Task Scheduling in Big Data Platforms: A Systematic Literature Review

Mbarka Soualhia, Foutse Khomh, Sofiène Tahar

PII: S0164-1212(17)30195-4
DOI: 10.1016/j.jss.2017.09.001
Reference: JSS 10033

To appear in: *The Journal of Systems & Software*

Received date: 20 October 2016
Revised date: 18 July 2017
Accepted date: 1 September 2017

Please cite this article as: Mbarka Soualhia, Foutse Khomh, Sofiène Tahar, Task Scheduling in Big Data Platforms: A Systematic Literature Review, *The Journal of Systems & Software* (2017), doi: 10.1016/j.jss.2017.09.001

Highlights

- A systematic review of scheduling models for Hadoop/Spark/Storm/Mesos (2005-2016)
- An analysis of the scheduling models proposed for Hadoop, Spark, Storm, and Mesos
- A research taxonomy for succinct classi
  cation of the proposed scheduling models
- A discussion of some future challenges pertaining to improving the current models

Task Scheduling in Big Data Platforms: A Systematic Literature Review

Mbarka Soualhia*[1], Foutse Khomh**, Sofiène Tahar*

*Concordia University, **Polytechnique Montréal, Montréal, Quebec, Canada
{soualhia,tahar}@ece.concordia.ca , foutse.khomh@polymtl.ca

Abstract

Context: Hadoop, Spark, Storm, and Mesos are very well known frameworks in both research and industrial communities that allow expressing and processing distributed computations on massive amounts of data. Multiple scheduling algorithms have been proposed to ensure that short interactive jobs, large batch jobs, and guaranteed-capacity production jobs running on these frameworks can deliver results quickly while maintaining a high throughput. However, only a few works have examined the effectiveness of these algorithms.
Objective: The Evidence-based Software Engineering (EBSE) paradigm and its core tool, *i.e.*, the Systematic Literature Review (SLR), have been introduced to the Software Engineering community in 2004 to help researchers systematically and objectively gather and aggregate research evidences about different topics. In this paper, we conduct a SLR of task scheduling algorithms that have been proposed for big data platforms.
Method: We analyse the design decisions of different scheduling models proposed in the literature for Hadoop, Spark, Storm, and Mesos over the period between 2005 and 2016. We provide a research taxonomy for succinct classification of these scheduling models. We also compare the algorithms in terms of performance, resources utilization, and failure recovery mechanisms.
Results: Our searches identifies 586 studies from journals, conferences and workshops having the highest quality in this field. This SLR reports about different types of scheduling models (dynamic, constrained, and adaptive) and the main motivations behind them (including data locality, workload balancing, resources utilization, and energy efficiency). A discussion of some open issues and future challenges pertaining to improving the current studies is provided.

Keywords: Task Scheduling, Hadoop, Spark, Storm, Mesos, Systematic Literature Review.

1. Introduction

The processing and analysis of datasets in cloud environments has become an important and challenging problem, because of the exponential growth of data generated by social networks, research and healthcare platforms, just to name a few. Hadoop [1], Spark [2], Storm [3], and Mesos [4] are examples of widely used frameworks for distributed storage and distributed processing of ultra large data-sets in the cloud. Many large organisations like Yahoo!, Google, IBM, Facebook, or Amazon have deployed these well-known big data frameworks [5]. Hadoop, Spark, Storm, and Mesos are multi-tasking frameworks that support a variety of different types of tasks processing. They have a pluggable architecture that permits the use of schedulers optimized for particular workloads and applications. The scheduling of tasks in these frameworks is of a paramount importance since it affects the computation time and resources utilisation. However, because of the dynamic nature of cloud environments, efficient task scheduling is very challenging. Multiple algorithms have been proposed to improve how tasks are submitted, packaged, scheduled and recovered (in case of failures) in these frameworks. Yet, only a few works have compared the proposed algorithms and investigated their impact on the performance of the aforementioned frameworks. To the best of our knowledge, there is no published literature that clearly articulates the problem of scheduling in big data frameworks and provides a research taxonomy for succinct classification of the existing scheduling techniques in Hadoop, Spark, Storm, and Mesos frameworks. Previous efforts [6], [7] [8] that attempted to provide a comprehensive review of scheduling issues in big data platforms were limited to Hadoop only. Moreover, they did not include all papers that were published during the periods covered by their studies (*i.e.*, 2012 and 2015). Also, these three studies only propose general descriptions of Hadoop schedulers in terms of architecture and objectives (*e.g.*, learning, resources management) and do not discuss their limitations. Neither do they discuss future research directions to improve these existing task scheduling approaches.

[1]Corresponding author: Mbarka Soualhia (soualhia@ece.concordia.ca)

In this paper, we follow the Evidence-based Software Engineering (EBSE) paradigm in order to conduct a Systematic Literature Review (SLR) [9] of task scheduling techniques in Hadoop, Spark, Storm, and Mesos, with the aim to identify and classify the open challenges associated with task scheduling in these frameworks.
We discuss different approaches and models of task scheduling proposed for these four frameworks, that gained a lot of momentum in the last decade in both research and commercial communities. Also, we analyse the proposed design decision of each approach in terms of performance, resources utilization, failure recovery mechanisms, and energy efficiency. Our searches identified 586 journals, conferences and workshops papers published in top ranked software engineering venues between 2005 and 2016. We organize our SLR in three parts:

- ***Part 1:*** *Task Scheduling Issues in Big Data Platforms:*

First, we present the main issues related to task scheduling in Hadoop, Spark, Storm, and Mesos, and explain how these issues are addressed by researchers in the existing literature. We classify the issues into 6 main categories as follows: *resources management*, *data management* (including *data locality*, *replication* and *placement* issues), *fairness*, *workload balancing*, *fault-tolerance*, and *energy-efficiency*.

- ***Part 2:*** *Task Scheduling Solutions in Big Data Platforms:*

Second, we describe the different types of scheduling approaches available in the open literature and discuss their impact on the performance of the schedulers of the four frameworks. Overall, we observe that we can classify the scheduling approaches used in Hadoop, Spark, Storm, and Mesos into three main categories: *dynamic*, *constrained* and *adaptive* scheduling.

- ***Part 3:*** *Research Directions on Task Scheduling in Big Data Platforms:*

Third, we describe some of the future research directions that can be addressed in each category discussed previously in *part 1* and *part 2* of the SLR. From the limitations of previous work (discussed in *part 2*), we build a roadmap for future research to improve existing scheduling approaches.

The remainder of this paper is organized as follows: Section 2 briefly introduces Hadoop, Spark, Storm, and Mesos. Section 3 describes the methodology followed in this Systematic Literature Review. Sections 4, 5 and 6 discuss our study and the findings of this review, and position our work in the existing literature. Section 7 presents our conclusions, and outlines the main findings of this systematic review.

2. Background

Figure 1 describes the relationships between MapReduce, Hadoop, Spark, Storm, and Mesos. Hadoop is a well-known processing platform that implements the MapReduce programming model. Spark is a novel in-memory computing framework that can be running on Hadoop. Storm is a distributed computation framework for real time applications. Spark and Storm can implement the MapReduce programming model, but with different features to handle their topologies and data models. These platforms can be typically deployed in a cluster, that can be managed by Mesos or YARN (Yet Another Resources Negotiator), which are cluster managers. In the sequel, we briefly describe MapReduce, Hadoop, Spark, Storm, and Mesos.

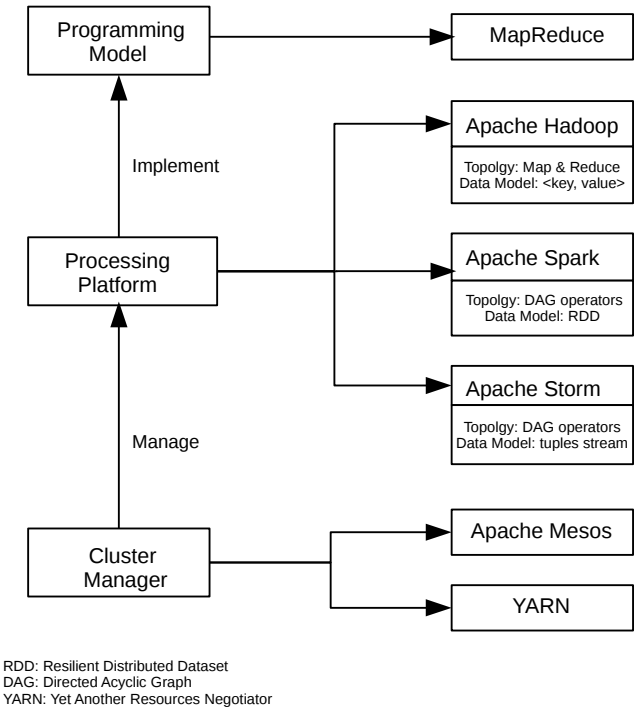

Figure 1: An overview of relationships among MapReduce, Hadoop, Spark, Storm, and Mesos

2.1. Programming Model: MapReduce

MapReduce [10] is a programming model for processing big amounts of data using a large number of computers (nodes). It subdivides the received users' requests into parallel jobs and executes them on processing nodes where data are located, instead of sending data to the nodes that execute the jobs. A MapReduce job is composed of *"map"* and *"reduce"* functions and the input data. The input data represents a set of distributed files that contain the data to be processed. The *map* and *reduce* functions are commonly used in functional programming languages like Lisp. The map function takes the input data and outputs a set of <key, value> pairs. The reduce function takes the set of values for a given key as input and emits the

output data for this key. A shuffling step is performed to transfer the map outputs to the corresponding reducers. The set of intermediate keys are sorted by Hadoop and given to the reducers. For each intermediate key, Hadoop passes the key and its corresponding sorted intermediate values to the reduce function. The reducers (*i.e.*, worker running a reduce function) use a hash function to collect the intermediate data obtained from the mappers (*i.e.*, worker running a map function) for the same key. Each reducer can execute a set of intermediate results belonging to the mappers at a time. The final output of the reduce function will be stored in a file in the distributed file system [11]. MapReduce follows a master-slave model. The master is known as *"JobTracker"*, which controls the execution of the "map" and "reduce" functions across the slave workers using *"TaskTrackers"*. The JobTracker and the TaskTrackers control the job execution to ensure that all functions are executed and have their input data as shown in Figure 2.

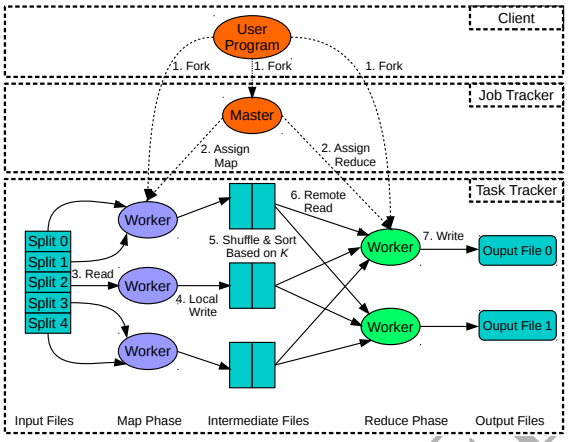

Figure 2: An overview of Job Execution in MapReduce [11]

2.2. Processing Platforms

In the sequel, we describe Hadoop, Spark, and Storm processing platforms and we briefly discuss task scheduling issues in these platforms.

2.2.1. Apache Hadoop

Hadoop [12] has become the *de facto* standard for processing large data in today's cloud environments. It is a Java-based MapReduce implementation for large clusters that was proposed by Cutting and Cafarella in 2005 [12]. Hadoop is composed of two main components: the Hadoop Distributed File System (HDFS) and the MapReduce framework. The HDFS is responsible for storing and managing the input of the map function as well as the output of the reduce function. The Hadoop MapReduce framework follows a master-slave model [11]. The JobTracker running on the master is responsible for managing the job execution, progress and the status of the workers (slaves). Each worker in Hadoop is composed of a TaskTracker and a DataNode. The TaskTracker is responsible for processing the jobs using their corresponding input data located in the DataNode [11]. Hadoop allows the processing of large data-sets across a distributed cluster using a simple programming model. It is designed to hide all details related to the job processing (such as error handling or distribution of tasks across the workers). This allows developers to focus only on enhancing computation issues (in terms of response time, resources utilisation, energy consumption etc.) in their parallel programs rather than parallelism.

2.2.2. Apache Spark

Spark [2] is a novel in-memory computing framework written in Scala for Hadoop, proposed in 2010. It was developed to address the problem in the MapReduce model, which accepts only a particular linear data flow format for distributed programs. Spark uses a data structure called Resilient Distributed Dataset (RDD), which is a distributed memory abstraction that allows for in-memory computations on large clusters in a fault-tolerant way [13]. In MapReduce programs, the input data are read from the disk then mapped through a map function, and reduced using a reduce function to get the output data that will be stored on the disk. Whereas in Spark programs, the RDDs serve as a working set for distributed programs, which offer a restricted form of distributed shared memory [2]. The RDDs support more functions, compared to MapReduce, that can be classified into two main categories; the "transformation" and the "action" functions. The tranformation function can be a map, filter, sample, union, or an intersetcion operation. While an action function can be a reduce, collect, countbykey, take, or takeordered operation. Consequently, the RDDs allow to reduce the latency for both iterative and interactive data analysis applications by several orders of magnitude when compared to Hadoop [13]. Spark is comprised of two main components: a cluster manager and a distributed storage system. Spark supports the native Spark cluster, Hadoop YARN [14], or Mesos [4] as a cluster manager. Also, it supports communication with a multitude of distributed storage systems including HDFS, MapR File System (MapR-FS), and Cassandra [15].

2.2.3. Apache Storm

MapReduce and Hadoop are designed for offline batch processing of static data in cloud environments, which makes them not suitable for processing stream data applications in the cloud (*e.g.*, Twitter) [3]. To alleviate this issue, Storm [3] has emerged in 2011 as a promising computation platform for stream data processing. Storm is a distributed computation framework written in Clojure and Java, and designed for performing computations of streams of data in real time. In order to be processed in Storm, an application should be modelled as a directed graph called a topology that includes spouts and bolts, and the data strems of the applications can be routed and grouped through this graph. Particularly, there are different grouping startegy to control the routing of data streams through the directed graph including the field

grouping, global grouping, all grouping, and shuffle grouping [3]. The spouts are sources of data stream (sequence of tuples), they read data from different sources including database, messaging frameworks, and distributed file systems. The bolts are used to process data messages and to acknowledge the processing of data messages when it is completed. Also, they can be used to generate other data messages for the subsequent bolts to process. Generally, one can utilize the bolts for filtering, managing, aggregating the data messages, or to interact with external systems. Storm can achieve a good reliability by using efficient procedures to control message processing. Also, it has fault-tolerant mechanisms that allow to restart failed workers in case of failures [16].

2.2.4. Task Scheduling

In general, task scheduling is of paramount importance since it aims at allocating a number of dependent and/or independent tasks to the machines having enough resources in the clusters. An effective scheduler can find the optimal task distribution across the machines in a cluster, in accordance with execution time requirements and resources availability. An optimal task distribution minimises the mean execution time of the scheduled tasks and maximises the utilisation of the allocated resources. This is in order to maximise the response time of the received computations (tasks to be processed), and reduce (avoid) resources waste. Each big-data platform in the cloud is equipped with a scheduler that manages the assignment of tasks. Here, we briefly present as examples the well-known schedulers proposed for Hadoop that gained a lot of attention from both industry and academia. In Hadoop, the JobTracker is responsible for scheduling and provisioning the submitted jobs and tasks. It has a scheduling algorithm, which initial implementation was based on the First In First Out (FIFO) principle. The scheduling functions were first regrouped in one daemon. Hadoop developers decided later to subdivide them into one Resource Manager and (per-application) Application Master to ease the addition of new pluggable schedulers. YARN (Yet Another Resources Negotiator) [14] is the daemon responsible for managing applications' resources. Facebook and Yahoo! have developed two new schedulers for Hadoop: Fair scheduler [17] and Capacity scheduler [18], respectively.

2.3. Cluster Manager: Apache Mesos

Mesos [4] is an open-source cluster manager that provides efficient resource usage and sharing across multiple cluster computing frameworks. It was proposed in 2009 by the University of California, Berkeley. Instead of a centralized approach, Mesos supports a two-level scheduling approach to allocate the resources to the frameworks (in this context, a framework is a software system that executes one or more jobs in a cluster). Hence, Mesos enables efficient resources sharing in a fine-grained way. So, the master node in Mesos decides the amount of resources to be assigned for each framework. Then, each framework accepts the resources it needs and decides which jobs to execute on those resources. This approach can help optimize the allocation of resources as well as provide near-optimal data locality [4].

3. Methodology of the S.L.R.

The following subsections present our proposed methodology to perform the Systematic Literature Review (SLR), and the outcomes of the SLR:

3.1. Conducting the Study

3.1.1. Data Sources

Following the guidelines given in [9], we start our SLR using the following relevant search engines: *IEEE Xplore*, *ACM*, *Google Scholar*, *CiteSeer*, *Engineering Village*, *Web of Science* and *ScienceDirect*. We perform an electronically-based search and consider the main terms related to this review: *"scheduling"*, *"task scheduling"*, *"scheduler"*, *"MapReduce"*, *"Hadoop"*, *"Spark"*, *"Storm"*, and *"Mesos"*. We use the same search strings for all seven search engines. We look for published scientific literature related to task scheduling in Hadoop, Spark, Storm and Mesos between 2005 and 2016. Then, we restrict our study to a number of journals, conferences, workshops and technical reports having the highest quality and considered as the most important resources in this field. We perform this step by selecting the studies published in journals with high impact factors, conferences/workshops with competitive acceptance rates and technical reports with high number of citations. Also, we check the citation of the studies in order to evaluate their impact in this field. Other studies are rejected for quality reasons (*e.g.*, the study is only a small increment over a previous study, a technical report that is extended into a journal or a conference/workshop paper, etc). Table 1 presents a non-exhaustive list of workshops, conferences and journals considered in our SLR.

Table 1: A Non-Exhaustive List of Journals, Conferences, and Workshops Considered in our SLR

Journals
- Journal of Systems and Software (JSS) - Future Generation Computer Systems (FGCS) - Transactions on Parallel and Distributed Systems (TPDS) - Transactions on Service Computing (TSC)
Conferences
- IEEE INFOCOM - IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) - IEEE International Conference on Distributed Computing Systems (ICDCS)
Workshops
- Workshop on Data Engineering - Workshop on Parallel Distributed Processing and Ph.D Forum - Symposium on High-Performance Parallel and Distributed Computing

3.1.2. Search and Selection Process

The search and selection process for the relevant studies from data sources is organized in three rounds as described in Figure 3.

- **Round 1:** we perform a mapping study named also a scoping review in order to identify and categorise the primary studies related to the SLR based on their scope. This scoping review helps identify the main issues addressed and studied in the available literature. Next, we select the most relevant studies based on their titles and abstracts. Any irrelevant study is removed. If there is any doubt about any study at this level, the study is kept.

- **Round 2:** it consists of a manual search of the studies obtained in the previous step (*i.e.*, Round 1), which are identified as the main sources for the SLR. It is necessary to check the reliability of the selected studies. To do so, the remaining studies at this step are carefully read. Then, the irrelevant studies are removed based on the selection criteria defined in the work of Dyba and Dingsoyr [19]. More details about the used criteria are given in Section 3.2.

- **Round 3:** we perform a snowball search based on guidelines from [20]. We apply a backward snowball search using the reference list of papers obtained in the second round, to identify new papers and studies. We use the same selection criteria (as in *Round 2*) to decide whether to include or exclude a paper. These remaining papers are read carefully.

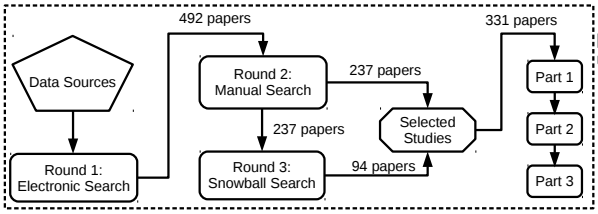

Figure 3: Overview of SLR Methodology

3.2. Quality of the Selected Papers

We apply different inclusion and exclusion criteria on the remaining studies in the second and third rounds. These selection criteria can help decide whether to include or not a paper for further search. Only relevant studies that are retained will be used in the SLR analysis to answer our research questions. (1) Only papers describing issues related to Hadoop, Spark, Storm, and Mesos schedulers and proposing models to improve their performance are included. (2) Documents presented in the form of *power point* presentations, abstract, and submitted papers are not included in this review.

3.3. Outcomes of the Study

The different search stages of our SLR identify a total of **586** papers. Specifically, we obtain 492 papers from **Round 1**, from which we extract 237 papers after **Round 2**. Next, we discover 94 new papers during the snowball phase (*i.e.*, **Round 3**). In total, the number of papers analyzed in this SLR is 492 + 94 = 586 (from Round 1 and Round 3). This is after removing those papers that are not related to task scheduling in Hadoop, Spark, Storm, or Mesos, and duplicates that are found by more than one search engine. When a paper is found by two search engines in Round 1, we keep the one published in the search engine having the highest number of papers. For example, if a paper is found by IEEE and ACM and IEEE has the highest number of obtained study, we keep the one in IEEE and remove the duplicate in ACM. The results obtained on each of the seven search engines, for the three rounds are presented in Table 2.

Table 2: Results of SLR Rounds

Search Engine	Round 1	Round 2	Round 3
- IEEE	252	143	50
- ACM	95	61	28
- CiteSeer	41	8	2
- Google Scholar	37	6	9
- ScienceDirect	39	12	5
- Web of Science	5	1	0
- Engineering Village	23	6	0
Total	492	237	94

3.4. SLR Organization

The following paragraphs describe the motivation for each part in the SLR:

Part 1: *Task Scheduling Issues in Big Data Platforms:*
This part provides a comprehensive overview of task scheduling in Hadoop, Spark, Storm, and Mesos. It aims at identifying the main topics of task scheduling addressed in these frameworks. Hence, it can help determine the challenges and issues that have been studied by both research and commercial communities. The identified challenges and issues will help draw the map of the state of research on task scheduling in these computing frameworks.

Part 2: *Task Scheduling Solutions in Big Data Platforms:*
This part describes the proposed solutions in the existing literature that addressed the scheduling issues identified in ***Part 1***. This exhaustive analysis can help give a comprehensive overview about the characteristics of the proposed solutions, their main objectives and their limitations. In fact, it presents the advantages and limitations of each solution that aimed to improve Hadoop, Spark, Storm, and Mesos schedulers over time. Furthermore, it can help identify some future work that can be addressed by researchers in order to better improve the schedulers of these frameworks.

Part 3: *Research Directions on Task Scheduling in Big Data Platforms:*
This part identifies some of the future work that can be done to cover the drawbacks of the solutions reported in ***Part 2***. Based on the limitations of these proposed solutions, we aim to identify some aspects that can be

enhanced to better improve Hadoop, Spark, Storm, and Mesos schedulers. ***Part 3*** draws a roadmap for further studies on task scheduling in these frameworks.

3.5. SLR Analysis Approach

We perform a manual search over the papers found at the different search stages. To do so, we proceed in two steps. First, we skim through the papers, reading the most relevant parts to get an overview of the issues addressed in the existing literature to construct ***Part 1***. Next, we classify the obtained studies in different categories based on their scope to ease the analysis of the selected papers. Also, we compute statistics about the number of published studies and papers *(i)* in each category; and *(ii)* the evolution of this metric over time (from 2005 up to 2016) to get an overview of the most studied scheduling issues in Hadoop, Spark, Storm, and Mesos. Second, we carefully analyse all papers in order to extract the relevant information about the proposed approaches and their limitations to build ***Part 2***. Indeed, we classify the proposed approaches in different categories following the list of issues identified in ***Part 1***. If there is a study that is addressing two issues at the same time, it will be included in both categories. Finally, to develop ***Part 3***, we identify some future work that can be addressed to cover the limitations of the approaches discussed in ***Part 2***. The following sections present and discuss the results of the three parts in our SLR.

4. Task Scheduling Issues In Big Data Infrastructures

Before addressing the first part of the SLR, we examine the candidate papers based on their publication years to identify the distribution of the related studies over time. Figure 4 shows the interest of researchers on task scheduling for big data frameworks over time. We notice that during the first three years, after proposing Hadoop in 2005, there was no study that analysed scheduling issues in Hadoop. This is arguably due to the fact that researchers were more interested in the computing functions of Hadoop and were striving to improve them. Next, we observe that in 2008, the topic of scheduling in Hadoop started gaining attraction, with 2 papers published on the topic in 2008. A limited number of studies were performed on the topic between 2009 (11 papers) and 2010 (19 papers). Then, the number of studies significantly increased from 11 papers in 2009 to 50 papers in 2016. This can be explained by the constant increase of the popularity of Hadoop (Hadoop is now widely used by different companies and research labs). Also, the high number of Hadoop users was affecting the overall performance of the scheduler and hence many studies were needed to resolve the issues faced while deploying Hadoop. With the emergence of Mesos (2009), Spark (2010) and Storm (2011), many other works were done to study the performance of these platforms in Cloud environments. We can claim that during three years starting from 2014 until 2016, a minimum of 45 studies were published on Hadoop/Spark/Mesos/Storm scheduling issues each year; highlighting the importance of this research topic. Also, we find that the majority of the studies were addressing scheduling issues in Hadoop and only a few works were analysing the other three platforms. This can be explained by the popularity of Hadoop in both academia and industry and also because Hadoop was proposed before the other platforms (in 2005).

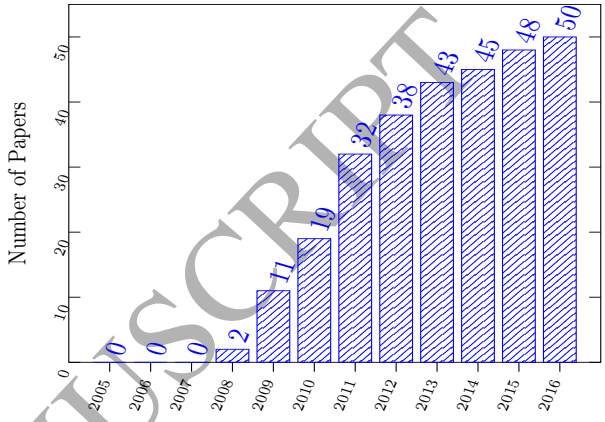

Figure 4: Distribution of Papers over Years [2005-2016]

Next, we identify the addressed task scheduling issues in big data platforms using the papers obtained from the three rounds. We find that we can classify these papers into six categories as shown in Figure 5. For each category, we briefly describe the addressed issues in the schedulers of the big data platforms. Next, we select a relevant work from the open literature, for each category, as an example that can formally describe these issues and better illustrate the addressed problems. The obtained categories can be described as follows:

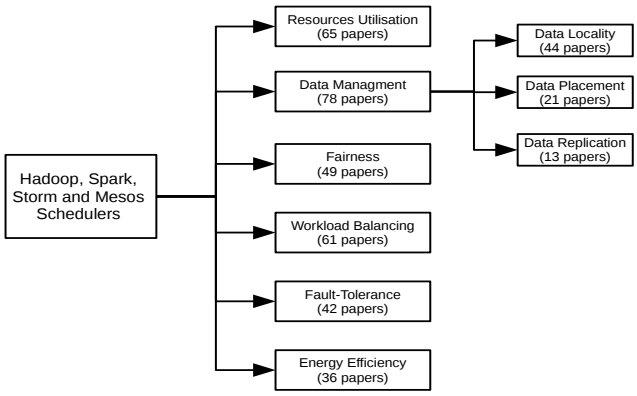

Figure 5: Scoping Review Results

4.1. Resources Utilisation (65 papers)

4.1.1. General Definition

In general, the resources allocation process aims to distribute the available resources across the scheduled

tasks, in order to ensure a good quality of services for users and to reduce the cost of the services for cloud providers. Particularly, the computing resources (*e.g.*, CPU cores, memory) are distributed using the two basic computing units in a MapReduce job; map and reduce slots. A slot is the basic unit used to abstract the computing resources (e.g., CPU cores, memory) in Hadoop, Spark, and Storm. It is used to indicate the capacity of a worker (e.g., TaskTracker). There exist two types of slots in a cluster: a slot that can be assigned to a map task, and a slot to be assigned to a reduce task. These computing units are statically configured by the administrator before launching a cluster. The map slots must be only allocated to the map tasks and the reduce slots must be only given to the reduce tasks. The allocation of these slots should be executed under the constraint that the map tasks should be executed and finished before their corresponding reduce task starts. Specifically, the map and reduce phases in a job have a tight dependency on each other. Indeed, the map tasks can be running in parallel since they are small and independent. The reduce tasks will be launched in parallel with the copy and merge phases and will not release their assigned slots until all reduce tasks are completed. Hence, they take much longer time to be finished. So, this fundamental interdependence can lead to a repeatedly observed starvation problem. Furthermore, the map and reduce tasks may have highly varying resources requests over time, which makes it difficult for the scheduler to efficiently utilize the cluster resources. In addition, given the dynamic nature of cloud applications, the resources allocation process can be complex and may fail to allocate the required resources for some jobs (*i.e.*, long running-time jobs) or fail to prevent tasks from the starvation problem. As a result, there can be some straggling map or reduce tasks because of the unexpected contention time for CPU, memory and other resources; resulting in unpredictable execution time for the tasks. Overall, we found 65 studies addressing this issue.

4.1.2. Problem Formulation Example

Cheng *et al.*[21] consider the problem of assigning the resources in Hadoop scheduler as an optimization problem where they aim to minimize the penalty of deadline misses. Precisely, they consider that there are J jobs running in a Hadoop cluster with dynamic resource availability r_a and a time control interval t (t $\in$ [1,..,T]). The scheduler assigns u_j^m to the map tasks and u_j^r to the reduce tasks belonging to a job j. The completion time of a job j is y_j and its reference time that meets its deadline is u_j^{ref}. Each job j has a constant w_j that represents its priority. The problem of resources assignment is formally specified as follows:

$$\min \sum_{t=1}^{T} \sum_{j=1}^{J} \omega_j \left(\frac{y_j - y_j^{ref}}{y_j^{ref}}\right) \tag{1}$$

$$s.t. \sum_{j=1}^{J} (u_j^m + u_j^r) \leq r_a \tag{2}$$

The equation 1 determines the lost revenue because of deadline misses. The penalty remains zero unless the job j misses its deadline. Equation 2 represents the constraints to verify that the sum of the map and reduce resources assigned to a job j are bounded by the amount of the available resources in the cluster.

4.2. Data Management (78 papers)

We can claim that the problem of data management in big data platforms can be sub-divided into three main sub-problems as follows:

4.2.1. Data Locality (44 papers):

4.2.2. General Definition

In big data platforms, the computations (*i.e.*, received workloads) are sent as close as possible to where the input data of the scheduled tasks is located. This is because of the large size of the processed data rather than moving these large data blocks to the computational nodes where the tasks will be running. So, the scheduler decides where to send the computations based on where the data exists. *Data locality* is an important issue addressed by many researchers as shown in Figure 5. In particular, we find 44 studies that addressed this problem in Hadoop, Spark, Storm and Mesos schedulers. Scheduling tasks based on the locality of their associated data is a crucial problem that can affect the overall performance of a cluster. Indeed, the execution of some jobs or tasks requires the processing of tasks having distributed data across different nodes. Therefore, it is necessary to find a better allocation of these tasks over the available nodes while maximizing the number of tasks executing local data. This is to reduce the total execution time of tasks by reducing the number of non-local-data tasks since these tasks spend more time to read and write data compared to the local-data tasks.

4.2.3. Problem Formulation Example

According to Xun *et al.* [22], the data locality issue can be solved by reducing the data transfer cost for the map tasks. They consider the input data for a job as a set of transactions D = $\{t_1, .., t_n\}$, the corresponding map tasks are M = $\{m_1, .., m_p\}$, and I the set of intermediate key-value pairs generated by the mappers I = $\{(G_1, D_1), .., (G_p, D_p)\}$ where D_i represents the set of transactions associated to a group G_i. S(G_i) and T(G_i) represent respectively the source and target node. Xun *et al.* [22] propose to measure p_i as a metric to indicate whether a pair is produced on a local node or not as follows:

$$p_i = \begin{cases} 1 & \text{if } S(G_i) \neq T(G_i) \\ 0 & \text{if } Otherwise \end{cases} \tag{3}$$

Xun *et al.* [22] propose to reduce the cost of data transfer as follows:

$$Minimize : \sum_{i=1}^{m} D_i * p_i \quad (4)$$

4.2.4. Data Placement (21 papers):

4.2.5. General Definition

Although, the proposed data locality strategies can help improve the processing of tasks in the nodes having the local input data and enough resources, an extra overhead can be added when processing the non-local data blocks and moving the intermediate data from one node to another to get the final output; which may decrease the overall performance of a cluster. Particularly, the processing of scheduled tasks highly depends on the location of the stored data and their placement strategy. This makes it difficult *(a)* for the platform (*e.g.*, Hadoop, Spark) to distribute the stored data; and *(b)* for the scheduler to assign the scheduled tasks across the available nodes in a cluster. Therefore, there are some studies that are proposed to improve the *data placement* issue within the distributed nodes in these big data platforms in order to improve the strategies responsible for moving the data blocks especially the large data-sets. We find 21 studies that addressed this problem.

4.2.6. Problem Formulation Example

According to Guo *et al.* [23], the problem of partition placement can be formulated as a minimization problem to find the nodes where to place the data. Given m map partitions characterized by the sizes $p_1, p_2, .., p_m$, the goal of the proposed model in [23] is to find the placement of n nodes $S_1, S_2, .., S_n$ while minimizing the placement difference σ defined as follows:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mu - \sum_{j \in S_i} p_j)} \quad (5)$$

Where μ represents the average data size on one node.

4.2.7. Data Replication (13 papers):

4.2.8. General Definition

Data locality and placement are very important problems as they significantly affect the system performance and the availability of the data. However, some slow or straggling nodes can go down and their data blocks will not be available for sometime, which may affect the overall performance. Therefore, several algorithms are proposed to replicate the data across different processing nodes. For example, if some data cannot be found, the scheduler may find other replicas in other nodes, racks or other clusters and run speculative copies of the straggling tasks. By increasing the number of data replicas, the scheduler may be able to increase the successful processing of tasks that are not able to find their input data. However, the distribution and number of these replica can vary over time due to the dynamic nature of the applications running on a cluster. Therefore, it is necessary to propose better schemes to replicate data over big data platforms nodes. This principle is known as *data replication* and is studied by 13 works as presented in Figure 5.

4.2.9. Problem Formulation Example

Convolbo *et al.* [24] consider a system composed of N data centers S = $\{DC_1, .., DC_N\}$. Each task t_i^j that represents the i^{th} task belonging to a job j is associated with a required data d_i^j. They propose to use a function $L_{j,k}^i$ to check whether the data of the task t_j^i is replicated or not in the system as follows:

$$L_{j,k}^i = \begin{cases} 1 & \text{if } data \text{ } is \text{ } replicated \text{ } in \text{ } the \text{ } system \\ 0 & \text{if } otherwise \end{cases} \quad (6)$$

4.3. Fairness (49 papers)

4.3.1. General Definition

Ensuring fairness is particularly important to improve the overall performance of a cluster for big data infrastructures, especially when there are different jobs running concurrently in the cluster. Particularly, 49 papers addressed this issue in the existing literature; which explains the importance of this problem. A job is composed of multiple map and reduce tasks that occupy the available resource slots in the cluster. The slot configuration can differ from one node to another. Also, jobs can have different resource requirements and consumptions over time. For example, some jobs can occupy their assigned resources for long time, more than expected, which may cause the starvation of some other jobs waiting for their turn in the queue. Also, some jobs may take the advantage of occupying the resources to finish their tasks much faster while other jobs can be waiting a long time for their turn to be executed using the same resources. Overall, the scheduler can experience many problems such as jobs starvation and long execution time if it does not define a fair plan to assign the available slots across the jobs and tasks. Consequently, the scheduler should fairly assign the received tasks to the node slots in order to reduce the makespan time between the scheduled tasks. Specifically, it should find the optimal task assignment across the available slots while reducing the overhead generated between the map and reduce tasks to communicate the intermediate results.

4.3.2. Problem Formulation Example

Niu *et al.* [25] propose to formally measure the fairness loss F of a scheduler. They consider J (J $\in$ {1,..,k}) is the

set of users sharing the resources in a cluster. The completion times of the application of a user i under a specific scheduler and a fair scheduler are t_i^* and t_i respectively. Niu *et al.* [25] calculate first the reduction of the completion time of the applications of a user i; s_i as follows:

$$s_i = \max\left\{\frac{t_i^* - t_i}{t_i}, 0\right\} \quad (7)$$

Next, they determine the fairness loss F of a scheduler when processing the received workload as follows:

$$F = \frac{\sum_{i \in J} s_i}{k} \quad (8)$$

4.4. Workload Balancing (61 papers)

4.4.1. General Definition

The schedulers of cloud platforms, like Hadoop, Spark, Mesos, or Storm, receive multiple jobs and tasks having different characteristics and different resource demands, which may lead to different system workloads. Furthermore, workloads can be imbalanced when jobs are running, under a current scheduler, across the cluster nodes, which may cause important delays and many problems related to the reliability of the system. Therefore, it is very important to balance the workload across the distributed nodes to improve its performance in terms of execution time and resources utilisation; especially for cloud applications that should satisfy some defined response requirements. The workload balancing is very dependent on the fairness allocation of the available slots across the scheduled tasks. In fact, one reason behind unbalanced workload is a bad slot allocation in a worker, which can generate straggling tasks waiting for slots to be released. So, a good slot allocation strategy allows to have a balanced workload. Other factors like resources allocation, type of scheduler, can also affect the workload behavior in each node. Many studies in the available literature (*i.e.*, 61 papers) proposed algorithms to balance the received load in a cluster.

4.4.2. Problem Formulation Example

Guo *et al.* claim in [26] that the load balancing is correlated to the skew in task execution; the lower is the skewness, the more balanced is the task execution. Therefore, they present a model to calculate the skew in a task runtime in a job J as follows:

$$\frac{1}{n}\sum_{i \in J}\frac{(x_i - \mu)^3}{\sigma^3} \quad (9)$$

Here, x_i represents the runtime of a task, μ and σ are the average and the deviation, respectively.

4.5. Fault-Tolerance (42 papers)

4.5.1. General Definition

Heterogeneity is the norm in cloud environments, where different software configurations can be found. Particularly, Sahoo *et al.* [27] claim that the complexity of software systems and applications running on a cluster can cause several software failures (*e.g.*, memory leaks, state corruption), make them prone to bugs and may lead to crashes in the cluster. Physical machines in cloud clusters are subject to failure (*e.g.*, they can be down for some time), which may lead to unexpected delays to process the received jobs. Moreover, big-data platforms' schedulers can experience several task failures because of unpredicted demands of service, hardware outages, loss of input data block, nodes failure, etc. Although, Hadoop, Spark, Mesos and Storm have built fault-tolerance mechanisms to cover the aforementioned limitations, one task failure can cause important delays due to the tight dependency between the map and reduce tasks. Also, it can lead to not-efficient resources utilisation because of an unexpected resources contention. Therefore, many researchers addressed this issue in these platforms. In particular, 42 studies are found in the literature addressing this crucial problem. These studies propose different *Fault-Tolerance* mechanisms to reduce the number of failures in the processing nodes, and hence improve the overall performance of the cluster. In addition, they describe different mechanisms to improve the availability and reliability of platforms components, in order to better improve the availability of the services offered to the users.

4.5.2. Problem Formulation Example

In [28], the authors formally define the scheduling outcome of an executed job composed of X map tasks and Y reduce tasks. $S(MapAtt_{ip})$ is the status of a map_i after the p^{th} attempt, and $S(ReduceAtt_{jq})$ the status of $reduce_j$ after the q^{th} attempt. $S(MapAtt_{ip})$ and $S(ReduceAtt_{jq})$ have 1 as a value when the attempt is successful and 0 when it is failed. K and L represent the maximum numbers of scheduling attempts allowed for map and reduce tasks respectively. The authors [28] model the scheduling outcome of a scheduled job as follows:

$$S(job) = [\prod_{i=1}^{X}(\sum_{p=1}^{K} S(MapAtt_{ip}))] * [\prod_{j=1}^{Y}(\sum_{q=1}^{L} S(ReduceAtt_{jq}))] \quad (10)$$

4.6. Energy Efficiency (36 papers)

4.6.1. General Definition

The cost of data-intensive applications running on large clusters represent a critical concern in terms of energy efficiency. For instance, data-intensive applications require more energy when processing the received workload, executing I/O disk operations, and managing and processing the huge amount of data on big data platforms

like Hadoop, Spark, or Mesos. Moreover, the design of a scheduler for big data platforms can largely affect the energy consumption of the system on which the applications are executed. For instance, when processing tasks on the nodes where the data exists, the node may receive a large number of tasks and these nodes require more resources to execute them, which can increase the level of energy consumed. Moreover, the nodes in a cloud cluster can experience several failures and can face straggling tasks resulting in more energy being consumed. Therefore, minimizing the energy consumption when processing the received workload is of paramount importance. We find 36 studies that addressed this critical issue in cloud environments. These studies show that there is a trade-off between improving the scheduler performance and the energy consumption on the studied platforms in our SLR.

4.6.2. Problem Formulation Example

Cheng *et al.* [29] model the problem of energy consumption in Hadoop as a minimization problem. They consider a Hadoop cluster composed of M workers, and there are J jobs to be processed composed of N tasks $\{T_1^j,..,u_n^j\}$. They formalize the problem of tasks assignment in order to minimize the overall energy consumption as follows:

$$\min \sum_{j=1}^{J} \sum_{n=1}^{N} \sum_{m=1}^{M} [E(T_n^j(m))] \quad (11)$$

$$s.t. \sum_{j=1}^{J} \sum_{n=1}^{N} |T_n^j(m)| \leq m_{slot} \quad (12)$$

Here, $E(T_n^j(m)$ is the amount of energy consumed by the n^{th} task from the j^{th} job that is running on the m^{th} machine. The formal definition of $E(T_n^j(m)$ can be found in [29]. The constraint defined in equation 12 ensures that the number of concurrent task executions of the scheduled jobs are bounded by the number of available slots on the machine m where they are processed.

Overall, the main issues related to task scheduling in Hadoop, Spark, Storm and Mesos can be classified into six main categories: resources management, data management (including data locality, replication and placement issues), fairness, workload balancing, fault-tolerance, and energy efficiency. The solutions and approaches proposed to solve the aforementioned issues in each category will be described in the next section.

5. Task Scheduling Solutions In Big Data Infrastructures

After carefully checking the content of the obtained papers, we can group them by scope/objective in order to analyse the solutions proposed to address the issues mentioned in Section 4. The different proposed solutions are described in the following subsections. In the following subsections, we describe each of the proposed approaches in more details. Since the majority of the papers is addressing scheduling issues on Hadoop framework in comparison to the other platforms (including Spark, Storm, and Mesos); this is because of the popularity of Hadoop in both academic and industrial communities. For each category, we first discuss the solutions proposed for Hadoop, and then, if applicable, we discuss solutions proposed for other platforms. Finally, we present a summary to classify and discuss the proposed approaches and the addressed issues.

5.1. Resources Utilisation-aware Scheduling

Although there is a tight dependency between the map and reduce tasks, these two phases are scheduled separately by existing schedulers. Additionally, Zhang *et al.* [30] show that the resources consumption varies significantly in these two phases. To mitigate this problem, many studies including [30] [31] [32] [33] are proposed to correlate the progress of map and reduce tasks while scheduling them and then assign them slots based on their requirements. Just to name a few, the *Coupling Scheduler* [31], *PRISM* [30], HFSP [33], and *Preedoop* [32] are proposed as fine-grained resource-aware MapReduce schedulers for Hadoop. The main goal of these schedulers is to assign available slots according to the variability of the requested resources in each phase of a Hadoop job and the task execution progress. This is in order to reduce the total execution time and to avoid a waste of resources.

Jian *et al.* [31] propose the *Coupling Scheduler*, which is composed of two main parts *(i)* a wait scheduling for the reduce tasks; and *(ii)* a random peeking scheduling for the map tasks. The coupling scheduler can reduce the average job processing time by 21.3% compared to the Fair Scheduler. However, some jobs still face long-waiting times because of other running jobs taking all reduce slots. The idea behind *Preedoop* [32] is to preempt reduce tasks that are idle and assign their allocated resources to scheduled map tasks, in order to allow for a faster processing at the map phase. This is because reduce tasks waiting for intermediate data, or results from some map tasks, often detain resources that could have been used by some pending map tasks. Liang *et al.* [32] report that *Preedoop* can reduce the execution time by up to 66.57%. However, the preemption of the reduce tasks can delay the copy/merge phases of the jobs, which may result in extra delays.

Hadoop Fair Sojourn Protocol (HFSP) [33] is a new scheduling protocol that automatically adapts to resources and workload changes while achieving resources efficiency and short response times for Hadoop. HFSP uses job size and progress information to allocate the available resources to the received tasks. HFSP uses an aging function to make scheduling decisions, such that jobs with higher priority have more chance to get the resources. The priorities of jobs are computed using the aging function. HFSP can reduce the execution time of the scheduled tasks but,

it cannot reduce the number of task failures since it is based on a preemptive technique. Moreover, HFSP delays the scheduling of the map tasks on non-local data, for a certain time and for a fixed number of attempts. This is in order to ensure the processing of the map tasks with local data and improve the performance of Hadoop. Meanwhile, tasks having a lower priority can be processed. However, postponing the processing of the map tasks several times can generate extra delays to the total execution times of the tasks. Hence, some tasks execution times can exceed their specified deadlines; resulting in tasks deadline no-satisfaction.

While processing batch jobs, Hadoop may encounter problems due to inefficient resources utilization, which may generate long and unpredictable delays. Particularly, the static and fixed configuration of slots allocated to the map and reduce tasks in Hadoop can affect their processing time and may lead to a degradation of the cluster resources. To alleviate this issue, some studies, including [34] [35] [36] [37] [38] [39], introduce the dynamic assignment of the slots to the mappers and reducers depending on their requirements.

For instance, Fair and Efficient slot configuration and Scheduling for Hadoop (*FRESH*) [37] is designed to find the matching between the slot settings and the scheduled tasks to improve the makespan (which is defined as the difference between the longest and the smallest execution time of the running tasks) while guaranteeing a fair distribution of the slots among the map and the reduce tasks. In the same line, Wolf *et al.* [34] propose a flexible scheduling allocation scheme called *FLEX* aiming to optimize the response time, deadline satisfaction rates, SLA (Service Level Agreement), and makespan of different type of Hadoop jobs, while allocating the same minimum job slots assigned in the Fair scheduler. Despite the fact that FRESH and FLEX show good performances in terms of total completion time and slot allocation, their proposed scheduling schemes should relax the scheduling decisions in terms of data locality. They only consider how to fairly distribute the slots across the scheduled tasks but, they do not take into account the necessity to schedule them as close as possible to their input data. In addition, they should take into account the remaining execution time of the scheduled jobs to better assign, on the fly, the resources slots.

FiGMR [40] is proposed as a fined-grained and dynamic scheduling scheme for Hadoop. FiGMR classifies the nodes in a Hadoop cluster into high or low level performance according to their resources utilisation, and tasks into slow map tasks and slow reduce tasks. It uses historical information from the nodes to dynamically find the tasks that are slowed by a lack of resources. Then, FiGMR launches speculative executions of the slow map and reduce tasks on the high level performance nodes in order to speed up their execution. Overall, FiGMR can reduce the execution time of tasks and improve data locality. But, it requires considerable time to find the slow tasks and to assign them to high level performance nodes, which can result in extra delays to the scheduler.

Because of the large scale of cloud environments, the applications running on top of Hadoop systems are increasingly generating a huge amount of data about the system states (*e.g.*, log-files, etc). These data can be used to make better scheduling decisions and improve the overall cluster performance. Whereas, the primary Hadoop schedulers rely only on a small amount of information about the Hadoop environment, particularly about the resources allocation/utilisation to make the scheduling decisions. Therefore, many research work [41] [30] [42] [43] have been proposed to build schedulers capable of collecting data about the resources utilisation and adapting their scheduling decisions based on the system states and the events occurring in the cloud computing environment.

For example, *HaSTE* [42] is designed as a pluggable scheduler to the existing Hadoop YARN [14]. HaSTE schedules the received tasks according to many system information like the requested resources and their capacities, and the dependencies between the tasks. It assigns the resources to tasks based on the ratio between the requested resources and the capacity of the available resources. Specifically, HaSTE measures the importance of the received tasks in order to prioritize the most important tasks in a job and to quantify the dependencies between the scheduled tasks. Despite the fact that HaSTE can optimize the resources utilisations, it is limited only to the CPU and memory resources.

Rasooli *et al.* [41] [43] propose to use the collected information about the Hadoop environment to classify the received jobs according to their resources requirements. They implement an algorithm that captures the changes on the system states and adapts the scheduling decisions according to the new system parameters (*e.g.*, queue state, free slots, etc.) in order to reduce the average execution time of the scheduled tasks. But, their proposed approach is associated with an overhead to estimate the execution time of each received job and to make the slot allocation in accordance to the captured system changes.

There are also considerable challenges to scheduling the growing number of tasks with constraints-meeting objectives. Along with the broad deployment of Hadoop schedulers, many studies [44] [45] [46] [47] [48] [49] have been proposed to improve the performance of Hadoop in terms of deadline satisfaction. In a nutshell, these schedulers identify the jobs (among the submitted ones) that could be finished within a specific deadline, then, they check the availability of resources to process the jobs. A job will be scheduled if there are enough slots to satisfy its requirements.

In [47], Bin *et al.* propose a scheduling algorithm that leverages historical information about scheduled and finished tasks and slot performance to make a decision about whether a resources slot (CPU, memory, bandwidth) is good enough for the assigned tasks, the delay threshold,

and the tasks' deadlines. Their proposed algorithm also makes a decision about whether a scheduled task should be delayed, since there will be some other available slots better than the selected ones. The proposed scheduler is able to assign the tasks to the suitable slots with acceptable delays. However, delaying small jobs while looking for the most suitable slots can affect their total completion time.

In [46], Ullah *et al.* consider the remaining execution time of each job when deciding to preempt, in order to maximize the utilization of the slots under the deadline constraints and the execution time requirements. However, they use a static approach to estimate the remaining time, which can affect the average of this value and hence negatively impact the scheduling decisions. In [48], Pletea *et al.* implement a genetic algorithm to speculate the execution time of the tasks with respect to deadline constraints and the heterogeneity of the distributed resources in a Hadoop cluster. Overall, the genetic-based algorithm must be efficient and fast in terms of execution time while providing the optimal solution to the scheduler. However, the authors do not implement any optimization function to improve the performance of their proposed algorithm; which may negatively impact the performance of the scheduler.

In [49], Khan *et al.* propose a Hadoop job performance model that can estimate the amount of required resources so that jobs are finished before their deadlines based on the estimation of job completion times. The proposed model uses historical information about job execution records and a Locally Weighted Linear Regression (LWLR) technique to determine the estimated execution time of Hadoop jobs. It could reduce the total execution time of jobs by up to 95.5% such that jobs are completed before their expected deadlines. However, it only considers the independent Hadoop jobs, which can affect the resource allocation mechanism in Hadoop.

Jiang *et al.* [50] claim that the existing scheduler in Spark does not consider any coordination between the utilization of computation and network performance, which may lead to a reduced resource utilisation. Therefore, they design and implement Symbiosis, which is an online scheduler that predicts resources imbalance in Spark cluster. Jiang *et al.* [50] propose to schedule computation-intensive tasks (with data locality) and network-intensive tasks (without data locality) on the same CPU core in Symbiosis. When several tasks are scheduled and competing on the same CPU core, they integrate a priority-based strategy to select which task to process first. Symbiosis is able to reduce the total completion times of Spark jobs by 11.9% when compared to the current scheduler of Spark framework. However, the authors do not consider the resource and network utilisation for the intermediate steps that involve especially network transfers hence, it can add extra delays to the processing of the jobs.

Apache Storm is the most popular stream processing system used in industry. It uses the default round-robin scheduling strategy, which does not consider the resources availability and demand. To alleviate this issue, R-Storm [51] is proposed to satisfy soft and hard resources constraints, minimize the network latency, and increase the overall throughput. Peng *et al.* implement a scheduling algorithm using the Quadratic Knapsack Problem (QKP) and find that R-Storm outperforms Storm in terms of throughput (30%-47%) and CPU utilisation (69%-350%).

Mesos [52] possesses a fined-grained resource sharing scheduler that controls the sharing of resources across the applications running on the platform. In other words, Mesos decides the amount of resources that can be assigned to an application, and this application decides the tasks to run on them. This approach allows the application to communicate with the available resources to build a scalable and efficient platform. Consequently, Mesos can achieve better resource utilisation and near-optimal data locality. But, it does not take into consideration the requirements of applications running on Mesos while assigning them the resources, which may result in a waste of resources.

5.2. Data Management-aware Scheduling

Data management is a hot issue that caught the attention of many researchers. This is because of its direct impact on the performance of big data platforms including those of the task scheduling techniques. For instance, the performance of big data platforms' schedulers is highly dependent on the procedures dedicated to managing the data to be processed in the computing nodes. This issue is extensively studied by researchers who aim to efficiently distribute data schemes across the nodes. In the following paragraphs, we describe different approaches proposed by researchers to improve the ***data locality***, ***data placement*** and ***data replication*** schemes.

5.2.1. Data Locality-aware Scheduling

MapReduce is widely used in many systems where there are dynamic changes over time. However, it lacks the flexibility to support small and incremental data changes. To cover this limitation, the *IncMR* framework [53] is proposed to improve the data locality incrementally. IncMR fetches the prior state of runs in the system and combines it with the newly added data. This can help find better scheduling decisions according to the new changes in the systems. So, the state of system runs periodically get updated for future incremental data changes. The conducted experiments in [53] show that their approach has a good impact on non-iterative applications running in MapReduce. Indeed, the running time is faster than the one obtained when processing the entire input data. But, IncMR is subject to high network utilisation and the large size of files storing the system states consumes resources. Therefore, it is very important to optimize the storage of the states of the system (in terms of size and location) in order

to get efficient processing times and optimize the network bandwidth.

Chen *et al.* [54] propose the Locality-Aware Scheduling Algorithm (*LASA*) in order to achieve better resource assignments and data locality in Hadoop schedulers. They present a mathematical model to calculate the weight of data interference that will be given to LASA. The data interference is derived using the number of data in each node having free slots. Next, LASA selects the node having the smallest weight and data locality to process the received tasks. But, LASA does not guarantee a fair distribution of the workload across the Hadoop nodes.

In [55], Chon *et al.* present a real-time scheduling framework for Hadoop that can guarantee data locality for interactive applications. In this work, the authors present both a scheduler and a dispatcher for Hadoop. The scheduler is responsible for assigning tasks when the required resources are available, and the dispatcher considers the data locality of these tasks. The performance of the proposed framework is evaluated using synthesized workload and it shows good results in terms of execution time and energy consumption optimization. Whereas it does not consider the priority of the tasks while assigning the tasks to the nodes having their local block data; which can affect the performance of the applications running on Hadoop.

Zaharia *et al.* [56] [57] present the Longest Approximate Time End (*LATE*) algorithm, which collects data about the running tasks and assigns weights to tasks based on their progress. Using historical information about the weights assigned to tasks in the past, LATE prioritizes the new tasks waiting to be executed. LATE predicts the finish times of each task and speculates on the ones that can meet most the response time in the future. The proposed algorithm can improve the response time of the schedulers by a factor of 2.

Later, Liying *et al.* [58] extended LATE by introducing a delay on the processing of tasks. Each task being delayed for a maximum of K times. They propose that a task should wait for T/S seconds before checking the availability of slots in the nodes having local data. In this equation, T is the average task execution time and S is the number of slots in the cluster. Since a task could be delayed up to K times, it is possible to have some tasks waiting for up to K * T/S seconds before being processed. Although, their proposed algorithm can reduce the overall response time of tasks and improve the system throughput, it has to sort twice the whole system to find the tasks having local input data and the task that will be launched speculatively; which may add extra delays to the response time. Moreover, the value of K should be suitable for the system status, to avoid the task starvation problem and system performance degradation. Also, LATE faces some issues (*i.e.*, inaccurate estimation of the remaining time of tasks) in calculating the task progress and identifying the straggling ones due to its static approach. In addition, it does not distinguish between the map and reduce tasks while calculating their progress, which may affect its performance.

Processing data within a requesting node for a data-intensive application represents a key factor to improve the scheduling performance in Hadoop. Many researchers (*e.g.*, [56] [59] [60] [61]) have been working extensively to solve this problem by evaluating the impact of many factors on the data locality. This can help identify the correlation between data locality and those identified factors and hence schedule tasks on the processing nodes as close as possible to their input data. As illustration for this solution, the research work in [59] [60] describe mathematical models and scheduling algorithms to evaluate the impact of many configuration factors on data locality. Examples of the configuration parameters can be the input data size and type, the dependency between the data input of tasks, the number of nodes, the network traffic, etc. They propose to perform a job grouping step before scheduling the tasks; the jobs belonging to the same group should be ordered based on their priority and the locality of their input data. Also, they propose to schedule multiple tasks simultaneously instead of one by one to consider the impact of other tasks that may not guarantee better scheduling's performance. These proposed algorithms can increase the number of tasks processed using local data-blocks, which can reduce their execution time. However, these solutions do not show a good improvement when job sizes are large. This is because large jobs have more distributed input data across different nodes and hence the proposed algorithms cannot guarantee to have a maximum number of local-data tasks for these jobs. The proposed approaches work well only when the job sizes are small.

Although, Hadoop and Spark are characterized by a good performance when allocating the resources and processing the received workload, they show a poor performance in handling skewed data. For instance, scheduled tasks can experience several types of failure, because of straggling tasks and skewed data. To solve this problematic issue, many studies are proposed to avoid data skewness and to find the optimal distribution of data (*e.g.*, [36] [62] [63] [64]). For example, FP-Hadoop [62] is a framework that tackles the problem of data skewness for the reduce tasks, by integrating a new phase in Hadoop job processing. The intermediate phase is called intermediate reduce (IR). The IR can process intermediate values between the map and reduce tasks in parallel with other tasks. This approach can help speedup the processing of the intermediate values even when all of them are associated with the same key. The experimental results show that FP-Hadoop has a better performance compared to Hadoop and can help reduce the execution times of reduce tasks by a factor of 10 and the total execution time of jobs by a factor of 5. But, Hadoop jobs can experience extra delays when there are no skewed data, because the IR workers add more time to the total execution time of a job.

Although, the data locality issue is tackled as one problem in the studies presented above, other research works

address it separately for map and reduce tasks, as described in the sequel.

Data Locality of Map Tasks

The pre-fetching techniques of the input data are very important to improve the data locality factor for the map tasks and avoid the data skewness problem. Particularly, there are several research work that address this issue including [65] [66] [67]. They propose pre-fetching and scheduling techniques to address the data locality of map tasks. These two techniques look for the suitable candidate input data for the map tasks. Also, they select which reducer is better in order to minimize the network traffic required to shuffle the key-value pairs. Although these techniques can improve the data locality of the map tasks, they cannot balance the load across the processing nodes. This is because the proposed techniques can only improve the number of local map tasks and does not take into account the resources utilisation and load balancing.

Asahara *et al.* [68] propose *LoadAtomizer* to improve the data locality of map tasks and minimize the completion time of multiple jobs. The LoadAtomizer strategy consists in assigning tasks on lightly loaded storage with consideration to data locality, which can balance the load between the storage nodes. LoadAtomizer can avoid I/O congestion and reduce the CPU I/O waiting time ratio of the map tasks. It could reduce the total execution time of jobs by up to 18.6%. However, it cannot reduce the data skewness for the map and reduce tasks since it aims at balancing the I/O load and increase the data locality of the scheduled tasks.

In [69] [70], the authors present scheduling techniques to improve the data locality of map tasks by dynamically collecting information about the received workload. Also, they propose to dynamically control and adjust the process responsible for allocating the slots across the received jobs to meet their specified deadlines. The obtained results show that the proposed algorithm gives a better performance in terms of the amount of transmitted data across the network and the execution time. To better improve these proposed scheduling techniques, the authors may consider the different types of received tasks (short, long, continuous, etc.) and calculate the remaining execution time of the scheduled tasks.

Data Locality of Reduce Tasks

There are a few other research work [71] [72] that are proposed to improve the data-locality for the reduce tasks. Hammoud *et al.* [71] proposed the Locality Aware Reduce Task Scheduling (*LARTS*) algorithm to maximize data locality for the reduce tasks, *i.e.*, the intermediate results generated by the mappers. LARTS uses an early shuffling technique to minimise the overall execution time by activating the reduce task after a defined percentage of mappers commit (*e.g.*, a default value of 5%). Therefore, it can help avoid data skewness and reduce the scheduling delay between the mappers and reducers. LARTS is based on locating the sweet spots of the reducers. These sweet spots can be defined as the time during which a reducer can recognize all its partitions. These spots are located by LARTS statically. Therefore, dynamic identifications of these spots can improve the performance of LARTS. Jian *et al.* [72] propose a stochastic optimization framework to improve the data locality of reducers and minimize the cost associated with fetching the intermediate data. However, this approach works under a fixed number of map and reduce slots in Hadoop; which may lead to an under or over utilization of the available resources.

Motivated by the challenges associated with the default scheduler in Storm, Xu *et al.* [73] proposed a new stream-processing framework T-Storm based on the Storm framework. In fact, Storm uses a default scheduler that assigns the received workload based on the round-robin algorithm without considering the data locality factor. Also, Storm assigns the workload to the nodes regardless of their requirements or the availability of the resources [73]. Hence, T-Storm is proposed to use run time states to dynamically assign tasks to the nodes where the data are located so that none of the workers is overloaded or underloaded, which could accelerate the task processing and minimize the online traffic in Storm. Moreover, it can achieve better performance with a smaller number of nodes since it allows fine-grained control over the nodes consolidation. The experimental analysis shows that T-Storm can achieve a better performance (up to 84% speedup), and a better data locality for the stream processing applications. Although T-STorm can achieve a good performance with 30% less worker nodes, T-Storm still lacks a fault-tolerance mechanism to handle failures in these nodes which have to process more workload than others.

5.2.2. Data Placement-aware Scheduling

Many studies are proposed to improve data placement strategies within Hadoop and provide optimized data placement schemes, *e.g.*, [74] [75] [76]. These optimized schemes can help improve the data locality for the scheduled tasks.

For instance, Xie *et al.* [74] proposed to adapt the data placement schemes in accordance to the workload distribution in Hadoop clusters. They introduce an algorithm to initially distribute input data across the nodes in accordance to the node's data processing speed. Second, they describe a data redistribution algorithm to dynamically solve the data skew issue, by reorganizing file fragments through the cluster nodes based on their computing ratios. Although these proposed algorithms can help improve the placement and the locality of data in Hadoop clusters, they do not include a mechanism to handle redundant file fragments, neither do they provide a mechanism to redistribute dynamically the data for data-intensive applications working together.

A Hierarchical MapReduce scheduler called *HybridMR* is presented in [76] to classify the received MapReduce

jobs based on their expected overhead to guide the placement between the physical and virtual machines in Hadoop clusters. In addition, *HybridMR* can dynamically organize the resources orchestration between the different map and reduce tasks and hence decrease their processing time by 40% over a virtual cluster and save around 43% of energy. Despite the fact that *HybridMR* shows a good performance, it cannot handle different types of workload in heterogeneous Hadoop environments and ensure a balanced workload between the nodes.

MRA++ [77] is a new Mapreduce framework for Hadoop proposed to handle large heterogeneous clusters. It allows Hadoop to efficiently process data-intensive applications. This is by training tasks to collect information about data distribution in order to dynamically update the data placement schemes within the framework. MRA++ is mainly composed of a data division module responsible for dividing the data for the tasks, a task scheduling module that controls the task assignment to the available slots, a clustering control module, that controls task execution, and a measuring task module that controls and distributes the data. MRA++ can improve performance of Hadoop by 66.73%. It can also reduce the network traffic by more than 70% in 10 Mbps networks. But, it adds extra delays to the tasks' processing times since they are collecting more information and have to wait for the measuring task module to assign them to the appropriate nodes.

5.2.3. Data Replication-aware Scheduling

Several studies address the problem of data replication in Hadoop to improve storage space utilization, *e.g.*, [78] [79] [80]. For instance, Jin *et al.* [78] propose the Availability-Aware MapReduce Data Placement (*ADAPT*) algorithm to optimally dispatch data across the nodes according to their availability, to reduce network traffic without increasing the number of data replica. Their strategy can improve network traffic, however, it may lead to more disk utilization.

Ganesh *et al.* [79] propose *Scarlett*, which uses a proactive replication scheme that periodically replicates files based on the predicted popularity of data. In other words, Scarlett calculates a replication factor for the data based on their observed usage probability in the past history in order to avoid the problem of data skewness. Scarlett is an off-line system that improves data replicas using a proactive approach but, many changes can occur in a Hadoop storage system including recurrent as well as nonrecurrent changes.

While Scarlett uses a proactive approach, Abad *et al.* [80] present the Distributed Adaptive Data REplication (*DARE*) algorithm, which is a reactive approach to adapt the data popularity changes at smaller time scales. The DARE algorithm aims at determining how many replicas to allocate; and at controlling where to place them using a probabilistic sampling and competitive ageing algorithm. As a result, the data locality factor in DARE is improved by 7 times when compared to the FIFO scheduler and by 85% in comparison to the Fair scheduler. However, both Scarlett and DARE do not take into account data with low replica factors.

5.3. Fairness-aware Scheduling

In big data platforms' clusters, data locality and fairness represent two conflicting challenges. Indeed, to achieve a good data locality, a maximum number of tasks should be submitted close to their computation data. However, to achieve fairness, resources should be allocated to the tasks after being requested in order to reduce tasks delays [57]. Many research work including [37] [81] [82] [83] [84] [85] are proposed in the available literature to solve the above issues.

Jiayin *et al.* [37] present FaiR and Efficient slot configuration and Scheduling algorithm for Hadoop (*FRESH*), to find the matching between the submitted tasks and the available slots. FRESH can help not only minimize the makespan but, also fairly assign available resources across the scheduled tasks. In Hadoop, each node has a specific number of slots. However, the Hadoop scheduler continuously receives concurrent jobs that require different slots configurations. Therefore, Jiayin *et al.* [37] extend FRESH by adding a new management plan to dynamically find the best slot setting. In other words, FRESH allows to dynamically change the assignment of slots between the map and reduce tasks according to the availability of slots and the requirement of the tasks. After a slot finishes its assigned task, FRESH can assign it to another task. While FRESH can improve the assignment of slots and the fairness of the distribution of resources among the scheduled tasks, it does not ensure a better memory usage.

Isard *et al.* [81] propose Quincy, which is a flexible and efficient scheduling algorithm to compute the scheduling distribution among the different nodes with a min-cost flow while improving data locality, fairness and starvation freedom factors. However, Quincy is only formulated based on the number of computers in a cluster and there is no effort to dynamically reduce its cost in terms of data transfer.

In [82], Yin *et al.* show that processor-based schedulers like the Fair scheduler can lead to a degradation of performance in terms of execution time, in a multi-user environment. Therefore, they propose the Hybrid Parallel pessimistic Fair Schedule Protocol (*H-PFSP*), which is able to finish jobs later than the Fair Scheduler and improve the mean flow time of jobs while improving the fairness between the tasks and jobs. The *H-PFSP* use information about the finished tasks over time to estimate the remaining execution time of the scheduled jobs at predefined intervals and make incremental estimations updates. The *H-PFSP* can reduce the total execution time but, it cannot guarantee an efficient resources utilisation in the cluster.

Zhao *et al.* [83] describe a scheduling algorithm based on a multi-queue task planning to adjust the maximum number of tasks assigned to each node by defining the value of fairness threshold "*K%*". The K%-Fairness scheduling

algorithm can be suitable for different types of workloads in MapReduce to achieve maximum of data locality under this constraint. However, this approach cannot support much continuous/dependent jobs in the queue since it cannot decide how to fairly distribute them (due to tight dependencies between them) and reduce the associated overhead while processing them.

Phuong *et al.* [84] propose a HyBrid-Scheduling (*HyBS*) algorithm for Hadoop. It is dedicated for processing data-intensive workloads based on the dynamic priority and data locality of the scheduled tasks. In other words, it uses dynamic priorities information, estimated map running times, and service level values defined by the user to minimize the delays for concurrent running tasks which may have different lengths. *HyBS* can guarantee a fair distribution between the map and reduce tasks. Also, it decreases the waiting time between the map and reduce tasks by resolving data dependencies for data intensive MapReduce workloads. This is by assigning a dynamic priority, obtained from historical Hadoop log files, to the different tasks received in order to reduce the latency for different length (in terms of execution time) concurrent jobs. *HyBS* is using a greedy fractional Knapsack algorithm [84] to assign jobs to the appropriate processing nodes.

The authors of [85] propose *Natjam* to evaluate the smart eviction policies for jobs and tasks, the priorities for real time job scheduling and the resources availability and usage. Natjam is based on two main priorities policies. These policies are based on the remaining time of each task: Shortest Remaining Time (SRT) in which tasks characterized by the shortest remaining time are the candidate to be suspended; and Longest Remaining Time (LRT) in which tasks characterized by the longest remaining time will be suspended. The two proposed policies that are based on priorities aim to reduce the execution time of each task. Next, they propose *Natjam-R*, a generalization of *Natjam*, which specifies hard and fix deadlines for jobs and tasks [85]. So, the deadline of Hadoop jobs can automatically define the priority of the jobs and their composing tasks for accessing the resources slots. This approach was found to have a negative impact (*i.e.*, delay) on short running tasks that have low priorities, since they can get evicted several times.

In [86], Guo *et al.* present FlexSlot a task slot management scheme for Hadoop schedulers that can identify the straggling map tasks and adjust their assigned slots accordingly. This approach can accelerate the execution of these straggling tasks and avoid extra delays. FlexSlot changes the number of slots on each node in Hadoop according to the collected information about resource utilisation and the straggling map tasks. Hence, the available resources in Hadoop cluster are efficiently utilised and the problem of data skew can be mitigated with an adaptive speculative execution strategy. The obtained results show that FlexSlot could reduce the total job completion times by up to 47.2% compared to the Hadoop scheduler. However, FlexSlot generates a delay that can impact the processing of the Hadoop job since it is using the task-killing-based approach in the slot memory resizing. In addition, FlexSlot allows to kill tasks multiple times, which may generate not only extra delays but also may cause the failure of the whole job.

5.4. Workload Balancing-aware Scheduling

Distributing the received loads across computing nodes represents a crucial problem in big data platforms' systems. An efficient distribution can help improve the resources utilisation and guarantee a fair distribution of tasks to be processed, resulting in a better performance for their schedulers.

For instance, Chao *et al.* [87] report that the First Come First Served (FCFS) strategy works well only for jobs belonging to the same class (*e.g.*, having the same size, the same resources requirements). Thus, they propose a *Triple-Queue Scheduler*, which dynamically classifies the received Hadoop jobs into three different categories based on their expected CPU and I/O utilisation. Also, they integrate a workload prediction mechanism called *MR-Predict*, which determines the type of the workloads on the fly and distributes them fairly (based on their type) across the different queues. MR-Predict can increase the map tasks throughput by up to 30% and reduce the total makespan by 20% over the Triple-Queue scheduler. However, it still faces other issues to efficiently manage the resources utilisation; to reduce the resources waste and to improve the data locality.

Hong *et al.* [88] propose a load-driven Dynamic Slot Controller (*DSC*) algorithm that can adjust the slots of map and reduce tasks according to the workload of the slave nodes. Hence, DSC can improve the CPU utilisation by 34% and the response time by 17% when processing 10 GB of data. But, the DSC algorithm does not take into account the issue of data locality while balancing the load between the nodes.

Teng *et al.* [89] propose Shortest Period Scheduler (*SPS*) to ensure that most of the jobs are finished before their specified deadlines. SPS supports preemption and can make dynamic decisions when new workflow plans are received periodically in the scheduler. SPS is limited to scheduling the independent tasks within the received workflow. However, it should cover dependent tasks and analyse the impact of the communication between the scheduled tasks on their expected deadline and the resources utilisation.

Lu *et al.* [90] propose a Workload Characteristic Oriented (*WCO*) scheduler to consider the characteristics of the running workloads and make smart decisions that can improve the resource utilisation. The WCO scheduler is able to dynamically detect the difference between the received and the running workloads. Consequently, it can help balance the CPU and I/O usage among Hadoop nodes which could improve the system throughput by 17%. WCO can be improved by enhancing its static analysis method used for the workload characteristics.

Cheng *et al.* [91] proposed to use the configuration of large MapReduce workloads to design a self-adaptive task scheduling approach. Their proposed solution consists of an Ant-based algorithm that allows for an efficient workload distribution across the available resources, based on the tasks characteristics. As a result, their approach can improve the average completion time of the scheduled tasks by 11%. Also, they find that their proposed Ant-based algorithm is more suitable for large jobs that have multiple rounds of map task execution. However, this proposed algorithm cannot cover multi-tenant scenarios in MapReduce. In addition, Cheng *et al.* [91] do not provide details about the optimization of the Ant-algorithm to reduce its execution overhead.

Zhuo *et al.* [92] propose a scheduling algorithm (to optimize the workflow scheduling) in which jobs are represented as Directed Acyclic Graph (DAG) and classified into I/O intensive or computations intensive jobs. Then, the scheduler can assign priorities to the jobs based on their types and assign the available slots with respect to data locality and load balancing. But, this proposed approach was found to work well only for large jobs. It can negatively impact the performance of small jobs.

Li *et al.* [93] propose WOrkflow over HAdoop (*WOHA*) to improve workflow deadline satisfaction rates in Hadoop clusters. WOHA relies on the job ordering and progress requirements to select the worklflow that falls furthest from its progress based on the Longest Path First and Highest Level First algorithms. As a result, WOHA can improve workflow deadline satisfaction rates in Hadoop clusters by 10% compared to the existing scheduling solutions (FIFO, Fair and Capacity schedulers). WOHA uses the workloads received over time to estimate the deadline of each task that are not known by the scheduler ahead of time. In addition, the dynamic nature of Hadoop workloads may affect the performance of the scheduler. But, developers of WOHA do not include these two criteria while implementing it.

In [41], Rasooli *et al.* propose a hybrid solution to select the appropriate scheduling approach to use based on the number of the incoming jobs and the available resources. This proposed solution is a combination of three different schedulers: FIFO, Fair sharing and Classification, and Optimization based Scheduler for Heterogeneous Hadoop (*COSHH*). The COSHH scheduler uses Linear Programming (LP) to classify the incoming workloads and find an efficient resources allocation using job classes requirements. The aim of this hybrid scheduler is to improve the average completion time, fairness, locality and scheduling times in order to improve Hadoop's scheduling performance. The FIFO algorithm is used for under-loaded systems, the Fair Sharing algorithm is used when the system is balanced and the COSHH is used when the system is overloaded (*i.e.*, peak hours). Rasooli *et al.* define three different usage scenarios and specify when to use each of them, however, they do not provide thresholds that can be used to decide about which scheduler to follow.

Sidhanta *et al.* [94] propose OptEx, which is a closed-form model that analytically analyses and estimates the job completion time on Spark. OptEx model uses the size of input dataset, the number of nodes within the cluster and the number of operations in the job to be scheduled, to estimate the total completion time of the given job. The results show that it can estimate the job completion time in Spark with a mean relative error of 6% when integrated with the scheduler of Spark. Furthermore, OptEx can estimate the optimal cost for running a Spark job under a specific deadline in the Service Level Objective (SLO) with an accuracy of 98%. Although OptEx is the first model in the open literature to analytically estimate the job completion time on Spark, it only considers the job profiles of PageRank and WordCount as parameters along with the size of the cluster and the dataset. This model cannot be representative for real cluster where different workload having different profiles are running.

Sparrow [95] is a distributed scheduler that allows the machines to operate autonomously and support more requests from different applications running Hadoop or Spark. In other words, Sparrow is a decentralized scheduler across a set of machines that operate together to accommodate additional workload from users. When a scheduler in a machine fails, other machines may accept its received requests and process it according to their availability. The challenge in Sparrow consists in balancing the load between the machines' schedulers and providing shorter response times, especially when the distributed schedulers make conflicting scheduling decisions. Sparrow uses three main techniques to improve the performance of its schedulers: *Batch Sampling*, *Late Binding*, and *Policies and Constraints*. *Batch Sampling* schedules m tasks in a job on the lightly loaded machines, rather than scheduling them one by one. *Late Binding* places the m tasks on the machine queue only when it is ready to accept new tasks to reduce the waiting time in the queue that is based on FCFS. The *Policies and Constraints* are to control the scheduling decisions and avoid the conflicts on the scheduling decisions. Sparrow allows to distribute the received workload and balance it across the available workers in a shorter time. While Sparrow can reduce the execution time of the jobs by up to 40%, it lacks mechanisms to take into account the requirements of the received workload while distributing them across the nodes. Also, it does not consider the resources availability on each node, the schedulers accept the new requests if the queue is not yet empty, which can overload the machines. Moreover in case of a scheduler failure, the meta-data scheduling of the tasks running on that machine will not be shared with the other machines.

5.5. Fault-Tolerance-aware Scheduling

Although, Hadoop, Spark, Storm, and Mesos are equipped with some built-in fault-tolerance mechanisms,

they still experience several tasks failures due to unforeseen events in the Cloud. For example, the HDFS in Hadoop keeps multiple replicas of data blocks on different machines to ensure an effective data restoration in case of a node failure. The failed map and reduce tasks will be rescheduled on other nodes and re-executed from scratch. This fault-tolerant solution is associated with a high cost because of the task re-execution events, which can significantly affect the performance of the Hadoop scheduler. To address the aforementioned limitations, researchers have proposed new mechanisms to improve the fault-tolerance of Hadoop (*e.g.*, [96] [97]).

Quiane-Ruiz *et al.* [96] proposed Recovery Algorithm for Fast-Tracking (*RAFT*) for Hadoop to dynamically save the states of tasks at regular intervals and at different stages. This approach allows the JobTracker to restart the tasks from the last checkpoint in the event of a failure. Indeed, RAFT enables the Hadoop scheduler to not re-execute the finished tasks of the failed jobs since their intermediate data are saved. So, the scheduler will only re-execute the failed tasks. As a result of this strategy, RAFT can reduce the total execution time of tasks by 23% under different failure scenarios.

Bian *et al.* [97] propose an approach that dynamically detects the failures of scheduled tasks and makes backups of the tasks. In case of a failure, the scheduler would launch the failed tasks on other nodes without losing their intermediate data. Although the two works presented in [96] [97] can improve the fault-tolerance of the system, they do not provide a mechanism to improve the availability of the checkpoints and the used backups.

In [98], Xu *et al.* claim that the long delays of jobs are due to the straggling tasks and that the LATE scheduler [56] can make inaccurate estimations of the remaining time of tasks, which may lead to resource waste. Thus, they propose a dynamic tuning algorithm that uses historical information about tasks progresses to tune the weights of each map and reduce tasks. In addition, they design an evaluation approach that decides whether to launch a straggling task on another node when there are free slots in order to reduce the execution time and resources waste. However, they do not propose a mechanism to distinguish between different types of straggling tasks, *i.e.*, whether it is a map or a reduce task. This is particularly important since it can affect the speculative executions.

Dinu *et al.*[99] analyse the behavior of the Hadoop framework under different types of failures and report that the recovery time of the failed components in Hadoop can be long and can cause important delays, which may affect the overall performance of a cluster. They claim that sharing information about straggling and failed tasks between JobTrackers and TaskTrackers, can significantly improve the success rate of task executions.

To quickly detect Hadoop nodes failures, Hao *et al.* [100] develop an adaptive heartbeat interval module for the JobTracker. Using this module, the JobTracker can dynamically estimate its *expiry interval* for various job sizes. They show that when the expiry interval decreases (which means that the average number of heartbeats sent to the JobTracker increases), the total execution time of small jobs decreases. In addition, they propose a reputation-based detector to evaluate the reputation of the workers. A worker will be marked as failed when its reputation is lower than a threshold. They claim that if equipped with their proposed tools, Hadoop can detect node failures in shorter times and balance the load received by the JobTracker to reduce job execution times. However, they only consider the job size when deciding to adjust the heartbeat interval and they do not include other parameters related to the nodes environment (*e.g.*, running load, availability of resources, failure occurrence).

In addition to the above work, *Astro* [101] is designed to predict anomalies in Hadoop clusters and identify the most important metrics contributing towards the failure of the scheduled tasks using different machine learning algorithms. The predictive model in Astro can detect anomalies in systems early and send a feedback to the scheduler. These early notifications can improve resources usage by 64.23% compared to existing implementations of Hadoop schedulers. Astro can be improved by adding mechanisms that enable a better distribution of workloads between the nodes of the cluster. This would reduce the execution time of the scheduled tasks by 26.68% during the time of an anomaly.

The execution of MapReduce jobs in Hadoop clusters can undergo many failures or other issues, which may affect the response time and delay submitted jobs. Preemtion is proposed as an effective solution to identify straggling jobs and tasks in advance and make quick scheduling decisions to prevent a waste of resources. Different approaches based on speculative executions have been proposed to address this issue in distributed Hadoop clusters:

Qi *et al.* [102], develop an algorithm called Maximum Cost Performance (*MCP*), to improve existing speculative execution strategies. However, MCP was found to negatively impact the scheduling time of some jobs (batch jobs in particular) [103].

The Combination Re-Execution Scheduling Technology (*CREST*) [104] algorithm is proposed to improve MCP, by considering data locality during the speculative scheduling of slow running tasks. The authors propose to optimally re-execute map tasks having local data instead of launching speculative tasks without considering data locality. However, there is a cost associated with the replication of executed map tasks.

Self-Adaptive MapReduce scheduling (*SAMR*) uses hardware system information over time to estimate the progress of the tasks and adjust the weights of the map and reduce tasks, in order to minimize the total completion time [105]. However, SAMR does not consider jobs characteristics in terms of size, execution time, weights, etc.

Enhanced Self-Adaptive MapReduce scheduling (*ESAMR*) is designed to overcome the drawbacks of

SAMR and consider system information about straggling tasks, jobs length, etc. ESAMR uses the K-means clustering algorithm to estimate tasks execution times and identify slow tasks. It is more accurate than SAMR and LATE [56]. Although ESAMR can identify straggling map and reduce tasks and improve the execution time of jobs, it does not provide rescheduling mechanisms for these straggling tasks and does not improve the number of the finished tasks.

AdapTive faiLure-Aware Scheduler (*ATLAS*) [28] is proposed as a new scheduler for Hadoop that adapts its scheduling decisions to events occurring in the cloud environment. ATLAS can identify task failures in advance and adjust its scheduling decisions on the fly based on statistical models. It can reduce task failure rates, resources utilisation and total execution time. However, it requires training its predictive model at fixed time intervals, which may negatively impact the scheduling time. Also, it may face problems to find the appropriate scheduling rule or it can give wrong predictions that can cause the failure of tasks.

Yildiz *et al.* [106] [107] propose *Chronos*, a failure-aware scheduling strategy that enables early actions to recover the failed tasks in Hadoop. Chronos is characterized by a pre-emption technique to carefully allocate resources to the recovered tasks. It can reduce the job completion times by up to 55%. However, it is still relying on wait and kill pre-emptive strategies, which can lead to resource waste and degrade the performance of Hadoop clusters.

5.6. Energy Efficiency-aware Scheduling

The total energy consumption of the applications running on big data platforms depends on many factors including the number of the low-load nodes and the processed load on each node. Several studies addressed the issue of finding good task assignments while saving the energy, *e.g.*, [108] [109] [110] [111].

Lena *et al.* [108] propose to model the problem of saving energy on MapReduce jobs as an integer programming problem and design two heuristics Energy-MapReduce Scheduling Algorithm I and II (EMRSA-I and EMRSA-II). The proposed model considers the dependencies between the reduce tasks and the map tasks such that all tasks are finished before their expected deadlines, while the main goal of the proposed approach is to minimize the amount of energy consumed by these map and reduce tasks. EMRSA-I and EMRSA-II are evaluated using TeraSort, PageRank, and K-means clustering applications and are able to reduce the energy consumption by up to 40% on average, when compared to the default scheduler of Hadoop. In addition, they can reduce the makespan between the processed MapReduce jobs. However, in the proposed model, the authors assume that map tasks belonging to the same job should all receive resources slots (same assumption for the reduce tasks), before the execution of the job. However, this is not generally the case in Hadoop and such restriction can delay the execution of a MapReduce job if even a single map or reduce task fail to obtain a slot.

Wen [109] propose a dynamic task assignment approach to reduce the overall system energy consumption for dynamic Cloud Hosts (CHs). The idea of the approach is to have a set of power-on/suspending thresholds to satisfy the constant and variable traffic loads, migration overhead, and the processing power between the CHs. Based on the proposed thresholds, the Hadoop scheduler can dynamically assign tasks to satisfy those constraints and achieve better energy efficiency. The evaluation of these schemes shows that setting the thresholds between the CHs can help obtain the lowest energy consumption and acceptable execution times for Hadoop jobs. However, there is an overhead that comes when suspending or powering on the CHs, which can affect the network traffic in Hadoop, especially when the frequency of these two operations is high.

Paraskevopoulos *et al.* [110] propose a strategy to schedule the tasks for Hadoop, while balancing between energy consumption and response time. Their proposed strategy can help identify the nodes in a cluster that can satisfy the constraint of less energy in a reasonable response time. Next, it can determine which nodes should be turned on–or–off, and when that should be done, based on the derived nodes and the received workload in the cluster. The experimental results show a significant improvement on energy consumption without sacrificing the scheduler performance. In this work, the authors only consider the response times of the jobs when deciding about the best task scheduling policies that can minimize energy consumption. They do not consider balancing the workload between the nodes that have the highest impact on the energy consumption of a Hadoop cluster.

Chen *et al.* [111] introduce a scheduling approach for Hadoop to minimize the energy consumption of MapReduce jobs. The proposed approach consists in dividing the jobs into time-sensitive jobs and less time-sensitive ones. The former group of jobs are run on dedicated nodes where there are enough resources, while the later ones run on the remaining nodes in the cluster. Chen *et al.* [111] introduce a management framework for Hadoop named Berkeley Energy Efficient MapReduce (BEEMR). BEEMR is able to reduce the energy consumption in Hadoop clusters by 40-50% under tight design constraints. However, although BEEMR can achieve energy savings for workloads with significant interactive analysis, BEEMR cannot reduce the processing times of long running jobs that have inherently low levels of parallelism, even when all resources in the cluster were available [111].

5.7. Discussion

In the previous subsections, we describe the different types of scheduling approaches available in the open literature to solve the issues presented in Section 4. Table 3 presents a classification of these approaches and the addressed issues. The main addressed issues in the studied

papers are: improve the resources utilisation, reduce tasks delays, tasks dependency consideration, reduce the execution times of tasks and jobs, improve the deadline satisfaction of tasks, reduce network traffic, improve the data locality/placement/replication strategies, reduce the data skew, balance the workload, reduce failures rates (tasks, workers, etc), and reduce the amount of energy consumed in big data platforms. The proposed approaches can be classified into three main categories: *dynamic*, *constrained*, and *adaptive*. We observe that most of the existing solutions propose to collect and use data about the environment where the computations are processed (*e.g.*, clusters, machines, workers). This can be explained by the dynamic behavior and structure of the cloud computing environment where Hadoop, Spark, Storm, and Mesos platforms are deployed. This requires to adapt the scheduling decisions of these platforms according to the continuous changes across the clusters.

In general, we observe the lack of formal description of the addressed issues and the proposed solutions in the papers analysed in this SLR. Indeed, we notice that most papers conduct empirical studies (*e.g.*, [34] [35] [36]) and very few work propose analytical models (*e.g.*, [24] [26] [29]) to solve the scheduling issues. So, an interesting direction could be to improve the empirical studies by developing formal models in order to improve the performance of the Hadoop, Spark, Storm, and Mesos' schedulers. Another concern is the benchmarks (*e.g.*, WordCount, TeraSort) used to implement and build the proposed solutions. The use of these benchmarks is highly dependent on the objective of the study (*e.g.*, resources optimization, failures recovery). The absence of dataset to configure and define the parameters of these benchmarks may lead to biased results. Furthermore, we find that several studies conducted by the academics do not become commercialized and part of Apache projects (Hadoop, Spark, and Storm) or Mesos. Finally, we can conclude that applying and adapting the proposed solutions for Hadoop to Spark, Storm, and Mesos could be an interesting direction since we notice that only a few work are done to improve the performance of Spark, Storm, and Mesos compared to Hadoop.

6. Research Directions on Task Scheduling in Big Data Infrastructures

In this section, we present some suggestions for potential research directions using the results from the above paragraphs. These suggestions can help build a roadmap for future work related to task scheduling in the studied platforms, *i.e.*, Hadoop, Spark, Storm, and Mesos.

6.1. Resources Utilisation-aware Scheduling

During our study, we observe that existing work in the literature propose different approaches to assign the map and reduce slots and evaluate the performance of the scheduler. Moreover, most of the studies randomly select the map tasks that satisfy the slots requirements. But, it is very important to include the data locality issue while scheduling the map tasks in order to improve their execution and hence, avoid the data skewness problem and reduce the execution times of the reduce tasks (as mentioned in Section 5). Besides, we notice that the task preemption while occupying or waiting for a slot can cause unpredictable delays. So, an efficient approach is required to manage task preemption in a way that do not generate an overhead and avoid task starvation. Furthermore, analysing several factors (*e.g.*, queue state, available slots, received workload, number of nodes) on the resources utilisation can be helpful to guide the scheduler to change its scheduling decisions based on the events occurring in its environment. The scheduler may consider different constraints-objectives along with a fair distribution of load constraint while scheduling the tasks. Examples of these constraints can be the Service Level Agreement (SLA), the number of tasks having local data, the transferred data in the network, etc. Finally, more studies need to be done in order to improve the performance of Spark, Mesos, and Storm schedulers in terms of resources utilisation since we find very few work done in this aspect. For instance, considering the characteristics of workload running on Spark, Mesos, and Storm can guide the scheduler to make better scheduling decisions while assigning the available resources to the tasks. Also, we believe that some of the existing solutions proposed to improve the performance of Hadoop-Mapreduce can be reused and adapted for the other platforms. To do that, one should consider the data structure/format and the way these data are processed within these platforms. For example, new approaches should consider the characteristics of the in-memory operations performed in Spark using the Resilient Distributed Dataset (RDD). The allocation of resources to tasks in Spark should be done, taking into account the amount of memory required to store the intermediate data between the tasks, since a Spark job does the whole computation and then stores the final output to the disk. In the case of Storm, it is important to consider the structure of the applications processing the spouts and the bolts, and the dependency between these tasks, especially the bolts that process the data read from input streams or the generated output of other bolt tasks. This is because there might be some bolt tasks running in sequential or/and parallel. Therefore, one should consider the parallel and the sequential aspects while scheduling the tasks in Storm. For Mesos, the type of frameworks should be considered (*e.g.* CPU-intensive, memory-intensive or network-intensive frameworks), when adapting existing solutions to improve the resources utilisation. This is because the type of the framework can affect the performance of the assigned resources by Mesos. So, Mesos should consider not only the amount of the assigned resources to each framework but also the type of resources to offer them. In addition, the two levels of the scheduling process of Mesos require synchronization between them.

Table 3: An Overview of Task Scheduling Issues and Solutions in Big Data Platforms

Issue/Solution	Map/Reduce Dependency	Slot Assignment	Data Collecting	Load Profiling	Prefetchning & Shuffling Techniques	Recovery Techniques	Failure Prediction
Resources Variability	[30] [31] [32] [33]	[34] [35] [36] [37] [38] [39] [81]	[50] [52]				
Task Delays		[34] [35] [36] [37] [38] [39] [81]	[81] [83] [84] [85]				
Dependency Tasks	[42] [51]		[42] [51]				
Execution Time		[94]	[41] [30] [42] [43]				
Deadline Satisfaction			[44] [45] [46] [47] [48] [49] [51]				
Data Locality/ Network Traffic	[56] [57] [58]		[51] [52] [53] [54] [55] [56] [57] [58] [73]	[56] [59] [60] [61]			
Data Skew	[36] [62] [63] [64]	[86]	[69] [70]	[68]	[65] [66] [67] [71] [72]		
Data Placement			[74] [75] [76]				
Data Replication			[78] [79] [80]				
Unbalanced Workload			[89] [91] [93] [95]	[87] [88] [90] [92] [41]			
Failures Rates			[96] [97] [104] [98] [99] [100] [102] [105]			[96] [97] [104] [98] [99] [100] [102] [105]	[101] [28] [106] [107]
Energy Consumption	[108] [109]	[108] [109]	[110]	[111]			

6.2. Data Management-aware Scheduling

While reviewing the data-management aware scheduling solutions in the literature, we notice that the proposed schemes that place or duplicate the data across the cluster nodes are not made based on a workload analysis. Therefore, analysing the impact of different workloads, data placements and replication mechanisms are needed to improve the data locality of the scheduled tasks. Moreover, having a large number of local tasks may cause an unbalanced workload across the processing nodes. Hierarchical scheduling can be a solution for this issue; this approach consists in having one scheduling layer to consider the data locality and another scheduling layer to handle workload balancing. These two layers should communicate their global and local information to cooperate together. In addition, we observe that most of the solutions that try to achieve data locality are characterized by an overhead due to the cost of finding the optimal distribution of file fragments. This would significantly affect the performance of the scheduler. So, better solutions need to be developed to reduce this overhead. On the other hand, existing scheduling solutions cannot guarantee high data locality for large jobs since there is a lot of data to be transferred. Therefore, efficient approaches should be developed in this direction in order to handle different job scales. Moreover, we notice that the majority of the studies we find are related to Hadoop schedulers. So, more work should be done to analyse the performance of Spark, Mesos and Storm in terms of data locality, replication and duplication.

6.3. Fairness-aware Scheduling

Distributing the available resources slots among the scheduled tasks is important in order to avoid the starvation problem. However, to the best of our knowledge, the research studies that address this issue do not consider the difference between the map and reduce tasks during slots assignments. Indeed, while estimating the remaining execution time, the proposed solutions do not distinguish between them; which may affect the slots assignment since the map and reduce tasks have different slot requirements. In addition, developing a constraint-solver that combines several objectives including data locality, resources utilisation and fairness can significantly improve the performance of schedulers on these platforms. Implementing heuristics can solve this constrained-problem, however, it may cause an overhead. Moreover, we notice that most of the studies in this research direction do not handle fairness for continuous jobs that can occupy the available slots for longer time than the small ones. Therefore, proposing an efficient approach that can estimate the amount of slots required to guarantee successful processing for both continuous and non-continuous jobs while having a fair distribution for the available slots would be useful. Also, it is very important to reduce the amount of data communication for dependent jobs, in order to mitigate the overhead due to the transfer of intermediate data.

6.4. Workload Balancing-aware Scheduling

To improve the performance of schedulers of the studied platforms, one can develop efficient solutions that estimate the remaining execution time of the scheduled tasks

based on their progress rate in order to redistribute the loads across the nodes. Indeed, existing schedulers use a simple approach to estimate the remaining execution time and hence the resulting average execution time cannot be used in heterogeneous clusters and may lead to unbalanced workloads. Besides, the performance of the prediction models used to estimate the type of received workloads can significantly affect the load distribution. Hence, it is required to build robust models with high accuracy, to predict the characteristics of the upcoming loads. Based on these predictions, the scheduler can make better decisions when assigning the slots and guarantee data locality for the scheduled tasks. Moreover, it is very important to propose a model to adjust the scheduling decisions considering dependencies between the tasks within the workload. This can help reduce the overhead to communicate the intermediate results and allow for faster processing. Also, it can reduce the amount of data transferred in the network. The analysis of the impact of virtual machines placements on the processing of the load could enable a better workload distribution.

6.5. Fault-Tolerance-aware Scheduling

Reducing the occurrence of failures in big data platforms is very important in order to improve the resources utilisation and the performance of the scheduled tasks. However, existing schedulers only make use of a limited amount of information when re-executing failed tasks. This is due to the lack of information sharing between the different components of these frameworks. Adaptive solutions that collect data about the events occurring in the cloud environment and adjust the decisions of the scheduler accordingly could help avoid decisions leading to task failures. Moreover, the speculative execution still experiences many failures and waste of resources due to inaccurate estimations of the scheduled tasks progresses or the availability of resources. This can affect their starting time and the number of speculatively executed tasks. Therefore, it is very important to analyse the impact of different factors on the start time and the number of speculative executions required for the straggling tasks. Finally, it is very important to distinguish between the failure of a map and a reduce task since they have different impacts on the processing of tasks.

6.6. Energy-Efficiency-aware Scheduling

Determining the configuration for big data platforms like Hadoop or Spark can be very helpful to achieve energy savings and make efficient scheduling decisions. Also, analysing the correlation between the number of nodes in a cluster and the amount of energy consumed can be relevant in order to specify the nodes to turn on–or–off, so that the number of active nodes satisfy the requirements of the received tasks. Moreover, analysing the level of parallelism in a cluster when the number of active nodes increases can be an important direction to guide the scheduler to scale up–or–down the level of parallelism for the scheduled tasks especially for the long jobs. In addition, most of the studies proposed to improve the performance of Hadoop schedulers does not consider the impact of delaying the execution of tasks on the overall performance of the scheduler in terms of users' requirements. Another aspect that can be interesting for future studies is to analyse the impact of frequencies at which machines are turned on or off in a cluster, especially large ones, on the amount of energy consumed. Although turning off some nodes in a cluster can help reduce energy consumption, this can generate more traffic on the network since the scheduled tasks may not find their data on the nodes where they will be executed; which could increase the number of data transfer in the cluster.

7. Conclusion

In recent years, task scheduling has evolved to become a critical factor that can significantly affect the performance of cloud frameworks such as Hadoop, Spark, Storm and Mesos. This crucial issue is addressed by many researchers. However, to the best of our knowledge, there is no extensive study on the literature of task scheduling for these frameworks that classifies and discusses the proposed approaches. Hence, we perform a SLR to review existing literature related to this topic. In this work, we review 586 papers and identify the most important factors affecting the performance of the proposed schedulers. We discuss these factors in general with their associated challenges and issues namely, resources utilisation, total execution time, energy efficiency etc. Moreover, we categorize the existing scheduling approaches from the literature (*e.g.*, adaptive, constrained, dynamic, multi-objective) and summarise their benefits and limitations. Our mapping study allows us to classify the scheduling issues in different categories including resources management, data management (data locality, data placement and data replication), fairness, workload balancing, fault-tolerance, and energy efficiency. We describe and discuss the approaches proposed to address these issues, classifying them into four main groups; dynamic scheduling approaches, constrained scheduling approaches, and adaptive scheduling approaches. Finally, we outline some directions for future research that can be included in a roadmap for research on task and jobs scheduling in Hadoop, Spark, Storm and Mesos frameworks.

References

References

[1] S. Kurazumi, T. Tsumura, S. Saito, H. Matsuo, Dynamic Processing Slots Scheduling for I/O Intensive Jobs of Hadoop MapReduce, in: Proceedings of International Conference on Networking and Computing, 2012, pp. 288–292.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster Computing with Working Sets, in: Proceedings of USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, 2010, pp. 1–7.

[3] B. Peng, M. Hosseini, Z. Hong, R. Farivar, R. Campbell, R-Storm: Resource-Aware Scheduling in Storm, in: Proceedings of Annual Middleware Conference, 2015, pp. 149–161.

[4] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: A Platform for Fine-grained Resource Sharing in the Data Center, in: Proceedings of USENIX Conference on Networked Systems Design and Implementation, 2011, pp. 295–308.

[5] T. Jian, M. Shicong, M. Xiaoqiao, Z. Li, Improving ReduceTask Data Locality for Sequential MapReduce Jobs, in: Proceedings of IEEE INFOCOM, 2013, pp. 1627–1635.

[6] S. Patil, S. Deshmukh, Survey on Task Assignment Techniques in Hadoop, International Journal of Computer Application 59 (14) (2012) 15–18.

[7] B. Rao, L. Reddy, Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments, CoRR abs/1207.0780.
URL `http://arxiv.org/abs/1207.0780`

[8] N. Singh, S. Agrawal, A review of research on MapReduce scheduling algorithms in Hadoop, in: Proceedings of International Conference on Computing, Communication Automation, 2015, pp. 637–642.

[9] B. Kitchenham, Procedure for Performing Systemic Reviews, Tech. rep., Keele University and NICTA, Australia (2004).

[10] K. Lee, Y. Lee, H. Choi, Y. Chung, B. Moon, Parallel Data Processing with MapReduce: A Survey, SIGMOD Record 40 (4) (2012) 11–20.

[11] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, ACM Communication 51 (1) (2008) 107–113.

[12] Apache Hadoop Project (2017).
URL `http://hadoop.apache.org/`

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing, in: Proceedings of USENIX Conference on Networked Systems Design and Implementation, 2012, pp. 1–14.

[14] N. Liu, X. Yang, X. H. Sun, J. Jenkins, R. Ross, YARNsim: Simulating Hadoop YARN, in: Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 637–646.

[15] S. Karpate, A. Joshi, J. Dosani, J. Abraham, Cascket: A Binary Protocol Based C Client-Driver for Apache Cassandra, in: Proceedings of International Conference on Advances in Computing, Communications and Informatics, 2015, pp. 387–393.

[16] J. Xu, Z. Chen, J. Tang, S. Su, T-Storm: Traffic-Aware Online Scheduling in Storm, in: Proceedings of IEEE International Conference on Distributed Computing Systems, 2014, pp. 535–544.

[17] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Job Scheduling for Multi-User MapReduce Clusters, Tech. rep., EECS Department, University of California, Berkeley, USA (2009).
URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.html`

[18] A. Raj, K. Kaur, U. Dutta, V. Sandeep, S. Rao, Enhancement of Hadoop Clusters with Virtualization Using the Capacity Scheduler, in: Proceedings of International Conference on Services in Emerging Markets, 2012, pp. 50–57.

[19] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Information and Software Technology 50 (9-10) (2008) 833–859.

[20] C. Wohlin, Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering, in: Proceedings of International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 38:1–38:10.

[21] D. Cheng, J. Rao, C. Jiang, X. Zhou, Resource and Deadline-Aware Job Scheduling in Dynamic Hadoop Clusters, in: IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 956–965.

[22] Y. Xun, J. Zhang, X. Qin, X. Zhao, FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters, IEEE Transactions on Parallel and Distributed Systems 28 (1) (2017) 101–114.

[23] Y. Guo, J. Rao, D. Cheng, X. Zhou, iShuffle: Improving Hadoop Performance with Shuffle-on-Write, IEEE Transactions on Parallel and Distributed Systems PP (99) (2016) 1–14.

[24] M. W. Convolbo, J. Chou, S. Lu, Y. C. Chung, DRASH: A Data Replication-Aware Scheduler in Geo-Distributed Data Centers, in: IEEE International Conference on Cloud Computing Technology and Science, 2016, pp. 302–309.

[25] Z. Niu, S. Tang, B. He, An Adaptive Efficiency-Fairness Meta-scheduler for Data-Intensive Computing, IEEE Transactions on Services Computing PP (99) (2016) 1–14.

[26] Y. Guo, J. Rao, C. Jiang, X. Zhou, FlexSlot: Moving Hadoop into the Cloud with Flexible Slot Management, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 959–969.

[27] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, Y. Zhang, Failure data analysis of a large-scale heterogeneous server environment, in: International Conference on Dependable Systems and Networks, 2004, pp. 772–781.

[28] M. Soualhia, F. Khomh, S. Tahar, ATLAS: An Adaptive Failure-Aware Scheduler for Hadoop, in: Proceedings of International Performance Computing and Communications Conference, 2015, pp. 1–8.

[29] D. Cheng, P. Lama, C. Jiang, X. Zhou, Towards Energy Efficiency in Heterogeneous Hadoop Clusters by Adaptive Task Assignment, in: IEEE International Conference on Distributed Computing Systems, 2015, pp. 359–368.

[30] Q. Zhang, M. Zhani, Y. Yang, R. Boutaba, B. Wong, PRISM: Fine-Grained Resource-Aware Scheduling for MapReduce, IEEE Transactions on Cloud Computing 3 (2) (2015) 182–194.

[31] T. Jian, M. Xiaoqiao, Z. Li, Coupling Task Progress for MapReduce Resource-Aware Scheduling, in: Proceedings of IEEE INFOCOM, 2013, pp. 1618–1626.

[32] Y. Liang, Y. Wang, M. Fan, C. Zhang, Y. Zhu, Predoop: Preempting Reduce Task for Job Execution Accelerations, in: Big Data Benchmarks, Performance Optimization, and Emerging Hardware, Vol. 8807 of LNCS, Springer, 2014, pp. 167–180.

[33] M. PASTORELLI, D. Carra, M. Dell'Amico, P. Michiardi, HFSP: Bringing Size-Based Scheduling To Hadoop, IEEE Transactions on Cloud Computing PP (99) (2015) 1–14.

[34] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K. Wu, A. balmin, FLEX: A Slot Allocation Scheduling Optimizer for MapReduce Workloads, in: Proceedings of International Conference on Middleware, 2010, pp. 1–20.

[35] T. Shanjiang, L. Bu-Sung, H. Bingsheng, Dynamic Slot Allocation Technique for MapReduce Clusters, in: International Conference on Cluster Computing, 2013, pp. 1–8.

[36] Z. Liu, Q. Zhang, R. Ahmed, R. Boutaba, Y. Liu, Z. Gong, Dynamic Resource Allocation for MapReduce with Partitioning Skew, IEEE Transactions on Computers 13 (9) (2014) 1–14.

[37] W. Jiayin, Y. Yi, M. Ying, S. Bo, M. Ningfang, FRESH: Fair and Efficient Slot Configuration and Scheduling for Hadoop Clusters, in: International Conference on Cloud Computing, 2014, pp. 761–768.

[38] Y. Yao, J. Wang, B. Sheng, C. Tan, N. Mi, Self-Adjusting Slot Configurations for Homogeneous and Heterogeneous Hadoop Clusters, IEEE Transactions on Cloud Computing PP (99) (2015) 1–14.

[39] S. Tang, B. S. Lee, B. He, Dynamic Job Ordering and Slot Configurations for MapReduce Workloads, IEEE Transactions on Services Computing 9 (1) (2016) 4–17.

[40] Y. Mao, H. Zhong, L. Wang, A Fine-Grained and Dynamic MapReduce Task Scheduling Scheme for the Heterogeneous Cloud Environment, in: Proceedings of International Symposium on Distributed Computing and Applications for Business Engineering and Science, 2015, pp. 155–158.

[41] A. Rasooli, D. Down, A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems, in: Proceedings of IEEE Conference on High Performance Computing, Networking Storage and Analysis, 2012, pp. 1284–1291.

[42] Y. Yao, J. Wang, B. Sheng, J. Lin, N. Mi, HaSTE: Hadoop YARN Scheduling Based on Task-Dependency and Resource-Demand, in: Proceedings of IEEE International Conference on Cloud Computing, 2014, pp. 184–191.

[43] A. Rasooli, D. Down, An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems, in: Conference of the Center for Advanced Studies on Collaborative Research, 2011, pp. 30–44.

[44] D. Cheng, J. Rao, C. Jiang, X. Zhou, Resource and Deadline-Aware Job Scheduling in Dynamic Hadoop Clusters, in: IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 956–965.

[45] Z. Wei, S. Rajasekaran, T. Wood, Z. Mingfa, MIMP: Deadline and Interference Aware Scheduling of Hadoop Virtual Machines, in: Proceedings of International Symposium on Cluster, Cloud and Grid Computing, 2014, pp. 394–403.

[46] I. Ullah, C. Jihyeon, R. Yonjoong, Y. Man, Y. Hee, Hadoop Preemptive Deadline Constraint Scheduler, in: Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2014, pp. 201–208.

[47] Y. Bin, D. Xiaoshe, Z. Pengfei, Z. Zhengdong, L. Qiang, W. Zhe, A Delay Scheduling Algorithm Based on History Time in Heterogeneous Environments, in: Proceedings of ChinaGrid Annual Conference, 2013, pp. 86–91.

[48] D. Pletea, F. Pop, V. Cristea, Speculative Genetic Scheduling Method for Hadoop Environments, in: Proceedings of International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2012, pp. 281–286.

[49] M. Khan, Y. Jin, M. Li, Y. Xiang, C. Jiang, Hadoop Performance Modeling for Job Estimation and Resource Provisioning, IEEE Transactions on Parallel and Distributed Systems 27 (2) (2016) 441–454.

[50] J. Jiang, S. Ma, B. Li, B. Li, Symbiosis: Network-aware task scheduling in data-parallel frameworks, in: IEEE International Conference on Computer Communications, 2016, pp. 1–9.

[51] B. Peng, M. Hosseini, Z. Hong, R. Farivar, R. Campbell, R-Storm: Resource-Aware Scheduling in Storm, in: Proceedings of Annual Middleware Conference, 2015, pp. 149–161.

[52] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: A Platform for Fine-grained Resource Sharing in the Data Center, in: Proceedings of Conference on Networked Systems Design and Implementation, 2011, pp. 295–308.

[53] Y. Cairong, Y. Xin, Y. Ze, L. Min, L. Xiaolin, IncMR: Incremental Data Processing Based on MapReduce, in: Proceedings of International Conference on Cloud Computing, 2012, pp. 534–541.

[54] C. Tseng-Yi, W. Hsin-Wen, W. Ming-Feng, C. Ying-Jie, H. Tsan-sheng, S. Wei-Kuan, LaSA: A Locality-Aware Scheduling Algorithm for Hadoop-MapReduce Resource Assignment, in: Proceedings of International Conference on Collaboration Technologies and Systems, 2013, pp. 342–346.

[55] Y.-C. Kao, Y.-S. Chen, Data-locality-aware mapreduce real-time scheduling framework, Journal of Systems and Software 112 (2016) 65 – 77.

[56] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, I. Stoica, Improving MapReduce Performance in Heterogeneous Environments, in: Proceedings of USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 29–42.

[57] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, in: Proceedings of European Conference on Computer Systems, 2010, pp. 265–278.

[58] L. Liying, T. Zhuo, L. Renfa, Y. Liu, New Improvement of the Hadoop Relevant Data Locality Scheduling Algorithm Based on LATE, in: Proceedings of International Conference on Mechatronic Science, Electric Engineering and Computer, 2011, pp. 1419–1422.

[59] Z. Hui, Y. Shuqiang, C. Zhikun, Y. Hong, J. Songchang, An Locality-aware Scheduling based on a Novel Scheduling Model to Improve System Throughput of MapReduce Cluster, in: Proceedings of International Conference on Computer Science and Network Technology, 2012, pp. 111–115.

[60] Z. Guo, G. Fox, M. Zhou, Investigation of Data Locality in MapReduce, in: Proceedings of International Symposium on Cluster, Cloud and Grid Computing, 2012, pp. 419–426.

[61] R. Xue, S. Gao, L. Ao, Z. Guan, BOLAS: Bipartite-Graph Oriented Locality-Aware Scheduling for MapReduce Tasks, in: Proceedings of International Symposium on Parallel and Distributed Computing, 2015, pp. 37–45.

[62] M. Liroz-Gistau, R. Akbarinia, D. Agrawal, P. Valduriez, Fp-hadoop: Efficient processing of skewed mapreduce jobs, Information Systems 60 (2016) 69 – 84.

[63] E. Coppa, I. Finocchi, On Data Skewness, Stragglers, and MapReduce Progress Indicators, in: Proceedings of ACM Symposium on Cloud Computing, 2015, pp. 139–152.

[64] S. Zheng, Y. Liu, T. He, L. Shanshan, X. Liao, SkewControl: Gini Out of the Bottle, in: Proceedings of IEEE International Parallel Distributed Processing Symposium Workshops, 2014, pp. 1572–1580.

[65] S. Sangwon, J. Ingook, W. Kyungchang, K. Inkyo, K. Jin-Soo, M. Seungryoul, HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation Environment, in: Proceedings of International Conference on Cluster Computing and Workshops, 2009, pp. 1–8.

[66] W. Chunguang, W. Qingbo, T. Yusong, W. Wenzhu, W. Quanyuan, Locality Based Data Partitioning in MapReduce, in: Proceedings of International Conference on Computational Science and Engineering, 2013, pp. 1310–1317.

[67] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang, A Throughput Optimal Algorithm for Map Task Scheduling in Mapreduce with Data Locality, ACM SIGMETRICS Performance Evaluation Review 40 (4) (2013) 33–42.

[68] M. Asahara, S. Nakadai, T. Araki, LoadAtomizer: A Locality and I/O Load Aware Task Scheduler for MapReduce, in: Proceedings of IEEE International Conference onCloud Computing Technology and Science, 2012, pp. 317–324.

[69] Z. Xiaohong, Z. Zhiyong, F. Shengzhong, T. Bibo, F. Jianping, Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments, in: Proceedings of International Symposium on Parallel and Distributed Processing with Applications, 2011, pp. 120–126.

[70] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, E. Ayguade, Deadline-Based MapReduce Workload Management, IEEE Transactions on Network and Service Management 10 (2) (2013) 231–244.

[71] M. Hammoud, M. Sakr, Locality-Aware Reduce Task Scheduling for MapReduce, in: Proceedings of International Conference on Cloud Computing Technology and Science, 2011, pp. 570–576.

[72] T. Jian, M. Shicong, M. Xiaoqiao, Z. Li, Improving ReduceTask Data Locality for Sequential MapReduce Jobs, in: Proceding of IEEE INFOCOM, 2013, pp. 1627–1635.

[73] J. Xu, Z. Chen, J. Tang, S. Su, T-Storm: Traffic-Aware Online Scheduling in Storm, in: Proceedings of IEEE International Conference on Distributed Computing Systems, 2014, pp. 535–544.

[74] X. Jiong, Y. Shu, R. Xiaojun, D. Zhiyang, T. Yun, J. Majors, A. Manzanares, Q. Xiao, Improving MapReduce Performance Through Data Placement in Heterogeneous Hadoop clusters, in: Proceedings of International Symposium on Parallel Distributed Processing, 2010, pp. 1–9.

[75] Z. Xiaohong, F. Yuhong, F. Shengzhong, F. Jianping, M. Zhong, An effective Data Locality Aware Task Scheduling Method for MapReduce Framework in Heterogeneous Environments, in: Proceedings of International Conference on Cloud and Service Computing, 2011, pp. 235–242.

[76] B. Sharma, T. Wood, C. Das, HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers, in: Proceedings of International Conference on Distributed Computing Systems, 2013, pp. 102–111.

[77] J. C. Anjos, I. Carrera, W. Kolberg, A. L. Tibola, L. B. Arantes, C. R. Geyer, Mra++: Scheduling and data placement on mapreduce for heterogeneous environments, Future Generation Computer Systems 42 (2015) 22 – 35.

[78] J. Hui, Y. Xi, S. Xian-He, I. Raicu, ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing, in: Proceedings of International Conference on Distributed Computing Systems, 2012, pp. 516–525.

[79] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris, Scarlett: Coping with Skewed Content Popularity in Mapreduce Clusters, in: Proceedings of Conference on Computer Systems, 2011, pp. 287–300.

[80] C. Abad, L. Yi, R. Campbell, DARE: Adaptive Data Replication for Efficient Cluster Scheduling, in: Proceedings of International Conference on Cluster Computing, 2011, pp. 159–168.

[81] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg, Quincy: Fair Scheduling for Distributed Computing Clusters, in: Proceedings of ACM SIGOPS Symposium on Operating Systems Principles, 2009, pp. 261–276.

[82] L. Yin, L. Chuang, R. Fengyuan, G. Yifeng, H-PFSP: Efficient Hybrid Parallel PFSP Protected Scheduling for MapReduce System, in: Proceedings of International Conference on Trust, Security and Privacy in Computing and Communications, 2013, pp. 1099–1106.

[83] Z. Hui, Y. Shuqiang, C. Zhikun, F. Hua, X. Jinghu, K%-Fair scheduling: A Flexible Task Scheduling Strategy for Balancing Fairness and Efficiency in MapReduce Systems, in: Proceedings of International Conference on Computer Science and Network Technology, 2012, pp. 629–633.

[84] N. Phuong, T. Simon, M. Halem, D. Chapman, Q. Le, A Hybrid Scheduling Algorithm for Data Intensive Workloads in a MapReduce Environment, in: Proceedings of International Conference on Utility and Cloud Computing, 2012, pp. 161–167.

[85] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, P. Lin, Natjam: Design and Evaluation of Eviction Policies for Supporting Priorities and Deadlines in Mapreduce Clusters, in: Proceedings of Annual Symposium on Cloud Computing, 2013, pp. 6:1–6:17.

[86] Y. Guo, J. Rao, C. Jiang, X. Zhou, Moving MapReduce into the Cloud with Flexible Slot Management and Speculative Execution, IEEE Transactions on Parallel and Distributed Systems PP (99) (2016) 1–14.

[87] T. Chao, Z. Haojie, H. Yongqiang, Z. Li, A Dynamic MapReduce Scheduler for Heterogeneous Workloads, in: Proceedings of International Conference on Grid and Cooperative Computing, 2009, pp. 218–224.

[88] H. Mao, S. Hu, Z. Zhang, L. Xiao, L. Ruan, A Load-Driven Task Scheduler with Adaptive DSC for MapReduce, in: Proceedings of International Conference on Green Computing and Communications, 2011, pp. 28–33.

[89] T. Fei, Y. Hao, L. Tianrui, Y. Yan, L. Zhao, Scheduling Real-time Workflow on MapReduce-based Cloud, in: Proceedings of International Conference on Innovative Computing Technology, 2013, pp. 117–122.

[90] L. Peng, C. L. Young, W. Chen, B. Bing, C. Junliang, A. Zomaya, Workload Characteristic Oriented Scheduler for MapReduce, in: Proceedings of International Conference on Parallel and Distributed Systems, 2012, pp. 156–163.

[91] D. Cheng, J. Rao, Y. Guo, X. Zhou, Improving MapReduce Performance in Heterogeneous Environments with Adaptive Task Tuning, in: Proceedings of International Middleware Conference, 2014, pp. 97–108.

[92] Z. Tang, M. Liu, A. Ammar, K. Li, K. Li, An Optimized MapReduce Workflow Scheduling Algorithm for Heterogeneous Computing, Journal of Supercomputing 72 (6) (2016) 2059–2079.

[93] S. Li, S. Hu, S. Wang, L. Su, T. Abdelzaher, I. Gupta, R. Pace, WOHA: Deadline-Aware Map-Reduce Workflow Scheduling Framework over Hadoop Clusters, in: Proceedings of International Conference on Distributed Computing Systems, 2014, pp. 93–103.

[94] S. Sidhanta, W. Golab, S. Mukhopadhyay, OptEx: A Deadline-Aware Cost Optimization Model for Spark, in: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2016, pp. 193–202.

[95] K. Ousterhout, P. Wendell, M. Zaharia, I. Stoica, Sparrow: Distributed, Low Latency Scheduling, in: Proceedings of ACM Symposium on Operating Systems Principles, 2013, pp. 69–84.

[96] J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, J. Dittrich, RAFTing MapReduce: Fast Recovery on the RAFT, in: Proceedings of International Conference on Data Engineering, 2011, pp. 589–600.

[97] Z. Yuan, J. Wang, Research of Scheduling Strategy Based on Fault Tolerance in Hadoop Platform, in: Geo-Informatics in Resource Management and Sustainable Ecosystem, Vol. 399 of Communications in Computer and Information Science, Springer, 2013, pp. 509–517.

[98] Z. Xu, D. Xiaoshe, C. Haijun, F. Yuanquan, Z. Huo, A Parameter Dynamic-Tuning Scheduling Algorithm Based on History in Heterogeneous Environments, in: Proceedings of ChinaGrid Annual Conference, 2012, pp. 49–56.

[99] F. Dinu, T. Ng, Understanding the Effects and Implications of Compute Node Related Failures in Hadoop, in: Proceedings of International Symposium on High-Performance Parallel and Distributed Computing, 2012, pp. 187–198.

[100] Z. Hao, C. Haopeng, Adaptive Failure Detection via Heartbeat under Hadoop, in: Proceedings of IEEE Asia-Pacific Services Computing Conference, 2011, pp. 231–238.

[101] C. Gupta, M. Bansal, T.-C. Chuang, R. Sinha, S. Benromdhane, Astro: A Predictive Model for Anomaly Detection and Feedback-based Scheduling on Hadoop, in: Proceedings of International Conference on Big Data, 2014, pp. 854–862.

[102] C. Qi, L. Cheng, X. Zhen, Improving MapReduce Performance Using Smart Speculative Execution Strategy, IEEE Transactions on Computers 63 (4) (2014) 954–967.

[103] T. Shanjiang, L. Bu-Sung, H. Bingsheng, DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters, IEEE Transactions on Cloud Computing 2 (3) (2014) 333–347.

[104] L. Lei, W. Tianyu, H. Chunming, CREST: Towards Fast Speculation of Straggler Tasks in MapReduce, in: Proceedings of International Conference on e-Business Engineering, 2011, pp. 311–316.

[105] C. Quan, Z. Daqiang, G. Minyi, D. Qianni, G. Song, SAMR: A Self-Adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment, in: Proceedings of International Conference on Computer and Information Technology, 2010, pp. 2736–2743.

[106] O. Yildiz, S. Ibrahim, T. A. Phuong, G. Antoniu, Chronos: Failure-aware Scheduling in Shared Hadoop Clusters, in: Proceedings of IEEE International Conference on Big Data, 2015, pp. 313–318.

[107] O. Yildiz, S. Ibrahim, G. Antoniu, Enabling fast failure recovery in shared hadoop clusters: Towards failure-aware scheduling, Future Generation Computer Systems (2016) –doi:http://dx.doi.org/10.1016/j.future.2016.02.015.

[108] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, W. Shi, Energy-Aware Scheduling of MapReduce Jobs for Big Data Applications, IEEE Transactions on Parallel and Distributed Systems 26 (10) (2015) 2720–2733.

[109] Y.-F. Wen, Energy-aware dynamical hosts and tasks assignment for cloud computing, Journal of Systems and Software 115 (C) (2016) 144–156.

[110] P. Paraskevopoulos, A. Gounaris, Optimal Tradeoff between Energy Consumption and Response Time in Large-Scale MapReduce Clusters, in: Proceedings of Panhellenic Conference on Informatics, 2011, pp. 144–148.

[111] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz, Energy Efficiency for Large-scale MapReduce Workloads with Significant Interactive Analysis, in: Proceedings of the ACM European Conference on Computer Systems, 2012, pp. 43–56.

Mbarka Soualhia holds an M.Sc degree in Engineering concentration Information Technology from École de Technologie Supérieure (ÉTS),Canada and a bachelor degree in Computer Sciences from École Nationale Superieure d'Ingenieurs de Tunis (ÉNSIT), Tunisia. She is currently a Ph.D candidate at Concordia University, Canada and she is working as research assistant under the supervision of Prof. Sofiène Tahar and Prof. Foutse Khomh. Her research focuses on designing adaptive software components and software architecture to process intensive data applications in distributed systems and their verification using formal methods such as Theorem Proving and Model Checking.

Foutse Khomh is an assistant professor at the École Polytechnique de Montréal, where he heads the SWAT Lab on software analytics and cloud engineering research. He received a Ph.D in Software Engineering from the University of Montreal in 2010. His research interests include software maintenance and evolution, cloud engineering, service-centric software engineering, empirical software engineering, and software analytic. He has published several papers in international conferences and journals, including ICSM, MSR, WCRE, ICWS, JSS, JSP, and EMSE. He has served on the program committees of several international conferences including ICSM, WCRE, MSR, ICPC, SCAM, and has reviewed for top international journals such as SQJ, EMSE, TSE and TOSEM. He is program co-chair of the Workshops track at WCRE 2013, program chair of the Tool track at SCAM 2013, program chair for Satellite Events at SANER 2015, and program co-chair for SCAM 2015. He is one of the organizers of the RELENG workshop series and guest editor for a special issue on Release Engineering in the IEEE Software magazine.

Sofiène Tahar received the Diploma degree in computer engineering from the University of Darmstadt, Germany, in 1990, and the Ph.D. degree with distinction in computer science from the University of Karlsruhe, Germany, in 1994. Currently, he is a professor and the research chair in formal verification of system-on-chip at the Department of Electrical and Computer Engineering, Concordia University. His research interests are in the areas of formal hardware verification, system-on-chip verification, analog and mixed signal circuits verification, and probabilistic, statistical and reliability analysis of systems. Dr. Tahar, a professional engineer in the Province of Quebec, is the founder and director of the Hardware Verification Group at Concordia University. He is a senior member of ACM and a senior member of IEEE.