, including resource management and scheduling applications. Unlike YARN, it does not consider the MapReduce application specifically; and, like YARN, it also uses the master and slave architecture for resource management.

There are significant issues that occur while scheduling in Hadoop, and as we discussed in the previous section, various improvements and optimizations have been proposed for each of them in both academic and industrial research. We will review, criticize, and explicate these techniques according to our scientific classification. Our classification is based on the different areas covered for optimizing scheduling and mitigating problems (as discussed in the previous section) related to scheduling. We divide the Hadoop scheduling techniques and optimizations into four broad categories, which are further divided into subcategories, as shown in Figure 7. The categories are job level schedulers, task level schedulers, a hybrid of both job and task level schedulers, and cloud-based schedulers. These categories and their subcategories are explained in the following subsections.

### *4.1 Job level schedulers*

The job level scheduling algorithm selects a job from the job queue to schedule its tasks. A Hadoop cluster can have either a single job queue or multiple job queues. In the case of a single job queue, the jobs of all users are submitted to this sole job queue in the cluster, while in the case of multiple job queues, each job queue contains jobs from different users. There are numerous approaches that optimize job scheduling in Hadoop systems which will be discussed in the following subsections.

#### *4.1.1 Deadline schedulers*

These schedulers are specifically designed for special use cases where the deadline constraints are essential requirements in the Service Level Agreement (SLA). The schedulers usually try to predict the near-optimal approximation of the deadline for long-running jobs. Authors (Teng *et al.*, 2014) proposed the paused rate monotonic (PRM) scheduling algorithm that defines a certain response time for deadline constraints services of SLA and provisions coexisting service sharing in a cloud environment. It proposes a real-time scheduling technique to deliver broad formulation and theoretical analysis of Hadoop scheduling. It considers different dimensions of the real-time scheduling problem. First, it models services as a series of map reduce tasks and articulates the problem by identifying the features of tasks, algorithms, and clusters. Second, it develops a PRM scheduling algorithm for priority determined algorithms and to schedule tasks on the basis of their constraints. It pauses the MapReduce job in between the map and reduce phases to optimize the default system by using this time for partitioning, sorting, and combining intermediate results of the map phase. It helps maximize the number of tasks meeting their deadline and analyzed them claiming better performance than other real-time scheduling. In contrast , ParaTimer (Morton *et al.*, 2010) is designed for Dynamic Acyclic Graphs (DAGs) flair jobs written for Hadoop, and it estimates DAG progress in MapReduce. It distinguishes a critical path that takes longer than others in a comparable query plan. It makes the indicator overlook other shorter paths when approximating progress because the longest

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

path would subsidize the overall execution time. It leads to this rule of thumb about Hadoop performance: 'making the size of the data block bigger makes Hadoop work faster'.

D-Scheduler (Kc & Anyanwu 2010) is a deadline-based scheduling approach designed to satisfy the problem of long-running Hadoop jobs. In this work, the authors proposed two models to solve this problem. First is a job execution model which considers metrics like mappers and reducers running time, data distribution, and input data; and schedules the jobs on the basis of these metrics. Second is a deadline approximation approach to estimate the deadline for a single job based on certain assumptions, such as homogeneity in cluster, uniform distribution of keys throughout the cluster, data already transferred and stored in HDFS, and asynchronous execution of mappers and reducers; although they allow these assumptions to be broken at a later stage. To meet the response time SLA constraints (i.e., deadline), it is the obligation of the scheduling algorithm to estimate the number of container slots necessary to finish the job within the deadline and stated requirements to ensure the specified job receives those slots. The SLA Constraints algorithm accepts a job only if the estimated minimum number of slots required to meet the deadline SLA are available at its arrival time. After acceptance of a job, the scheduling algorithm maintains the minimum number of tasks running for the job to meet the deadline. It calculates the number of map and reduce tasks ($N_m^{min}$, $N_r^{min}$) depending on the input data set, output data sizes ($\sigma$, F$\sigma$), time required to process the block of data by mapper, time for reduce task execution, and time for Shue phase to transfer data ($c_m$, $c_r$, $c_d$). It also takes the arrival time of job (A), deadline of job (D), and starting time of map and reduce tasks ($s_m$, $s_r$) into account for calculating the minimum number of mappers and reducers using the following equations.

$$N_m^{min} = \left[\frac{\sigma * c_m}{(S_r^{max} - S_m)}\right] \tag{1}$$

$$N_r^{min} = \left[\frac{F\,\sigma * c_r}{(A + D - F\sigma * c_d - s_r)}\right] \tag{2}$$

The earliest deadline first algorithm accepts a job if its deadline can be met with the idle containers available after allotting the minimum required container slots to the jobs in the system. The algorithm assigns the necessary slots to jobs in increasing order of their deadlines. All extra free slots are distributed among certain jobs to finish them swiftly before the arrival of the new jobs.

### *4.1.2 Dynamic schedulers*

Dynamic schedulers are the scheduling algorithms that can work in dynamic environments and are able to adopt the changes in the required situations. The default scheduler in Hadoop schedules the reduce tasks of a job the moment one of the map tasks has completed. This often leads to a 'slot hoarding' problem where long-running reduce tasks continue waiting for all the map tasks to finish and thus waste CPU resources in busy-waiting. Authors in the multi-user MR job scheduler (Zaharia *et al*., 2009) propose a different approach from the default one in the case where the reduce task is executed. They split the reduce task into 'copy' and 'compute' phases and use their new algorithm to schedule the reduce tasks. The copy phase begins to collect the intermediate data after the completion of map tasks, while the compute phases process those intermediate data after they are collected and made available. They set a limit on the number of copy and compute phases running on each node such that the number of compute processes is always higher than that of the copy processes.

Nanduri *et al*. (2011) propose a task vector-based model, job-aware scheduler, to maintain congruence among jobs in the cluster by allocating tasks to nodes without affecting other tasks. They divide the job into map tasks and reduce tasks based on characteristics of the task, such as CPU-intensive, memory-intensive, disk-intensive, or network-intensive. It uses the task vector for the representation of each task and resource vector for each node. A task vector ($T_j$) is the combination of CPU, memory, disk and network utilization variables ($CPU_j$, $mem_j$, $disk_j$, $NW_j$) of job j, as shown in the equation below,

$$T_j = CPU_j * e1 + mem_j * e2 + disk_j * e3 + NW_j * e4 \tag{3}$$

where e1, e2, e3, and e4 are a weight base for each variable. Similarly, a resource vector is the summation

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

of available resources on the node. The task assignment uses two algorithms to check task compatibility with the requesting node. First, incremental Naive Bayes classifier is used to determine task compatibility with the task tracker specification using its vectors. Second, the heuristic-based algorithm is used to test the compatibility of the task with the task tracker. This is accomplished through computing the $T_{availabilityvector}$ as the difference of $T_r$(total resource) and $T_{compoundand}$ finally compute the $T_{unused}$ (unused resources). Using these heuristics, a combination value is computed and compared to the threshold to determine the compatibility of a task with the specified task tracker. This approach claims a 21% improvement of response time in the heuristic-based technique and a 27% improvement using the machine learning technique.

The dynamic proportional (DP) share (Sandholm & Lai 2010) scheduling technique is an extension of the default Hadoop scheduler to provide QoS to users based on certain priorities. It provides a user interface to select which job to schedule and prioritize among the total jobs. It also gives users the capability to adjust their allotted resources to scale-up and scale-down according to the job requirements. It supports preemption to avoid starvation and guarantees extra payment for holding the resources for a long time. It adopts to dynamic job priorities determined from their SLAs. The DP scheduler addresses the problem of Hadoop's Fair scheduler that is maintaining separate queues for separate pools, guaranteed service over time, and manual priority settings. It ensures that resource consumption is only allowed until the budget for the user is available. Despite being one of the most reliable schedulers that take QoS into account, it does not consider data locality or data replication, which has a firm ground on the dynamic proportional shares scheduling decision much precise and improved performance guarantee.

Dynamic smart speculative (Chen *et al*., 2014) proposed a new speculative execution algorithm to stun the problems that affect the performance for already implemented speculation techniques in Hadoop system, such as data skew, improper phase percentage configuration, and asynchronous start of certain tasks. However, it degrades cluster efficiency which implies a degradation of performance for batch jobs. While job scheduler (Xia *et al*., 2011) schedules tasks only on healthy nodes that have a lower ratio of error occurrence, it reduces the resource waste from failing tasks. When a healthy node reports an available slot, the Euclidean distance D between characteristic variables of a job is calculated by the following equation,

$$D = (C_{idle} - C_{usage})^2 + (I_{idle} - I_{usage})^2 + \left[\frac{(M_{idle} - M_{usage})}{(M_{idle})}\right]^2 \quad (4)$$

where $C_{idle}$, $I_{idle}$, and $M_{idle}$ are the available resources on the node and $C_{usage}$, $I_{usage}$, and $M_{usage}$ are the resources utilized by map and reduce tasks running on the node.

#### *4.1.3 Delay schedulers*

These schedulers usually consider delaying some jobs for a certain time, so that other jobs can benefit from the time delay. They attempt to ensure fairness by giving a slot to the job that has the fewest running tasks. Delay-based schedulers reduce starvation for small jobs at the cost of hurting the data locality. The delay scheduler (Zaharia, Borthakur, *et al*., 2010) follows the principle of giving a fair share of resources to jobs, including newly arrived jobs. In the Hadoop default FIFO scheduling algorithm, the task is scheduled and launched on any node (even to a non-local node) when it reaches the head of the waiting queue. Delay scheduler interrupts this FIFO principle and allows the task at the head of the queue to wait for its locality by delaying it and executing it at the data local node. Delay scheduler uses long task balancing and hot-spot replication to reduce the chance of a node becoming stuck in long tasks and simultaneous access to a small data file by multiple jobs. In long task balancing, it considers the new jobs as long tasks and identifies them as small tasks if they finish swiftly. In hot-spot replication, it replicates the tiny data file at runtime to be available on numerous nodes and it could be used by multiple jobs simultaneously. Through experimentation, a Facebook workload mix showed that smaller jobs benefit more from the Delay algorithm (Zaharia, Borthakur, *et al*., 2010) at the cost of longer jobs. In the case of IO-intensive jobs, those jobs with a large number of map tasks perform better than the jobs with a higher number of reduce tasks. It claims an almost five-times improvement in the response times of small jobs and a two-times increase in throughput for IO-intensive jobs.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

### 4.2 Task level schedulers

The task level scheduling algorithm includes the selection of tasks from a specific job of a specific user for scheduling on available nodes. Hadoop MapReduce framework has two types of tasks, i.e., map tasks and reduce tasks. Map tasks run the first phase of the MapReduce program by processing the entire input dataset where each map task processes one block of input data. Reduce tasks process the intermediate data produced by mappers and write back the results to the HDFS. Furthermore, there is another kind of task created in the Hadoop system called speculative tasks which handle straggler tasks in the system by duplicating a similar task in another available container in the cluster. The scheduling algorithms necessitate optimizing multiple attributes associated to these tasks, which is why a large number of research proposals developed novel scheduling algorithms to improve the scheduling of tasks (map, reduce, and speculative tasks) in the Hadoop cluster. These research proposals will be discussed in subsequent subsections.

#### 4.2.1 Environment-aware schedulers

The environment-aware schedulers are those schedulers which can adopt to a variety of different situations and environment. These environments include the running environment, i.e., homogeneous or heterogeneous environment, data distributions, resource characteristics, job characteristics, etc. There are numerous research efforts in this area including LATE (Zaharia *et al*., 2008), Self-Adaptive MapReduce (SAMR) (Chen *et al*., 2010), Enhanced Self-Adaptive MapReduce (ESAMR ) (Sun *et al*., 2012), and (Ghodsi *et al*., 2011). We believe that some other reseach proposal from other section can also be considered as part of environment-aware scheduler, but we classify those based on most relevance to a category or subcategory. These and other research efforts in the same category will be explained in detail in the upcoming paragraphs.

LATE (Zaharia *et al*., 2008), was initially developed for the heterogeneous cluster. It demonstrates that the implicit Hadoop default assumptions could be broken in a heterogeneous (Rao *et al*., 2012) cluster; for example, default Hadoop assumes nodes that perform roughly at the same rate (i.e., the cluster is composed of homogeneous machines) and constant throughput time could be broken owing to heterogeneity and assumptions, such as no extra cost for launching tasks. LATE exploits the speculative (Chen *et al*., 2014) execution strategy and proposes a new scheduling technique by prioritizing speculative tasks, selecting faster nodes for execution, and limiting the number of speculative copies of tasks using a threshold. It uses heuristics for calculating the Progress Rate (Zaharia *et al*., 2008) for the identification of faster nodes using the following equation:

$$ProgressRate = \frac{ProgressScore}{Execution\ Time\ of\ Task} \tag{5}$$

It schedules the speculative task for only those tasks whose Progress Rate is lower than a threshold based on the slowest task. Furthermore, it schedules the speculative task for the task which has the longest time to completion estimation using the equation below:

$$Time\ to\ Completion = \frac{(1 - ProgressScore)}{ProgressRate} \tag{6}$$

LATE compares the tasks progress rate and maintains a SlowTaskThreshold for the classification of a task as a slow task. It claims to double performance compared to the default Hadoop scheduler in a 200 Amazon's EC2 (Amazon! 2016d) virtual machine (VM) cluster. Unlike other state-of-the-art research proposals, the estimated remaining execution time for tasks in LATE is based on static information which is not very reliable in many scenarios, leading to improper slower nodes classification within the server.

The SAMR (Chen *et al*., 2010) scheduling algorithm proposed to maintain different weights for different map and reduce phases on every node based on task execution time on the specified node. These dynamic weights are used to estimate the time to end of execution of tasks on a specified node. The tasks with the longest time to end are scheduled for speculation. This solves the problem with the LATE (Zaharia *et al*., 2008) scheduler of using static information for decisions about slow tasks and nodes.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

SAMR maintains the historical information of each task, and identifies map slow tasks and reduces slow tasks separately. It uses Progress Rate with Average Progress Rate (APR) as

$$PR_i < (1 - STaC) * APR \tag{7}$$

where STaC is SlowTaskCap ranging from 0 to 1. In the same manner, for the identification of slow TaskTrackers, it uses the following equation:

$$TrR_{mi} < (1 - STrC) * ATrR_m \tag{8}$$

where STrC is SlowTrackerCap ranging from 0 to 1. SAMR launches the speculative copies for tasks based on slow tasks and accepts the results of the early finished task of both the original and the speculative copies. It launches the speculative copies on diverse nodes to ensure that they will no longer be slow. SAMR claims a 25% improvement in response time in comparison with the default Hadoop scheduler and a 14% improvement over the LATE (Zaharia *et al*., 2008) scheduler. However, it often does not consider the type of job, input data size, and nodes types in the calculation of their metrics to be used as the criteria for estimating remaining time of the execution, leading to incorrect classification and poor performance accordingly.

Dynamic scheduling for speculative execution (Jung & Nakazato 2014) considers the processing power and system resources of each node to improve the performance while scheduling the speculative copies of the tasks. The authors use the effective speculative execution term to execute backup copies of tasks for a slow node in a heterogeneous environment on faster nodes. They calculate the throughput per second (TPS) (Jung & Nakazato 2014) using the following formula,

$$TPS = \frac{(Progress_{i+1} - Progress_i)}{(ExecutionTime_{i+1} - ExecutionTime_i)} \tag{9}$$

where $Progress_i$ is the $i$th measurement of task execution ranges from 0 to 1. They calculate the Approximate-time-to-Finish (ATF) as follows,

$$ATF = \frac{(1 - Progress_i)}{TPS[t]} \tag{10}$$

and check if the ATF of the specified task tracker is shorter than the first ATF in the ATF list, then a speculative copy of the task will be launched; otherwise it will not launch any copy for the task.

ESAMR (Sun *et al*., 2012) scheduling algorithm is the proposed successor of SAMR (Chen *et al*., 2010) and LATE (Zaharia *et al*., 2008). It solves the issues of LATE and SAMR, such as LATE's approximation of remaining execution time of tasks not being accurate enough because of its usage of static information which leads to the incorrect classification of slower nodes in the system. ESAMR considers the type of job and input data size in combination with the node type in the calculation of weights for different phases of map and reduce tasks. It categorizes tasks on a node using k-means clustering and stores the weights of different phases of map and reduce tasks for each job class, which in turn is used for estimating time for the completion of each task. From the experimental results, ESAMR claims the lowest error rate in the estimation of time to completion, as compared to SAMR and LATE.

DRF (Ghodsi *et al*., 2011) was proposed to show highly desirable theoretical properties under Leontief preferences. In DRF, the dominant resource of an entity is the resource for which the entity's task requires the major fraction of the total availability. DRF aims to maximize the number of allocated tasks, under the constraints that the fraction of the allocated dominant resources (called dominant shares) are equalized. Slowdown Predictor Model (Bortnikov *et al*., 2012) is another environment-aware approach that improves the Hadoop performance through predicting straggler tasks before they are launched using machine learning. It uses smarter scheduling and avoids unnecessary speculation of tasks, which leads to efficient resource utilization.

#### *4.2.2 Locality-aware schedulers*

Data locality is a basic supposition of Hadoop MapReduce framework which critically affects its performance. Hadoop scheduler schedules map and reduce tasks to process data on their local nodes where the input data resides. Execution of tasks locally on the same nodes where the data resides moderates network

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

communication in the cluster in shuing the data across the nodes, which consequently improves the performance of the Hadoop system. The Hadoop system may consist of homogeneous or heterogeneous nodes. There are numerous researchers in academia and industry who consider the performance improvements of Hadoop on the basis of data distribution over the cluster. Purlieus (Palanisamy *et al*., 2011) proposed a resource allocation system that focuses on clusters of VMs such as Amazon's elastic compute cloud (EC2) (Amazon! 2016d). They developed an algorithm which improved data locality in the map and reduced phases for the reduction of Hadoop MapReduce network traffic. Their proposed scheme has the following techniques: they place map tasks locally for map-input heavy jobs while reduce tasks can be scheduled on any node in the cluster; schedule reduce tasks locally for the reduce-input heavy jobs; and for the Map-and-Reduce input heavy jobs, both the map and reduce tasks are scheduled in a set of narrowly linked nodes. It introduces a data assignment method in a cloud environment for dynamically assigning VMs to the users, avoiding data redundancy and loading time. It suggests the locality-based cloud service provisioning using VMs placement in an autonomous way. It claims about a 70% reduction in network traffic and a 50% improvement in response time or execution time.

MapReduce Adapted Algorithms to Heterogeneous Environments (MRA++) (Anjos *et al*., 2015) is a new heterogeneity-aware data distribution, task scheduling, and job control mechanism based MapReduce framework design. It uses training tasks to collect information before the distribution of data over the cluster. It adjusts the amount of data processed during the map phase to the distributed computing capability of the nodes. The intermediate data for the Reducer is partitioned into smaller sizes based on the sum of the granularity factors, i.e., the faster nodes process more data than other slower nodes. Owing to the reduction of intermediate data partition sizes, which increases the number of executed tasks and data granularity, a global reduce task queue controls the execution and provides the available nodes with data distribution information for its processing. MRA++ has a higher degree of parallelism for the tasks and higher improvement in the job response time. It achieves a performance gain of approximately 70% in 10-Mbps networks over the native Hadoop system.

The Locality and Network-Aware Reduce Task Scheduling (LoNARS) (Arslan *et al*., 2014) algorithm is proposed for the scheduling of reduce tasks in a Hadoop MapReduce system. It takes data locality and network traffic into consideration while making the scheduling decision. It attempts to schedule the reduce tasks near the map tasks in the system through the data locality approach which implicitly decreases the network traffic. Through network traffic awareness, it distributes the traffic over the whole network and condenses the hot-spots to minimize the effect of the network bottleneck in the Shue phase of Hadoop MapReduce job runtime. LoNARS provides an improvement of approximately 15% in network traffic reduction and 3–4% in total job completion time. Multithreading Task Scheduler (MTL) (Althebyan *et al*., 2014) uses a multi-threading technique to schedule the tasks to different data blocks where each thread is assigned to a predetermined block of jobs. It divides the whole cluster into $N$ blocks where each block's job in the wait queue is scheduled by a specified thread. Each thread searches for the local data block upon the arrival of a job into the ready queue. Threads receive a notification from other threads upon finding the local data for the jobs; hence, other threads stop searching. If none of the threads find a local data block for a task, then that task is scheduled on a non-local node. It acquires its processing advantage through parallel searches by different threads instead of single-threaded processes in similar schedulers. Bi-Hadoop (Yu & Hong 2013) is another locality-based approach which supports dual-map-input applications and attempts to minimize the network resource congestion for such an application.

FIFO with Shareability and Locality-Aware (Wei *et al*., 2015) scheduling algorithm is a FIFO-based scheduling policy that takes the locality and data sharing probability between tasks into account for its scheduling decisions. The main objective is to gather the tasks that require the same data, batch process them, and reduce the overhead of shuing the data over cluster nodes. The Shared-Input-Policy (Bezerra *et al*., 2013) scheduling algorithm chooses tasks from a waiting queue by investigating the available resources of the cluster to improve cluster performance. Although it cannot predict the resource consumption of waiting jobs, it appropriately submits tasks to the cluster by analyzing the available resources in the cluster.

A variation of the Delay Scheduling algorithm called the Matchmaking (He *et al*., 2011) algorithm has been proposed to schedule map tasks to improve data locality in the Hadoop system. It provides a fair

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

chance of executing a data-local task to all nodes before launching a non-local task, i.e., it delays the execution on a node until it receives a data-local task instead of delaying a job execution. It always prefers the local map task over non-local tasks. It uses a locality marker for node classification and ensures that all nodes will have a fair chance to seize its local tasks. This has been achieved through relaxation of job scheduling to improve the performance by avoiding data transfer in map tasks.

Hadoop performance in the heterogeneous environment degrades in comparison with a homogeneous cluster. Heterogeneous environment Locality-Aware Scheduler (Zhang *et al*., 2011) addressed the issues of heterogeneity and locality and proposed a solution to simulate optimal task execution time. It is based on maintaining a tradeoff between task waiting time and transmission time. It first attempts to respond to the Task Tracker request for a task with a task at a local DataNode. In case such task is non-existent, it attempts to assign a task with data closest to the requesting node. It makes this decision based on the metric whether or not waiting time for a task is less than transmission time.

Data-local Reduce Task Scheduler (Geetha *et al*., 2016) proposed to schedule tasks on nodes that tend to result in the least number of bytes shued. This scheduler exploits the index file and heartbeat protocol to gather the information about each map task's output data stored in the local file system of each Task Tracker node and maintains a data structure of it. For each reduce task ready to be scheduled, the scheduler checks if the requesting Task Tracker is the one that contains the largest amount of intermediate output data. If that is the case, the reduce task is scheduled to that Task Tracker. In other cases, the next requesting Task Tracker is considered, and so on. The modified Job Tracker maintains the data structure that ensures the Task Trackers with the largest intermediate output must run the reducer. Their scheduler minimizes the data-local traffic by 11–80% considering the varying number of mappers and reducers.

USSOP (Su *et al*., 2011) is a grid-enabled MapReduce framework which provides a set of C-Language based MapReduce APIs and a runtime system for exploiting the computing resources available on public resource grids. The authors proposed Locality-Aware Reduce Scheduling (LARS), specifically designed to minimize data transfer in USSOP. The intermediate data are hashed and stored as regions, as one region may contain different keys. Master node assigns reduce tasks to the grid nodes using the LARS algorithm until all maps are completed, i.e., nodes with the largest region size will be assigned reduce tasks. In this way, LARS avoids transferring large regions out to other nodes. It works better in heterogeneous grid environments. In another research effort, authors of the Minimum Network Resource Consumption (MNRC) (Shang *et al*., 2017) model took the liberty of taking two main data streams of slow tasks migration and remote copies of data in the cluster network into consideration using the use case of Hadoop MapReduce clusters. MNRC is used to calculate network resources consumption of reduce tasks to minimize the amount of remote copies of data, in combination with data locality, using their priority scheduling policy for the Reduce task.

In Hadoop, the input data of map tasks are transferred in numerous small parts. Mappers process the first part of the input data upon receiving it and wait until its execution is finished. From there, the next part is transferred and so on, which is a waste of time because data processing and transmission can be carried out in parallel. Improving MapReduce by Data Prefetching (IMPDP) (Gu *et al*., 2013) introduces a data prefetching mechanism for map tasks using intra-block prefetching in heterogeneous or shared environments. The main objective is to reduce the overhead of data transmission over the network. IMPDP creates a data prefetching thread which requests non-local input data and a prefetching buffer is used to store the temporary data for the task. The data prefetching thread pulls the input data through the network and stores it in the prefetching buffer temporarily, and the map task reads the data from this buffer to process. Experimental results show an approximate 15% improvement in job response time.

#### *4.2.3 Situation-aware schedulers*

The situation-aware schedulers are those schedulers which can adopt to different situations among task communications, map execution, shuing, and combining. A number of researchers address situation-awareness in their work in the area of Hadoop optimizations. Certain significant research approaches include Preemption-based Scheduler (Polo *et al*., 2010), Map Optimizer (Yoo & Sim 2011), Acclimate MapReduce (Acclimate MR) (Balmin & Beyer n.d.), MOMTH (Nita *et al*., 2015), and Speculative Aware

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Resource Allocation (SARA) (Ren 2015). These research approaches are improving the native Hadoop performance by adapting to situations and using different heuristics. A preemption-based scheduler (Polo *et al*., 2010) introduces an online job completion time estimator which can be used for fine-tuning the resource provisioning of different jobs. The main objective is performance-driven co-task scheduling in Hadoop MapReduce runtime to provide better resource utilization in a shared environment. The goal is the exploitation of MapReduce's multiple small tasks capability for the prediction of tasks completion time and characteristics of the wait queue tasks. The fine-tuning of resource provisioning among jobs is based on the estimation of the completion time for individual jobs based on a given set of resources. It also has the capability of preempting other low priority jobs to provision the resources to the high priority job. It allows applications to meet their performance goals without the over-provisioning of physical resources. The scheduler has two strategies to provision resources: a max-scheduler approach, in which all the resources of the Hadoop cluster are allocated when there are enough tasks; and a min-scheduler approach, in which the completion time goal for each job is wisely observed and resources are freed, if possible. Their evaluation is based on four things : job completion time, job with deadlines minimum scheduler, job with deadlines maximum scheduler, and comparison with Facebook's Fair scheduler.

MOMTH (Nita *et al*., 2015) expounds on a multi-purpose scheduling algorithm for numerous tasks in the Apache Hadoop big data processing framework. They consider the fine grain functions of users and resources with the constraints of deadline and budget simultaneously. MOMTH addresses the most relevant constraints for Hadoop (i.e., avoiding resource conflict and having a near-optimal cluster workload) and relevant objectives (deadline and budget). This type of scheduler is beneficial when there are cost and time limitations. The authors used a Hadoop integrated tool scheduling load simulator for the evaluation of this research work. They analyzed the performance of MOMTH using MobiWay, which is a collaboration podium to expose the interoperability between a huge number of sensing mobile devices and an extensive variety of mobility applications. SARA (Ren 2015) is a distributed and decentralized speculation-aware cluster scheduler. The speculative execution of tasks is a very effective method to mitigate the impact of stragglers, thus far. However, schedulers have to decide between the scheduling of speculative copies of some jobs and original tasks execution of other jobs. They define two guidelines for selecting which tasks should be scheduled for execution in the cluster. The first guideline states that: In the absence of enough slots for every job to maintain its anticipated level of speculation, slots should be dedicated to the smallest jobs, and each job should be given a specified number of slots equal to its virtual size to achieve near-optimal performance. The second guideline states that: In the presence of enough slots to permit every job to maintain its desired level of speculation, slots should be shared proportionally to the virtual sizes of jobs to achieve a near-optimal throughput.

Acclimate MR (Balmin & Beyer n.d.) is one of the situation-aware schedulers using the principle of 'situation awareness in mappers'. It changes the native Hadoop assumption about independent mappers by implementing the mappers which communicate with each other through Distributed Meta Data Store. Unlike the native Hadoop system, the mappers in Acclimate MR communicate for situation awareness. The mappers take a data block for the diminution of initial starting time, Acclimate MR combiner performs local aggregation and caches frequently appearing keys, and their sampling and partitioning (which sample intermediate data and provide well-balanced input to reducers) assists in handling the situation dynamically. The authors claimed evaluation shows that adaptive techniques provide up to three-time performance improvement over native Hadoop, and exponentially improve performance stability across the board.

#### *4.2.4 Replica-aware schedulers*

The Hadoop framework uses data replication for the availability and reliability of the system. By default, the replication factor is three, which can be modified using the replication property in the Hadoop configuration file. Replication-awareness is the property of the scheduler which is aware of and changes the behavior of the scheduling policy while taking the number of replicas and distribution information of each replica throughout the cluster. Maestro (Ibrahim *et al*., 2012) is a probability-based, fine-grained, replica-aware algorithm to alleviate the non-local Map tasks execution problem that relies on the replica-aware

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

execution of Map tasks. It stores the number of unprocessed data blocks in each node j as $HCN_j$ to keep track of all unprocessed data blocks in the cluster. It then calculates the probability of processing a data block non-locally as block weight ($WB_i$) using the following equation:

$$WB_i = 1 - \sum_{1}^{r} [\frac{1}{HCN_j}] \tag{11}$$

where $HCN_j$ represents the number of blocks of data stored in node j of the cluster. Using the replication feature of Hadoop, nodes that share some blocks as replicas present in both the nodes. ShareRate is the number of data blocks a node shares with other nodes and can be calculated as,

$$ShareRateN_j = \max_{1 \le i \le N} Sb_j \tag{12}$$

where $Sb_j$ is the number of shared blocks between $N_i$ and $N_j$. The scheduler schedules Map task in two phases using the above information: (a) It attempts to ensure Map task executes on all nodes locally. For that reason, it assigns the node-local tasks for blocks with the highest block weights to various nodes in descending order of their shares in the job initialization phase; (b) Upon availability of container at runtime, it schedules an unprocessed block with the highest block weight to minimize the non-local map tasks execution. Maestro demonstrates a 95% improvement in the speculative execution of data local map tasks and a 34% improvement in execution time.

Optimal Task Selection (Suresh & Gopalan 2014) scheduling algorithm is an optimal task selection scheme to assist the scheduler in case multiple local tasks are available for a node. The main objective is to launch the task with the least number of replicas of input data, individual load of disks attached to the node, and the maximum expected waiting time for the next local node to improve the probability of percentage of local tasks launched for a job in future. It shows an improvement of approximately 20% in terms of locality over the HDFS and a significant improvement in fairness without affecting the locality when used with delay scheduling.

Multiple-FIFO (Jin *et al*., 2012) based scheduling is an optimization of the native Hadoop's map assignment mechanism. The Hadoop works on the principle of 'moving computation to data'; hence, the map task assignments to the free slots on the data node holding the data become a key issue of research. Multiple-FIFO is a two-step technique to optimize the map task assignment mechanism in homogeneous Hadoop clusters. Its technique of scheduling works on: (a) data locality scheduling tactics and (b) replica selection methodologies. It introduces multiple queues in the MapReduce framework to overcome the limitations of FIFO scheduler by their data locality scheduling tactics. Using replica selection methodologies, it estimates the load on nodes for the load balancing in the system using the following equation.

$$load = A + m_i * B + r_i * C \tag{13}$$

where $A$ is the OS overhead, $m_i$ is the $i$th map task, $B$ is the combined overhead of map tasks, $r_i$ is the $i$th reduce task, and $C$ denotes the combined overhead of all reduce tasks in the system. They claimed approximately a 2.5-times performance gain over the native Hadoop FIFO scheduler.

#### *4.2.5 Network-aware schedulers*

Network-awareness is the property of schedulers to take network traffic information into account while making scheduling decisions. In the Hadoop framework, the network becomes a bottleneck in most situations compared to other resources owing to the shuing of intermediate data over the network. This all-to-all communication in the Shue phase creates the network congestion and degrades the performance of the whole system. Owing to these reasons, adding network awareness to the scheduler to optimize the Hadoop system has become a popular research topic in the last few years. Numerous researchers from academia and industry contributed in the area by adding network-awareness to the native Hadoop system through different techniques to enhance the overall performance of the Hadoop ecosystem. We discuss the key efforts in this area in the following paragraphs in this section.

Recently, Hadoop MapReduce clusters are usually shared among multiple users and jobs for better resource utilization. Job performance in multi-tenant Hadoop cluster is greatly impacted by all-to-all communication in the Shue phase, which leads to network blockage due to heavy network traffic.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

ShueWatcher (Ahmad *et al*., 2014) is a multi-tenant Hadoop scheduler that minimizes network traffic in the Shue phase while maintaining the particular fairness constraints. It works on the basis of three key techniques. First, it limits intra-job map-shue concurrency to shape network traffic in the Shue phase by procrastinating the Shue phase of a job according to the network traffic load. Second, it favorably allocates a job's Map tasks to localize the intermediate data to as few nodes as possible by exploiting the minimized intra-job concurrency and flexibility prompted by the replication of map input data for fault tolerance. Third, it exploits a localized map output and delayed shue to reduce the network traffic in Shue phase by preferentially scheduling a job's Reduce tasks in the nodes holding the intermediate data. It claims a performance gain of 39–46% in cluster throughput and 27–32% in job response time. ShueWatcher works well in shared environment situations, e.g., an overlapping map with Shue phase inter-jobs rather than intra-jobs and adjusting intra-job concurrency for minimized network traffic in the Shue phase.

The BAlance-Reduce (BAR) (Jin *et al*., 2011) scheduling algorithm introduced a heuristic task scheduling technique which takes a global view and adjusts the data locality of map tasks, vigorously conferring to the network traffic condition and cluster workload. The main objective is to iteratively schedule map tasks to improve the locality of input data, which leads to an improvement in job performance. At the first iteration, the scheduler's aim is to schedule tasks to local node slots. It considers rack-locality and off-rack locality (non-local execution of tasks) in succeeding iterations. In the case of a poor network situation and overloaded cluster environment, BAR attempts to adjust data locality to enhance resource optimization.

Data locality and the reduce tasks scheduling and partitioning skew may excessively degrade Hadoop performance. The Minimum Transmission Cost Reduce Task Scheduler (MTCRS) (Tang *et al*., 2015) proposed a sampling evaluation to solve the issues of partitioning skew and locality of intermediate data for the reduce tasks in the Hadoop system. It uses the transmission cost and waiting time of each reduce task as heuristics for decisions about scheduling reduce tasks in the cluster. It uses the Average Reservoir Sampling algorithm to generate the parameters as sizes and locations of intermediate data partitions to use in a mathematical model for estimating transmission cost as follows,

$$C_j(U, v) = \sum_{i=1}^{i=U} P(i,j) * Dis(U_i, v) \tag{14}$$

where $C_j(U, v)$ symbolizes the estimated transmission cost, $P(i, j)$ denotes partition proportion, and $Dis(U_i, v)$ is the hop distance between node u and node $v$. $U$ represents the set of nodes containing the intermediate data for a job. This model is used to ultimately calculate the best reduce task launching node in the cluster. MTCRS minimizes approximately 8.4% of the network traffic in the cluster in comparison with Fair scheduler.

High Performance MapReduce Engine (HPMR) (Seo *et al*., 2009) introduced two optimization schemes, prefetching and pre-shuing, that improve the overall performance in a multi-tenant environment while holding compatibility with the default Apache Hadoop framework. The prefetching exploits the data locality, and pre-shuing aims to minimize the network overhead in all-to-all communication during the Shue phase. The newly introduced prefetching mechanism has two sub-modules. The first one is for the intra-block prefetching to pre-fetch data within a single data block while performing a processing or complex computation. The second is inter-block prefetching to pre-fetch the entire block of replica to a local rack before the processing of that block. The pre-shuing mechanism changes the scheduler to check the map input split and predicts the partitioning of key-value pairs while taking the reducer locations into consideration. The main objective is to schedule map tasks near the future reducer before the map tasks execution. The evaluation results using Yahoo! Grid show a performance gain of approximately 73% in response time over the native Hadoop system.

The default Hadoop scheduler neither considers the data locality of reduce tasks nor addresses the load unbalance among map and reduce tasks, which causes performance degradation. Hammoud & Sakr (2011) proposed Locality-Aware Reduce Task Scheduling (LARTS), a technique that uses sizes and location of partitions to exploit data locality, i.e., combines data locality with a reduce scheduling policy. It attempts to schedule reducers close to their maximum amount of intermediate input data and conservatively switches

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

to a relaxation strategy seeking a trade-off among the scheduling delay, resource utilization, scheduling skew, and degree of parallelism. It outperforms the native Hadoop performance by an average of 7%. NW-Aware Scheduler (Kondikoppa *et al*., 2012) introduced a network-aware scheduling algorithm for Hadoop MapReduce system. The transferring of data over the network is costly and causes performance degradation more severely in federated clusters. The authors designed and implemented a network-aware scheduler to be used on federated clusters, improving map tasks scheduling in such environments and accordingly minimized the network traffic overhead leading to an improved performance.

In the Hadoop MapReduce system, if the node creating the intermediate data for a given reduce task is not close to it, then a large amount of data transferring through the network is required to get data to the reduce node, resulting in high network traffic. Another issue is that of the partitioning skew that ascends owing to an unbalanced distribution of map output across nodes, causing a huge amount of input data for some reduce tasks. Both have a negative impact on the application performance. To address these problems, the Centre-of-Gravity Reduce Scheduler (CoGRS) (Hammoud *et al*., 2012) reduce scheduling algorithm is introduced which adds locality and skew awareness to the scheduler. CoGRS schedules reduce tasks to the node based on location closeness to the nodes creating intermediate data for it. They used Weighted-Total-Network-Distance ($WTND_r$) for reduce 'r' as a heuristic for identifying the appropriate node for the reduce task, and is calculated using the following equation,

$$WTND_r = \sum_{i=0}^{n} (ND_i * r * w_i) \tag{15}$$

where '$n$' symbolizes number of partitions that are input to $Reducer_r$, $ND_i$ denotes network distance to transfer partition i to $Reducer_r$ in the shuffle phase, and $w_i$ is weight assigned to i. In the CoGRS algorithm, the node with the minimum $WTND_r$ is considered to be the Center of Gravity node and the reduce task $r$ is scheduled to it. CoGRS reduce scheduler reduces the network traffic and job response time by approximately 9% and 23%, respectively, as compared to native Hadoop.

#### *4.2.6 Priority schedulers*

Priority scheduling is a technique for scheduling progressions based on priority. It involves assigning priorities to different tasks or jobs and then processing them based on the assigned priorities. Hadoop MapReduce has priority based schedulers developed by different researchers. We will explain the significant schedulers in this subsection of the paper. The Task Co-Scheduler (Polo *et al*., 2010) presented performance-ambitious co-task allocation in Hadoop MapReduce for better resource utilization in a shared environment. The authors exploit Hadoop MapReduce's several small task's capability to forecast the completion time for the waiting tasks. Their resource allocation tuning methodology among jobs is centered on the estimated completion time for each job based on some given resources.

Gu *et al*. (2014) introduced Priority scheduler, a priority-based scheduling algorithm for the Hadoop MapReduce framework. They used two parameters as heuristics: the priority of slave nodes, and the distance between the master and compute node. The amount of total data to be processed by the job can be calculated as,

$$S = \sum_{j=1}^{n} AvgTaskSize_j * WeightValue_j \tag{16}$$

where $S$ is the total amount of data to be processed, $AvgTaskSize_j$ is the average size of a task in the job j, and $WeightValue_j$ indicates the number of tasks based on job weights. The performance evaluation shows that the improved scheduler based on priority minimizes data transfer in the Hadoop cluster, and the response time of jobs.

#### *4.2.7 Energy-aware schedulers*

Energy consumption is one of the most important issues in big data and modern world data centers. To handle this issue, certain researchers developed scheduling systems or cluster management systems which

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

take energy consumption into account. Energy-aware scheduling (EAS) techniques are those which take energy consumption of the cluster into account while allocating tasks to different nodes or making scheduling decisions. Several researchers developed energy-aware schedulers for Hadoop MapReduce frameworks, including EA-MapReduce (Li *et al*., 2011), MT-MapRed (Althebyan *et al*., 2014), and EAS (Liang *et al*., 2013). EA-MapReduce (Li *et al*., 2011) is an energy-aware scheduler which predicts the energy consumption of MapReduce workloads in the Hadoop cluster. Li *et al*., (2011) used a multivariate linear regression model to analyze the results of the benchmark traces after execution. They used number of instructions, map or reduce tasks written, bytes, etc., and introduced a linear prediction model based on these metrics. Using Word Count and Sort workload with various input data sizes, their evaluation results claim that their prediction model shows an approximately 0.15% inaccuracy in comparison to observed energy consumption.

Liang *et al*. (2013) introduced EAS, an energy-aware vibrant scheduling algorithm that aims at minimizing communication energy utilization through clustering those tasks whose executions depend on each other. The current static schedulers may cause an increase in energy consumption owing to task waiting time, as an estimation of the task's execution time is difficult. The EAS dynamic scheduling technique tunes the clustering group on the basis of an energy consumption threshold.

#### *4.2.8 Load/fairness-aware schedulers*

Load-awareness is the property of the scheduling algorithm which takes the load of the nodes into account while making scheduling decisions in the cluster. Fairness-awareness is the property of schedulers to make scheduling decisions on the basis of Fairness among nodes throughout the cluster or users. Fairness and load-balancing are very important issues in Hadoop scheduling. There are various efforts to optimize Hadoop scheduling while solving these issues. Chintapalli (2014) introduced CRBalancer, a mechanism which distributes input data splits to heterogeneous nodes in the Hadoop cluster according to their computing power capabilities. The main objective is to profile all the nodes in the cluster to identify their computing capabilities, and then balance the load in each node according to their resource capabilities. The balancer first obtains the network topological information and determines the over-utilized and under-utilized nodes to perform load-balancing among them.

In the Hadoop system, by default data blocks are replicated in three different nodes. In this situation, when the data blocks can be found in more than one location in the cluster, a challenging issue is determining which node is assigned the task. To solve this problem, Chen *et al*., (2013) introduced Locality-Aware Scheduling Algorithm (LaSA), a method that assigns weight to data which is input to the specified task and computes the node weight for data interference accordingly to estimate the weight of data in the specified node. Next, it sorts the nodes in ascending order and schedules the task to the first node in the list to minimize data transmission time for performance optimization. They produced two works for optimizing the default Hadoop scheduler: (a) they introduced a mathematical model of data interference weight in the Hadoop scheduler and (b) they designed and implemented the LaSA algorithm to exploit the weight of data interference to afford data locality-aware resource scheduling in the Hadoop system.

By default, Hadoop does an All-Map-to-All-Reduce communication model in the Shue phase. This strategy usually results in the saturation of network bandwidth while shuing intermediate data from mappers to reducers, called Reducers Placement Problem (RPP). There are several research efforts to improve the performance of Hadoop MapReduce by altering the data flow in transition between mappers and reducers. The Locality-Aware Fairness-Aware Key partitioning (LEEN) (Ibrahim *et al*., 2010) algorithm is a research effort to address the problem of efficiently partitioning the intermediate keys to minimize the amount of shued data over the network. The accurate partitioning of intermediate data also solves the issue of the RPP. LEEN attenuates the partitioning skew, minimizing data transfer by balancing the data distribution among nodes in the cluster. It also improves the data locality of MapReduce execution efficiency with the use of asynchronous Map and Reduce execution policy. It guarantees a fair distribution of intermediate data throughout all the reducers in the cluster. It attains an approximately 40% improvement on different workloads.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

The X-Flex (Wolf *et al*., 2014) is a cross-platform scheduler which is developed as an alternative to the DRF (Ghodsi *et al*., 2011) scheduler currently part of both YARN (Douglas *et al*., n.d.) and Mesos (Hindman *et al*., 2011). X-Flex screens instantaneous fairness to grasp the long-term view of the situation. Unlike DRF, the packing of containers into processing nodes is done oine for the improvement of packing quality. X-Flex takes into perspective that some frameworks have adequate structure to make sophisticated scheduling decisions. Therefore, it allows this , and also gives platforms an inordinate amount of freedom over the degree of sharing they will permit with other platforms. X-Flex has considerable qualitative and quantitative advantages over DRF, including durable view of fairness, an apparently more appropriate description of instantaneous fairness, an arithmetically refined off-line vector packing system to produce containers and their owners, flexibility, work with framework-specific scheduling algorithm that can take advantage of the inherent structure, and the ability of applications to share as much or as little as needed.

Optimal Task Selection (Suresh & Gopalan 2014) scheduling algorithm introduced an optimal task selection method to assist the scheduler in the case of multiple local tasks availability for a node. The authors select the task with the least number of replicas for execution to improve the probability percentage of a local task being launched for a job in the future and maintain load balancing in the nodes throughout the cluster. Node-Aware Locality-Aware and Fairness-Aware (NoLFA) (Hanif & Lee 2016), an efficient key partitioning algorithm, is a recent variance of the LEEN (Ibrahim *et al*., 2010) algorithm which considers the heterogeneity of the nodes throughout the cluster. NoLFA takes node capabilities as heuristics while making its scheduling decisions to obtain the best available trade-off between the locality and fairness in the system. It achieves approximately a 29% improvement over the native Hadoop scheduler.

#### *4.2.9 Hybrid schedulers*

Hybrid schedulers are schedulers that cover both the job and tasks levels while taking the scheduling decisions. These are the most powerful schedulers because they take both coarse-grain and fine-grain scheduling metrics into account when scheduling tasks or jobs to different nodes in the cluster. The broad adoption and universal usage of Hadoop has strained the initial design of Hadoop version 1.0 well beyond its intended goal, uncovering two key limitations: (a) a tightly coupled mechanism of a specified programming model with the resource management organization, compelling developers to abuse the MapReduce programming model and (b) a consolidated handling of the job control flow resulting in endless scalability concerns for the scheduler. YARN (Douglas *et al*., n.d.) is the new version of Hadoop scheduler which decouples the resource manager from the job scheduler. In this new architecture, the resources are management responsibilities which are given to Resource Manager while the scheduling obligation is handled by the Application Manager which handles all the running and ready applications in the cluster. With the decoupling of the programming model from the resource management infrastructure, YARN became a general-purpose resource management system for the cluster.

Conventionally, each organization has its own isolated cluster that has a satisfactory capacity to meet the organization's SLA under near peak conditions, which leads to poor utilization of resources. In contrast, sharing a cluster among organizations is a cost-effective policy for running large Hadoop clusters. However, organizations are concerned about the sharing policies of resources so that other organizations should not use their SLA-critical resources at a pivotal time. An engineering group of Yahoo! Researchers proposed Capacity Scheduler (Apache! 2015a), a pluggable Hadoop MapReduce scheduler for multi-tenant sharing clusters where resources are provisioned to applications in a timely manner under the allocated capabilities constraints. The main objective is that the available resources of the cluster should be partitioned among multiple organizations or tenants. The excess capacity can be utilized by a needy organization which is elasticity in a cost-effective manner. It supports multiple job queues, and each has an allocated fraction of the cluster's resources. Administrators can configure soft and hard limits on the capacity allocated to each queue. It supports job priorities within the queue which is disabled by default and users can enable it optionally. Users can submit jobs to individual queues following strict ACLs for each queue. To ensure the security of users' jobs, they use safeguards so that users cannot view or modify each other's jobs.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Facebook developers introduced a new scheduling algorithm called Fair Scheduler, which is part of both Hadoop version 1.0 and YARN version 2.0. Fair Scheduler (Apache! 2015b) aims to provide a fair share of resources to every job throughout the runtime. The system has different pools of a guaranteed minimum number of slots which accepts jobs from the users. Free superfluous slots in a pool are shared among jobs while the idle pool's resources are shared among other pools. In case of an under-capacity pool needing its share back, the over-capacity pool been preempted to maintain the minimum share among the pools. A job utilizes all the cluster in the case of solo execution, while the resources are divided fairly among the jobs in case of the arrival of new job submissions by users. Fair scheduler allows the short jobs to run within a reasonable period of time and simultaneously, and do not allow the long jobs to starve (Bincy & Binu 2013). It supports priority scheduling, and the administrator has been given the power of priority enforcement on certain pools. In the case of priority scheduling enabled, the tasks scheduling changes to an interleaved policy of execution based on their assigned priority within the specified pool considering the allocated and minimum share of cluster resources to the specified pool. Fair scheduler keeps track of the ratio of the amount of time resources actually used by a job to the ideal fair allocation of resources to the job. The job with the highest ratio tasks is scheduled accordingly, which ensures fair resource utilization.

Phan *et al*. (2010) introduced real-time scheduler, a formal model for capturing real-time Hadoop MapReduce applications in Hadoop clusters. It focused on task scheduling optimization in a heterogeneous Hadoop cluster by transforming the problem as a Constraints Satisfaction Problem through identifying key factors affecting real-time scheduling, including data placement, master scheduling overhead, and concurrent users. Heterogeneity-Aware Resource Allocation Scheduler (HARAS ) (Lee *et al*., 2011) proves that some resource may have poor job performance with high availability and other resources may have high job performance with low availability. For better performance, the resources should be scheduled on different and separate resources to ensure that each job receives a fair share of resources. The architectural design is to provision resources to a data analytic cluster in the cloud and suggest a metric of share in a heterogeneous cluster to comprehend a scheduling scheme that attains high performance and fairness.

There are numerous unique challenges at Facebook, including scalability (the largest cluster has more than 100 PB of data), processing needs (more than 60,000 Hive queries per day), and their data warehouse has grown by a factor of 2500-times in the last 5–6 years (growth is expected through increasing the user base and the ongoing addition of new features to the site). By early 2011, they began reaching the limits of the Hadoop MapReduce system; therefore, they developed Facebook-CM (Corona) (Facebook! 2015) which is a new scheduling framework that separates cluster resource management from job coordination. Corona proposed a cluster manager that keeps track of the amount of free resources and total working nodes in the cluster. Each job has a committed job tracker which can either run in the same process as the client (for small jobs) or as an isolated process in the cluster (for long jobs). Corona uses push-based scheduling, i.e., cluster manager pushes back the resource allowances to the job tracker upon receiving the resource request. The scheduling in Corona does not involve any periodic heartbeat messages leading to a reduced scheduling latency. The Corona cluster manager has its own fair-share scheduler which considers a full snapshot of clusters and jobs while making its scheduling decisions, which leads to better fairness. It supports pool groups to group scheduling pools in a multi-tenant environment. Only the team which is assigned to that pool group is allowed to manage the pools within their group. This gives every team fine-grained control over their allotted resources.

### *4.3 Cloud-based schedulers*

Cloud-based schedulers are those schedulers that take advantage of the running environment, such as virtualized cluster, cloud-based infrastructure, and grid-based execution environments. Modern Hadoop clusters are mostly provisioned over a cloud-based system such as Amazon's EC2 (Amazon! 2016d). Some researchers made efforts in this area by introducing different Hadoop schedulers, including Cloud MapReduce (Liu 2011), VM-H Scheduler (Jiang & Sheng 2012), Cloud-HDFS (Ko & Cho 2009), and many others. We will be discussing some of them in this section of the paper.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

The use of distributed storage services presented by different cloud provider organizations (such as Amazon S3 (Amazon! 2016c), Elastic Block Storage (Amazon! 2016a), and Relational Database Service (Amazon! 2016b)) is a simple way to avoid data loss in the case of cloud-based computing clusters; however, storing all intermediate data in them rather than local disk is absurdly inefficient. Cloud MapReduce (Liu 2011) introduced MapReduce implementation capable of taking advantage of the spot market. Spot Cloud MapReduce can progress the computation regardless of the termination of an enormous number of nodes on a regular basis. The main objective is to process tasks using spot instances and send the intermediate data stored in a temporary staged buffer to the reduce queues asynchronously. Jiang and Sheng (2012) proposed VM-H Scheduler, a graph-based task scheduling algorithm to attain the minimum cost in a hybrid cloud environment. Their algorithm takes both public and private cloud resources into account to attain a minimum cost.

One important challenge when using Hadoop in a cloud environment is to manage intermediate data which are generated during dataflow computations, including MapReduce, Dryad, Pig, and Hive. They proved experimentally that the existing local-write remote-read solution (such as HDFS) and support from transport protocols (e.g., TCP-Nice) cannot guarantee both data availability and minimal interference. Ko & Cho 2009 presented Cloud-HDFS, an Intermediate Storage System that considers intermediate data a first-class citizen of dataflow programs. Their system considers a master-slave architecture with three different replication schemes: (a) replication using spare bandwidth, (b) deadline-based replication, and (c) cost model replication. In (Gulati *et al*., 2011), a group of researchers from VMWare Inc. elaborated on some of the issues found in most VM management softwares including their own VMware DRS cluster, Eucalyptus, and Microsoft PRO. The problems are scalability, heterogeneity, islands of resources caused by network connectivity, storage, and a limited scale of storage resources. They suggested three different scaling techniques to solve the above problems in VM management softwares: (a) Hierarchical Scaling, (b) Flat-Scaling, and (c) Statistical Scaling.

Dynamic Resource Reconfiguration (DRR) (Park *et al*., 2012) provided an imperative contribution to virtualization techniques in virtualized clusters. DRR was developed for distributed data-intensive frameworks in a virtualized cloud-based Hadoop cluster. The dynamic reconfiguration of different VMs for the enhancement of data locality in a virtualized Hadoop cluster improves the overall MapReduce job throughput. To make it possible for a VM to run local tasks, DRR temporarily increases its number of cores. It also makes its scheduling decisions based on locality and adjusts the computational power of virtual nodes according to the load after accommodating the scheduled tasks. The central principle behind the DRR is that 'each node should have the ability to fine-tune according to the demanded resources from that node so that different resource requirements by different tasks or jobs do not cause virtual node underutilization'.

Data locality in Hadoop system is a critical factor impacting on the performance of Hadoop MapReduce applications. Nonetheless, legacy improvements of data locality in virtualized Hadoop employ two levels of distribution of data (VM level and physical node level) which is not effective. DSFvH (Sun *et al*., 2014) presented a flexible virtualized Hadoop system in which storage and computing nodes are placed in their respective VMs. The DSFvH task scheduling algorithm aims to improve data locality by migrating the computing VMs to the physical node hosting the storage VM, which holds the data replica for the scheduled task. The job profiler receives the information about the job in the job queue and passes that information to the task profiler. The task profiler collects the data locality information through the data locality detector and monitors the progress of the tasks through the task progress monitor module. The Migration Controller selects a destination for the migrating VM which contains the required replica of data. It outperforms the traditional virtualized Hadoop system by approximately 37% in terms of job response time.

In recent years, interest in running the Hadoop MapReduce application over the Internet has increased. However, the data distribution techniques used on the Internet to distribute the high volume of data as an input to the MapReduce applications are deficient and need to be reconsidered. Bruno & Ferreira (2014) proposed SCADAMAR, a BOINC compatible computing platform to enable the deployment of MapReduce applications over the Internet. SCADMAR allows nodes to help distribute input datasets, intermediate data, and final output data through BitTorrent protocol instead of point-to-point protocols

| Scheduling Technique | Taxonomy | Operating Environment | Situation Awareness | Constraints | Best For Situations | Data Locality | Data Replication | Scope |
|---|---|---|---|---|---|---|---|---|
| FIFO | Non- Adaptive | Homogeneous | | Priority | Short Jobs | ⊗ | ⊗ | Both |
| MU Job Scheduler | Non- Adaptive | Nil | | Fairness | Multi-tenancy | ✓ | ⊗ | Job Level |
| Job Aware | Nil | Nil | ✓ | Nil | Low Latency | ⊗ | ✓ | Both |
| DP Share | Adaptive | Nil | ✓ | Fairness | Priority | ⊗ | ⊗ | Job Level |
| DS Speculative | Adaptive | Nil | | Rework | Skew | ⊗ | ✓ | Task Level |
| Job Scheduler | Adaptive | Homogeneous | | Nil | Node Failure | ⊗ | ⊗ | Job Level |
| Delay Scheduler | Non- Adaptive | Nil | | Time | Shared Cluster | ✓ | ✓ | Both |
| LATE | Adaptive | Heterogeneous | ✓ | Nil | Slow Tasks | ⊗ | ⊗ | Task Level |
| SAMR | Adaptive | Heterogeneous | ✓ | Nil | Slow Tasks | ⊗ | ⊗ | Task Level |
| Acclimate MR | Adaptive | Nil | | Variation | Dynamic Situations | ⊗ | ⊗ | Task Level |
| Maestro | Adaptive | Nil | ✓ | | Shared Data | ⊗ | ✓ | Task Level |
| D-Scheduler | Adaptive | Nil | | Deadline | Deadline | ⊗ | ⊗ | Job Level |
| Multiple-FIFO | Non- Adaptive | Homogeneous | | | Shared Data | ✓ | ✓ | Task Level |

**Figure 8** Taxonomy of scheduling techniques

(such as FTP and HTTP); applies the benefits of the nodes network bandwidth to distribute data; and automatically replicates data to reduce the risk of intermediate data loss.

## 5 Discussion and analysis

This research effort discussed various scheduling algorithms and techniques proposed to optimize the default scheduling mechanism of Hadoop MapReduce. There are numerous scheduling approaches used in a distributed computing system which cannot be used in the Hadoop framework because of the unique processing mechanism of moving computation to make use of data locality. The Hadoop MapReduce scheduling system has special attributes, such as loosely coupled data nodes, and each node functions as an independent data-intensive compute node, which attracted researchers from both academia and industry to develop new schedulers for the framework. These research projects aim to optimize Hadoop scheduling at the job level, task level, or a combination of the two. The issues addressed in job level optimizations include proportional sharing of resources dynamically, better user experience with fair scheduling, and constraints-based scheduling. There are different problems related to task scheduling such as data replication, data locality, network awareness, and heuristic-based scheduling which are dealt with by different task level schedulers.

Scheduling is considered critical to the Hadoop framework performance. In this sense, selected papers are explained and categorized on the basis of the issues they addressed. There are several research efforts, e.g., (Tian *et al*., 2009) and (Althebyan *et al*., 2014), which propose multi-queue scheduling techniques aimed at performance optimization. Other scientists use different approaches to achieve better performance results, including (Zaharia, Borthakur, *et al*., 2010), (He *et al*., 2011), (Hammoud & Sakr 2011), (Hammoud *et al*., 2012), and (Ibrahim *et al*., 2012), which take the optimal placement of Map and Reduce tasks into account. Some tackle the problem using historical data from the cluster nodes which allow diverse and unique techniques and methods, such as speculative execution of MapReduce tasks as in (Zaharia *et al*., 2008), (Jung & Nakazato 2014), and (Yoo & Sim 2011). Some papers including (Hammoud *et al*., 2012), (Tang *et al*., 2012), and (Park *et al*., 2012) present solutions covering important correlated areas, such as resource provisioning in cloud infrastructure, reflecting directly on Hadoop performance in such environments. Thus, scheduling in heterogeneous environments is also an important topic addressed by several research projects including (Tian *et al*., 2009), (Lee *et al*., 2011), (Anjos *et al*., 2015), and (Xie *et al*., 2010). Issues addressed by different researchers from both academia and industry in

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

the Hadoop scheduling mechanism include: deadline constraints in (Teng *et al*., 2014), (Morton *et al*., 2010), and (Kc & Anyanwu 2010), environment awareness in (Zaharia *et al*., 2008), (Chen *et al*., 2010), (Jung & Nakazato 2014), (Sun *et al*., 2012), (Bortnikov *et al*., 2012), and (Ghodsi *et al*., 2011); locality awareness in (Palanisamy *et al*., 2011), (Anjos *et al*., 2015), (Arslan *et al*., 2014) (Althebyan *et al*., 2014), (Bezerra *et al*., 2013), (Gu *et al*., 2013), (Wei *et al*., 2015), and (Yu & Hong 2013); network awareness in (Ahmad *et al*., 2014), (Kondikoppa *et al*., 2012), (Jin *et al*., 2011), (Tang *et al*., 2015), (Seo *et al*., 2009), (Hammoud & Sakr 2011), and (Hammoud *et al*., 2012); priority consideration while scheduling tasks and jobs in (Polo *et al*., 2010 and (Gu *et al*., 2014); and replication based optimization of the Hadoop schedulers as in (Ibrahim *et al*., 2012), (Suresh & Gopalan 2014), and (Jin *et al*., 2012).

Approximately 100 out of more than 200 research studies were selected in the area of Hadoop scheduling; the analysis and taxonomy of those studies are shown in Figure 8. The taxonomy column in the table shows the flexibility of the runtime of the algorithm to be adaptive or non-adaptive. An adaptive scheduler uses any previous, current, and/or future values of parameters while making the scheduling decision. Furthermore, a non-adaptive scheduler does not take any environmental changes into consideration while scheduling tasks and jobs in the cluster. The operating environment is the atmosphere considered by the algorithm while optimizing the native Hadoop scheduler. The situation awareness column indicates whether the algorithm considers the situation at runtime. Specific constraints considerations are presented in the constraints column. Best for situations or jobs column indicates the types of jobs, situations, types of tasks, or any other specified situations where the said scheduler best perform and favor the special tasks or jobs. Data locality and data replication columns show whether these issues are considered. The scope column indicates the scope of the scheduling algorithm in terms of task/job level.

## 6 Conclusion

The Apache Hadoop has become a de facto framework for huge distributed data-insensitive applications and has been adopted by both research communities and industry. One proof of this is the number of publications about the framework in recent years. Development and improvement in the area of Hadoop scheduling is a key research issue because of the static configuration-based current implementation of native Hadoop. We categorized the optimizations to the Hadoop system and created a taxonomy to help researchers observe the well-explored and more recently addressed issues. The issues related to the area of scheduling in Apache Hadoop framework have been explored and explained to assist users in understanding how to solve these issues. The classification is conducted based on the issues that were addressed, including heterogeneity, replication, data locality, resource provisioning, and availability. This review was conducted to assist in selecting promising areas for research in the Hadoop MapReduce framework. The storage or distributed file system, cloud, and distributed computing areas have outpaced the number of publications and optimization efforts in the last decade. Main-stream contributions are made in data flow tuning, tasks scheduling, jobs scheduling, and resource provisioning management.

## Appendix

*Key Terminologies*

- **JobTracker:** It is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster.
- **JobTracker Port:** Port where you can access the JobTracker. The default port might be different for each distribution.
- **NameNode**: It is the core of the HDFS file system. It maintains a record of all files stored on the Hadoop cluster.
- **Mapper Size:** The memory allocated to each mapper task that will launch on each of the Hadoop Nodes.
- **Parse:** The parse operation converts an in-memory raw data set (in CSV format, for example) into a HEX format data set. The parse operation takes a data set named by a Key as input, and produces a HEX format (Key, Value) output.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

- **YARN:** A resource-management platform responsible for managing compute resources in clusters and using them for scheduling of user's applications.
- **Hadoop Common:** Usually only referred to by programmers, it is a common utilities library that contains code to support some of the other modules within the Hadoop ecosystem. When Hive and HBase want to access HDFS, for example, they do so using JARs (Java archives), which are libraries of Java code stored in Hadoop Common.
- **HBase:** An open-source, distributed, versioned, non-relational database modeled after Google's Bigtable (Distributed Storage System for Structured Data).
- **Hive:** It is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. It allows user to query data using a SQL-like language called HiveQL (HQL).
- **HiveQL (HQL):** It is a SQL like query language for Hadoop, used to execute MapReduce jobs on HDFS.
- **NoSQL:** NoSQL or 'not only SQL' is a broad class of database management systems identified by non-adherence to the widely used relational database management system model. NoSQL databases are not built primarily on tables, and generally do not use SQL for data manipulation.
- **NewSQL:** It is an elegant, well-defined database system that is easier to learn and better than SQL. It is even newer than NoSQL.
- **Impala:** It is an SQL query engine with massive parallel processing power, running natively on the Hadoop framework. It shares the same flexible file system (HDFS), metadata, resource management, and security frameworks as used by other Hadoop ecosystem components.
- **Oozie:** It is a workflow engine for Hadoop.
- **Pig:** It is a high level programming language for creating MapReduce programs used within Hadoop.
- **Sqoop:** It is a tool designed to transfer data between Hadoop and relational databases.
- **Whirr:** It is a set of libraries for running cloud services. It's ideal for running temporary Hadoop clusters to carry out a proof of concept, or to run a few one-time jobs.
- **HUE:** It is a browser-based desktop interface for interacting with Hadoop.
- **ZooKeeper:** It allows Hadoop administrators to track and coordinate distributed applications.
- **HCatalog:** It is a centralized metadata management and sharing service for Apache Hadoop. It allows for a unified view of all data in Hadoop clusters and allows diverse tools, including Pig and Hive, to process any data elements without needing to know physically where in the cluster the data is stored.
- **Latency:** Any delay in a response or delivery of data from one point to another.
- **Load balancing:** The process of distributing workload across a network or cluster to optimize performance.
- **Failover:** The automatic switching to another virtual machine or node in case of a failure.
- **Network analysis:** Viewing relationships among the nodes in terms of the network or graph theory, meaning analyzing connections between nodes in a network and the strength of the ties.
- **Scalability:** The ability of a system or process to maintain acceptable performance levels as workload or scope increases.

## Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2017R1A2B4010395).

## References

Ahmad, F., Chakradhar, S. T., Raghunathan, A. & Vijaykumar, T. N. 2014. ShuffleWatcher: shuffle-aware scheduling in multi-tenant MapReduce clusters. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 1–13. https://www.usenix.org/conference/atc14/technical-sessions/presentation/ahmad.

Althebyan, Q., ALQudah, O., Jararweh, Y. & Yaseen, Q. 2014. Multi-threading based MapReduce tasks scheduling. In *2014 5th International Conference on Information and Communication Systems (ICICS)*, 16. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6841943.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Amazon! 2016a. Amazon! Elastic Block Store (EBS) – AWS Block Storage. https://aws.amazon.com/rds/ [accessed January 18, 2016].

Amazon! 2016b. Amazon! Relational Database Service (RDS). https://aws.amazon.com/rds/. [accessed January 18, 2016]

Amazon! 2016c. Amazon! Simple Storage Service (S3) – Object Storage. https://aws.amazon.com/s3/. [accessed January 18, 2016]

Amazon! 2016d. Elastic Compute Cloud (EC2). https://aws.amazon.com/ec2/. [accessed January 11, 2016]

Anjos, J. C. S., Carrera, I., Kolberg, W., Tibola, A. L., Arantes, L. B. & Geyer, C. R. 2015. MRA++: scheduling and data placement on MapReduce for heterogeneous environments. *Future Generation Computer Systems* **42**, 22–35, http://dx.doi.org/10.1016/j.future.2014.09.001.

Apache! 2015a. Apache Hadoop: Capacity Scheduler. https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html [accessed December 31, 2015].

Apache! 2015b. Apache Hadoop: Fair Scheduler. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html [accessed December 31,2015].

Apache! 2015c. ApacheTM HadoopÂ®! http://hadoop.apache.org/ [accessed December 31, 2015].

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, Q., Patterson, D., Rabkin, A., Stoica, I. & Zaharia, M. 2010. A view of cloud computing. *Communications of the ACM* **53**(4), 50–58.

Arslan, E., Shekhar, M. & Kosar, T. 2014. Locality and network-aware reduce task scheduling for data-intensive applications. In *2014 5th International Workshop on Data-Intensive Computing in the Clouds*, 1724. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7017949.

Balmin, A. & Beyer, K. S. Adaptive MapReduce using situation-aware mappers. In *EDBT '12 Proceedings of the 15th International Conference on Extending Database Technology*, 420–431.

Bezerra, A., Hernandez, P., Espinosa, A. & Moure, J. C. 2013. Job scheduling for optimizing data locality in Hadoop clusters. In *Proceedings of the 20th European MPI User's Group Meeting on – EuroMPI '13*, 271. http://dl.acm.org/citation.cfm?doid=2488551.2488591.

Bincy, P. A. & Binu, A. 2013. Survey on job schedulers in Hadoop cluster. *IOSR Journal of Computer Engineering (IOSR-JCE)* **15**(1), 4650, http://www.iosrjournals.org/iosr-jce/papers/Vol15-issue1/I01514650.pdf?id=7558.

Bortnikov, E., Frank, A., Hillel, E. & Rao, S. 2012. Predicting execution bottlenecks in map-reduce clusters. In *Proceedings of 4th USENIX Conference on Hot Topics in Cloud Computing*. http://dl.acm.org/citation.cfm?id=2342781.

Bruno, R. & Ferreira, P. 2014. SCADAMAR: scalable and data-efficient internet MapReduce. In *Proceedings of the 2nd International Workshop on CrossCloud Systems*, 2. ACM.

Chen, Q., Zhang, D., Guo, M., Deng, Q. & Guo, S. 2010. SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In *Proceedings – 10th IEEE International Conference on Computer and Information Technology, CIT-2010, 7th IEEE International Conference on Embedded Software and Systems*, ICESS-2010, ScalCom-2010, (Cit), 27362743.

Chen, Q., Liu, C. & Xiao, Z. 2014. Improving MapReduce performance using smart speculative execution strategy. *IEEE Transactions on Computers* **63**(4), 954–967.

Chen, T. Y., Wei, H. W., Wei, M. F., Chen, Y. J., Hsu, T. S. & Shih, W. K. 2013. LaSA: a locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems*, CTS 2013, 342346.

Chintapalli, S. R. 2014. *Analysis of Data Placement Strategy based on Computing Power of Nodes on Heterogeneous Hadoop Clusters*. Doctoral dissertation, Auburn University.

Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Olukotun, K. & Ng, A. Y. 2007. Map-Reduce for machine learning on multicore. *Advances in Neural Information Processing Systems* **19**, 281–288.

Dean, J. & Ghemawat, S. 2008. MapReduce. *Communications of the ACM* **51**(1), 107. http://dl.acm.org/citation.cfm?id=1327452.1327492.

Douglas, C., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S. & Saha, B. 2013. Apache Hadoop YARN – Yet Another Resource Negotiator. In *Proceedings – IEEE Fourth International Conference on eScience*, 277–284. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=736768.

Ekanayake, J., Pallickara, S. & Fox, G. 2008. MapReduce for data intensive scientific analyses. In *2008 IEEE Fourth International Conference on eScience*, 277–284. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=736768.

Facebook! 2015. Under the Hood: Scheduling MapReduce jobs more efficiently with Corona. https://www.facebook.com/notes/facebook-engineering/under-the-hoodscheduling-mapreduce-jobs-more-efficiently-withcorona/10151142560538920[accessed December 31, 2015].

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Geetha, J., UdayBhaskar, N. & ChennaReddy, P. 2016. Data-local reduce task scheduling. *Procedia Computer Science* **85**, 598–605.

Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S. & Stoica, I. 2011. Dominant resource fairness: fair allocation of multiple resource types. In *Nsdi*, 11, 24–24. http://www.usenix.org/events/nsdi11/tech/fullpapers/Ghodsi.pdf.

Gu, L., Tang, Z. & Xie, G. 2014. The implementation of MapReduce scheduling algorithm based on priority. *Parallel Computational Fluid Dynamics*, (61103047), 100–111. http://link.springer.com/chapter/10.1007/978-3-642-53962-69.

Gu, T., Zuo, C., Liao, Q., Yang, Y. & Li, T. 2013. Improving MapReduce performance by data prefetching in heterogeneous or shared environments. *International Journal of Grid and Distributed Computing* **6**(5), 71–82, http://www.sersc.org/journals/IJGDC/vol6no5/7.pdf.

Gulati, A., Shanmuganathan, G., Holler, A. M. & Ahmad, I. 2011. Cloud-scale resource management: challenges and techniques. *HotCloud 2011*, 1–6 papers2://publication/uuid/EE3F25DD-34BB-4C32-9F0C-1FA53AAB86FD.

Gunelius, S. 2015. Per day information processed. http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/ [accessed December 31, 2015].

Hammoud, M., Rehman, M. S. & Sakr, M. F. 2012. Center-of-gravity reduce task scheduling to lower MapReduce network traffic. In *Proceedings – 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 4958.

Hammoud, M. & Sakr, M. F. 2011. Locality-aware reduce task scheduling for MapReduce. In *Proceedings – 2011 3rd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2011, 570–576.

Hanif, M. & Lee, C. 2016. An efficient key partitioning scheme for heterogeneous MapReduce clusters. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*, 364–367. IEEE.

He, C., Lu, Y. & Swanson, D. 2011. Matchmaking: a new MapReduce scheduling technique. In *Proceedings – 2011 3rd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2011, 40–47.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S. & Stoica, I. 2011. Mesos: a platform for fine-grained resource sharing in the data center. *NSDI*, **11**, 22–22. http://static.usenix.org/events/nsdi11/tech/fullpapers/Hindmannew.pdfnhttps://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center.

Ibrahim, S., Jin, H., Lu, L., Wu, S., He, B. & Qi, L. 2010. LEEN: locality/fairness-aware key partitioning for MapReduce in the cloud. In *Proceedings – 2nd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2010, (2), 17–24.

Ibrahim, S., Jin, H., Lu, L., He, B., Antoniu, G. & Wu, S. 2012. Maestro: replica-aware map scheduling for MapReduce. In *Proceedings – 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid 2012, 435–442.

Jiang, W. Z. & Sheng, Z. Q. 2012. A new task scheduling algorithm in hybrid cloud environment. In *International Conference on Cloud and Service Computing*, 45–49. http://dl.acm.org/citation.cfm?id=2469449.2469626.

Jin, J., Luo, J., Song, A., Dong, F. & Xiong, R. 2011. BAR: an efficient data locality driven task scheduling algorithm for cloud computing. In *Proceedings – 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid 2011, 295–304.

Jin, S., Yang, S. & Jia, Y. 2012. Optimization of task assignment strategy for map-reduce. In *Proceedings of 2nd International Conference on Computer Science and Network Technology*, ICCSNT 2012, 57-61.

Jung, H. & Nakazato, H. 2014. Dynamic scheduling for speculative execution to improve MapReduce performance in heterogeneous environment. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 119–124.

Kc, K. & Anyanwu, K. 2010. Scheduling Hadoop jobs to meet deadlines. In *Proceedings – 2nd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2010, 388–392.

Ko, S. Y. & Cho, B. 2009. On availability of intermediate data in cloud computations. *Solutions*, 6–6, http://portal.acm.org/citation.cfm?id=1855574.

Kondikoppa, P., Chiu, C. H., Cui, C., Xue, L. & Park, S. J. 2012. Network-aware scheduling of MapReduce framework on distributed clusters over high speed networks. In *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit,* 39–44. http://doi.acm.org/10.1145/2378975.2378985.

Lee, G., Chun, B. & Katz, R. H. 2011. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of HotCloud*, **1**, 47–52. http://www.usenix.org/events/hotcloud11/tech/finalfiles/Lee.pdf.

Li, H. PWBRR Algorithm of Hadoop Platform.

Li, W., Yang, H., Luan, Z. & Qian, D. 2011. Energy prediction for mapreduce workloads. In 2011 *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 443–448. IEEE.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press

Liang, A., Xiao, L. & Li, R. 2013. An energy-aware dynamic clustering-based scheduling algorithm for parallel tasks on clusters. *International Journal of Advancements in Computing Technology*, **5**(5), 785–792, http://www.aicit.org/ijact/global/paperdetail.html?jname=IJACT&q=2412.

Liu, H. 2011. Cutting MapReduce Cost with Spot Market. USENIX HotCloud'11, 5.

Mackey, G., Sehrish, S., Bent, J., Lopez, J., Habib, S. & Wang, J. 2008. Introducing map-reduce to high end computing. In *2008 3rd Petascale Data Storage Workshop*, **3**, 1–6. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4811889.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. & Byers, A. H. 2011. Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute, (June), 156.

Matsunaga, A., Tsugawa, M. & Fortes, J. 2008. CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications. In *2008 IEEE Fourth International Conference on eScience*, 222–229. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4736761.

Morton, K., Balazinska, M. & Grossman, D. 2010. ParaTimer: a progress indicator for MapReduce DAGs. In *Proceedings of the 2010 International Conference on Management of Data*, 507–518. papers://b48995dc-e14b-47dc-9998-dcf47f651d40/P aper/p66.

Nanduri, R., Maheshwari, N., Reddyraja, A. & Varma, V. 2011. Job aware scheduling algorithm for MapReduce framework. In *Proceedings – 2011 3rd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2011, (November), 724–729.

Nita, M. C., Pop, F., Voicu, C., Dobre, C. & Xhafa, F. 2015. MOMTH: multi-objective scheduling algorithm of many tasks in Hadoop. *Cluster Computing*, **18**(3), 1–14. http://dl.acm.org/citation.cfm?id=2740070.2626334.

Palanisamy, B., Singh, A., Liu, L. & Jain, B. 2011. Purlieus: locality-aware resource allocation for MapReduce in a cloud. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 1–11.

Park, J., Lee, D., Kim, B., Huh, J. & Maeng, S. 2012. Locality-aware dynamic VM reconfiguration on MapReduce clouds. In *HPDC '12: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing SE – HPDC* '12, 27–36. http://dx.doi.org/10.1145/2287076.2287082.

Phan, L. T., Zhang, Z., Loo, B. T. & Lee, I. 2010. Real-time MapReduce scheduling. Technical Reports (CIS), (January). http://repository.upenn.edu/cisreports/942.

Polo, J., Carrera, D., Becerra, Y., Torres, J., Ayguadé, E., Steinder, M. & Whalley, I. 2010. Performance-driven task co-scheduling for MapReduce environments. In *Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium*, NOMS 2010, 373–380.

Rao, B. T., Sridevi, N. V., Reddy, V. K. & Reddy, L. S. S. 2012. Performance issues of heterogeneous Hadoop clusters in cloud computing. XI(Viii), 6. http://arxiv.org/abs/1207.0894.

Rao, B. T. & Reddy, L. S. S. 2012. Survey on improved scheduling in Hadoop MapReduce in cloud environments. *International Journal of Computer Applications* **34**(9), 29–33, http://adsabs.harvard.edu/abs/2012arXiv1207.0780T.

Ren, X. 2015. Speculation-Aware Resource Allocation for Cluster Schedulers. CITP, California, 2015.

Sandholm, T. & Lai, K. 2010. *Dynamic Proportional Share Scheduling in Hadoop. Job scheduling Strategies for Parallel Processing 2010*. Springer Berlin Heidelberg, 110–131.

Seo, S., Jang, I., Woo, K., Kim, I., Kim, J. S. & Maeng, S. 2009. HPMR: prefetching and pre-shuffling in shared MapReduce computation environment. In *2009 IEEE International Conference on Cluster Computing and Workshops*, 1–8. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5289171.

Shafer, J., Rixner, S. & Cox, A. L. 2010. The Hadoop distributed filesystem: balancing portability and performance. In *ISPASS 2010 – IEEE International Symposium on Performance Analysis of Systems and Software*, 122–133.

Shang, F., Chen, X. & Yan, C. 2017. *A Strategy for Scheduling Reduce Task Based on Intermediate Data Locality of the MapReduce*. Cluster Computing.

Su, Y. L., Chen, P. C., Chang, J. B. & Shieh, C. K. 2011. Variable-sized map and locality-aware reduce on public-resource grids. *Future Generation Computer Systems* **27**(6), 843–849, http://dx.doi.org/10.1016/j.future.2010.09.001.

Sun, R., Yang, J., Gao, Z. & He, Z. 2014. A virtual machine based task scheduling approach to improving data locality for virtualized Hadoop. In *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*, 297–302.

Sun, X., He, C. & Lu, Y. 2012. ESAMR: an enhanced self-adaptive mapreduce scheduling algorithm. In *Proceedings of the International Conference on Parallel and Distributed Systems – ICPADS*, 148–155.

Suresh, S. & Gopalan, N. 2014. An optimal task selection scheme for Hadoop scheduling. *IERI Procedia* **10**, 70–75, http://dx.doi.org/10.1016/j.ieri.2014.09.093.

Tanenbaum, A. S. 2009. Modern Operating Systems. Education, 2. http://www.amazon.com/dp/0136006639.

Tang, X., Wang, L. & Geng, Z. 2015. A reduce task scheduler for MapReduce with minimum transmission cost based on sampling. *Evaluation.* **8**(1), 1–10.

Tang, Z., Zhou, J., Li, K. and Li, R. 2012. MTSD: a task scheduling algorithm for MapReduce base on deadline constraints. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012, 2012–2018.

Teng, F., Magoulès, F., Yu, L. & Li, T. 2014. A novel real-time scheduling algorithm and performance analysis of a MapReduce-based cloud. *The Journal of Supercomputing* **69**(2), 739–765, `http://link.springer.com/10.1007/s11227-014-1115-z`.

Tian, C., Zhou, H., He, Y. & Zha, L. 2009. A dynamic MapReduce scheduler for heterogeneous workloads. In *8th International Conference on Grid and Cooperative Computing, GCC 2009*, 218–224.

Tiwari, N., Sarkar, S., Bellur, U. & Indrawan, M. 2015. Classification framework of MapReduce scheduling algorithms. *ACM Computing Surveys* **47**(3), 1–38, `http://dl.acm.org/citation.cfm?doid=2737799.2693315`.

Wei, H. W., Wu, T. Y., Lee, W. T. & Hsu, C. W. 2015. Shareability and locality aware scheduling algorithm in Hadoop for mobile cloud computing. *Journal of Information Hiding and Multimedia Signal Processing* **6**, 1215–1230.

Wolf, J., Nabi, Z., Nagarajan, V., Saccone, R., Wagle, R., Hildrum, K., Pring, E. & Sarpatwar, K. 2014. The X-flex cross-platform scheduler: who's the fairest of them all? In *Proceedings of the Middleware Industry Track*, 1. ACM.

Xia, Y., Wang, L., Zhao, Q. & Zhang, G. 2011. Research on job scheduling algorithm in Hadoop. *Journal of Computational Information Systems* **7**(16), 5769–5775.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A. & Qin, X. 2010. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*, 1–9. IEEE.

Yoo, D. & Sim, K. M. 2011. A comparative review of job scheduling for MapReduce. In *CCIS2011 – Proceedings: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, 353–358.

Yu, X. & Hong, B. 2013. Bi-Hadoop: extending Hadoop to improve support for binary-input applications. In *Proceedings – 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, CCGrid 2013, 245–252.

Zaharia, M., Borthakur, D. *et al.*. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, 265–278. `http://portal.acm.org/citation.cfm?id=1755913.1755940`.

Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H. & Stoica, I. 2008. Improving MapReduce performance in heterogeneous environments. In *Osdi*, **8**(4), 29–42. `http://www.usenix.org/event/osdi08/tech/fullpapers/zaharia/zahariahtml/`.

Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S. & Stoica, I. 2009. Job scheduling for multi-user MapReduce clusters. EECS Department University of California Berkeley Tech Rep UCBEECS200955 Apr, (UCB/EECS-2009-55), 2009-55. `http://www.eecs.berkeley.edu/P ubs/T echRpts/2009/EECS-2009-55.pdf`.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. & Stoica, I. 2010. Spark: cluster computing with working sets. In *HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 10.

Zhang, X., Feng, Y., Feng, S., Fan, J. & Ming, Z. 2011. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In *Proceedings – 2011 International Conference on Cloud and Service Computing*, CSC 2011, 235–242.

https://doi.org/10.1017/S0269888918000371 Published online by Cambridge University Press