The Journal of Supercomputing (2021) 77:5289–5318
https://doi.org/10.1007/s11227-020-03487-5

An architecture for scheduling with the capability of minimum share to heterogeneous Hadoop systems

Abdol Karim Javanmardi[1] · **S. Hadi Yaghoubyan**[1,3] · **Karamollah BagheriFard**[1,3] · **Samad Nejatian**[2,3] · **Hamid Parvin**[4,5,6]

Accepted: 22 October 2020 / Published online: 5 November 2020

Abstract
Job scheduling in Hadoop has been thus far investigated in several studies. However, some challenges including minimum share (min-share), heterogeneous cluster, execution time estimation, and scheduling program size facing Hadoop clusters have received less attention. Accordingly, one of the most important algorithms with regard to min-share is that presented by Facebook Inc., i.e., FAIR scheduler, based on its own needs, in which an equal min-share has been considered for users. In this article, an attempt has been made to make the proposed method superior to existing methods through automation and configuration, performance optimization, fairness and data locality. A high-level architectural model is designed. Then a scheduler is defined on this architectural model. The provided scheduler contains four components. Three components schedule jobs and one component distributes the data for each job among the nodes. The given scheduler will be capable of being executed on heterogeneous Hadoop clusters and running jobs in parallel, in which disparate min-shares can be assigned to each job or user. Moreover, an approach is presented for each problem associated with min-share, cluster heterogeneity, execution time estimation, and scheduler program size. These approaches can be also utilized on its own to improve the performance of other scheduling algorithms. The scheduler presented in this paper showed acceptable performance compared with First-In, First-Out (FIFO), and FAIR schedulers.

Keywords Scheduling · Hadoop · High-level architecture · Minimum share · Heterogeneous clusters

✉ S. Hadi Yaghoubyan
yaghoobian.h@gmail.com

Extended author information available on the last page of the article

1 Introduction

Big data is a field of computer science, addressing information analysis and meta-data extraction methods from data sets. A number of software have been so far developed for processing data, which can be structured, semi-structured, or unstructured [1]. In big data, the data is accompanied by the concepts of velocity, variety, volume, and veracity. Data processing also includes some rows of the records that can show a high statistical power, while data with high complexity can merely lead to an increase in false discovery rate [2]. Data capturing, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy, data source, scheduling methods, etc., are correspondingly among challenges to big data.

Various schedulers have been similarly designed for big data processing software including Hadoop, developed by Doug Cutting as a set of open-resource projects. Different algorithms have been additionally presented for scheduling Hadoop systems. However, these algorithms are facing several challenges, as described briefly below:

1. *Energy efficiency:* Constant growth of information and high volume of generated information have resulted in high-energy consumption in data processing centers. More energy consumption has also led to higher processing costs.
2. *Load balancing:* Balancing load among information processing nodes can yield decreased cost and job execution time.
3. *Mapping scheme:* Creating an efficient scheme and optimizing communication costs between mapping (Map) and reduction (Reduce) steps are rarely seen in the presented algorithms. A suitable scheme can thus increase the efficiency of a scheduler.
4. *Automation and configuration:* Hardware configuration and use of algorithms working better with this configuration can boost the efficiency of a scheduler.
5. *Optimized data shuffling:* Cutting the input and output of a disk at this step and ultimately reducing execution time leads to higher efficiency.
6. *Performance optimization:* Lack of support for overlapping and pipelining at the Map and Reduce steps has given rise to poor performance in Hadoop systems. Therefore, re-using previous results can eliminate this problem to some extent.
7. *Fairness:* Assigning resources to jobs during Map and Reduce steps can provide a good response time for light jobs in a fair manner. In most presented algorithms, Reduce step does not start until Map step is finished.
8. *Data locality:* Smaller distance between nodes, performing processing, and ones, wherein data are stored, will lead to higher efficiency and lower job execution times.
9. *Synchronization:* As Reduce step is subsequent to Map step, performance of Hadoop cluster decreases in environments wherein nodes are heterogeneous due to the presence of a node with low efficiency.

If the timing of each task is determined before execution, a scheduling policy can be designed that can have high performance in most of the above-mentioned cases. To this end, numerous efforts have been thus far made to estimate execution times (see Sect. 2). Some of these attempts have made use of probabilistic methods to estimate execution times. Estimation errors, complex calculations, and high computational overhead have been also among problems facing probabilistic methods. Some other efforts have further estimated execution times by saving the history of previous schedules. Replacing old histories with new ones, use of memory to keep a history of scheduling, as well as high overhead has been some problems with this method. On the other hand, minimum share (min-share) is required in some applications. In Hadoop systems, FAIR scheduler has been designed to create minimal sharing between users, given that, in some recent needs, a min-share is required for the jobs. This article tries to solve the above problems. An architectural model is designed to improve existing challenges. The proposed model uses the full capacity of the clusters and creates a load balance based on the power of each node. It considers fairness in the scheduling. It introduces a new scheduling configuration with a three-layer architecture. Finally, the proposed architectural model achieves the main goals of scheduling and increases performance and locality. A scheduler is also designed. The scheduler ensures a minimum of sharing between users and jobs. The innovations of this article are as follows:

1. An architectural model is defined and developed for scheduling. This model performs scheduling at three levels: user system, scheduler server, and data nodes running tasks.
2. A basic unit is defined and this basic unit is used in scheduling development. The base unit is a measure of the performance of systems in the scheduling process.
3. A scheduler is defined and developed. This scheduler estimates the job execution time in the heterogeneous Hadoop clusters with low overhead and high accuracy.
4. The designed scheduler has the ability to share resources between users or jobs in the heterogeneous Hadoop clusters. This scheduler is able to ensure minimal sharing of resources between users or jobs.
5. Three algorithms have been designed for scheduling in the user's system, scheduling server and data nodes executing tasks. An algorithm is also designed to distribute data among data nodes based on performance.
6. The designed scheduler is evaluated with standard Hadoop algorithms (FIFO and FAIR) in real and simulated Hadoop environment.

This article is organized as follows: Sect. 2 presents the related work. Section 3 describes the Hadoop clusters and its parameters. Section 4 explains the importance of the base unit used in scheduling and the reason for its use. In Sect. 5, the designed scheduling system is defined and explained. This scheduler is designed on the heterogeneous Hadoop clusters, and its design uses the base unit described in Sect. 4. In Section 6, the designed scheduler is evaluated by standard Hadoop algorithms. Section 7 is the summary and finding. Finally, Sect. 8 is the conclusion and future work.

2 Related work

There have been quite a few efforts in the field of scheduling a set of jobs for execution on several systems. In the following, some recent ones are presented.

Genetic algorithm based on random key encryption had been used in [3] for assigning heterogeneous jobs to unrelated parallel batch processing machines. In [4], jobs had been divided into some families and then scheduled for execution on a single batching machine. An enhanced container scheduler (ECSched) had been also proposed in [5] for scheduling simultaneous container requests on heterogeneous clusters with resource constraints. For scheduling, capacities had been accordingly formulated as a minimum cost flow problem (MCFP) and the container requirements had been presented using a diagram-specific data structure (i.e., flow network). In [6], a scheduler had been introduced for executing jobs with deadlines and the data had been released on parallel machines with limited working capacity, employing the branch-and-price solution approach. A fault-tolerant job scheduling had been similarly presented in [7], and the given model had been utilized for multi-hybrid job scheduling.

In the following, different scheduling methods in Hadoop systems are reviewed. For this purpose, first, the most famous scheduling algorithms in Hadoop systems and their features are briefly presented. Then, some efforts made in the field of scheduling in this system are illustrated.

- *FIFO Scheduler:* The default scheduler of Hadoop is FIFO, in which jobs can be chosen for execution according to their arrival time. In this type of scheduling, jobs are usually mapped to a node in the same rack [8–16].

  Advantages: In the simplest type of this scheduler, jobs are executed in the same order as they arrive.

  Disadvantages: This scheduler is not effective in heterogeneous environments. It also damages locality because tasks of different jobs cannot be assigned until the first job schedules all its mappings. Response time and locality of small and big jobs are also different. Moreover, it does not pay much attention to resource assignment balancing between small and big jobs.
- *FAIR Scheduler:* Presented by Facebook Inc., this scheduler assigns an equal share of resources to each job. This is fulfilled by creating a pool consisting of a group of jobs based on user identifiers (IDs). If a pool user sends many jobs, it will be limited by the scheduler [8–14, 16].

  Advantages: Fairness and re-assignment of dynamic resources, quick response to small jobs, as well as ability to fix the number of running jobs for each user and pool can be mentioned as the main positive points concerning this scheduler.

  Disadvantages: Complex settings, not considering weight of each job in each pool, unbalanced performance in each pool, and limited number of running jobs in each pool are some weaknesses cited for this scheduler.
- *Capacity Scheduler:* This scheduler was presented by Yahoo Co. whose goal is to maximize resource utilization and efficiency. In this scheduler, a queue

is used instead of a pool. Each queue is also assigned to an organization. Resources are then assigned to queues. Each organization can only access its own queue. Minimum capacity is further guaranteed for each queue. After running jobs are terminated, resources are assigned to a new job. An organization can correspondingly access any extra capacity, not being used by others. This affordably provides resilience for organizations [8, 11, 12, 16].

Advantages: Among the benefits of this scheduler are increased resource efficiency and throughput, capacity not used in jobs reused in queues, as well as supporting hierarchical, resilient, and operational queues.

Disadvantages: High and extreme complexity, difficulty in choosing suitable queues, as well as uncertainty about stability and fairness for queues are among drawbacks facing this scheduler.

- *Late Scheduler:* The main objective of this scheduler is optimizing performance and decreasing job response times. Small jobs are also answered quickly, but big ones are executed in a slow manner, leading to increased background jobs, higher processor workload, unavailable resources, etc. This scheduler also supports homogeneous clusters by default [8, 11, 12, 14, 16].

  Advantages: Performance and response time optimization as much as possible is one of the major advantages of this scheduler.

  Disadvantages: It does not guarantee reliability and suffers from lack of fairness in assigning resources to jobs.
- *Delay Scheduler:* This scheduler is similar to FAIR scheduler; however, a time delay is considered in it in order to boost locality. If job mapping is not to a local node, it also waits for one D and executes a local job. In the case of unavailability of a local job, it waits as long as a D. If a local job is not still available, non-local job mapping is carried out. Enhancing the value of D also raises the probability of hunger and a small value for D yields decreased locality [12, 15].

  Advantages: Use of simple and low overload in complex computations for solving locality problem is a positive point regarding this scheduler.

  Disadvantages: This scheduler lacks efficiency, is not suited for long jobs, needs manual setting of waiting time, and does not consider locality at Reduce step.
- *Deadline Scheduler:* This scheduler has been designed based on a deadline and increased system use. The deadline is also set by the user. Moreover, it is determined using the execution cost model of the job. Input data dimension, data distribution, and processing section (i.e., Map/Reduce) execution time parameters are utilized for calculating the deadline.

  Advantages: Increased efficiency and much focus on optimizing Hadoop are among the advantages of this scheduler [8, 11, 12].

  Disadvantages: In this scheduler, nodes must be homogeneous and there is no support for limitations specified by users for each job.
- *Resource-Aware Scheduler:* This scheduler has been presented to improve resource utilization. It also works unlike FAIR, Capacity, and FIFO schedulers wherein managers first assign jobs to a queue and then resources are assigned to the jobs in the queue. In this scheduling, various resources like network, storage, central processing unit (CPU), input/output (I/O), and disks are shared in an

effective manner. Scheduling is correspondingly completed through two master and slave nodes. Besides, job tracker operation is carried out in the master node, while the task tracker operation is fulfilled in the slave one. The job tracker also keeps the tasks assigned to each task tracker, state of the tasks, and the queue where the running jobs are stored. The task tracker is responsible for executing jobs with maximum number of available slots. Likewise, the scheduler calculates the total number of slots dynamically [12].

Advantages: Increased performance, improved job management, and high efficiency have been mentioned as the positive points of this scheduler.

Disadvantages: Pre-emptive action or priority is not supported at the Reduce step in this scheduler, and it needs extra capabilities for managing bottlenecks.

- *Matchmaking Scheduler:* The objective of this scheduler is to enhance locality for job mapping. Each node also executes local jobs. Nodes that do not have a local job send the heartbeat signal to the main node and then wait for one heartbeat. After waiting for a heartbeat and being exposed to lack of a local job, they execute a non-local job [12, 15].

  Advantages: One of the main benefits of this scheduler is increased locality and efficiency.

  Disadvantages: This scheduler does not consider rack locality, and it needs configuration parameter that leads to algorithm complexity.

In [17], a history of work carried out on Facebook Inc. and Yahoo Co. had been reviewed and categorized into some classes based on their execution time parameters and system states at the moment of job execution. These classes had been used for estimating execution time of new jobs and assigning jobs to resources. In [14], racks had been used for grouping nodes with regard to CPU power and I/O of each system for assigning any CPU bound or I/O bound job. Burst buffer had been also employed in [18] for managing and scheduling I/O bound jobs. In [11], scheduling had been optimized by considering the number of data replications as a variable for each job. Besides, there had been attempts to augment locality in [15]. To this end, clustering had been used for putting existing nodes into clusters. For clustering, the mean execution time and the free and used memory parameters had been applied. In [19], existing nodes had been classified based on their history using the genetic clustering method. In [20], scheduling service quality had been improved in distributed systems via presenting the Quality-Driven Scheduling for Distributed Machine Learning. The Bayesian algorithm had been also employed in [13] for execution time estimation in order to improve scheduling in Hadoop systems. By implementing famous scheduling methods in Hadoop systems in [21], batch programs had been further tested. In [22], a high-performance architecture had been correspondingly presented for scheduling heterogeneous Hadoop clusters to decrease energy consumption. In [8], a scheduling algorithm with time limits had been introduced for stable calculations in the Hadoop environment. In [10], quick failure recovery had been achieved in Hadoop clusters using failure-aware scheduling. Moreover, a cost-efficient scheduling had been utilized in Hadoop clusters in [23] and an algorithm with the ability to predict job execution times had been presented in [16] for scheduling in Hadoop.

As mentioned above, the proposed scheduling methods have problems and weaknesses. These include the following:

- Some methods execute one task at a time, and they are not able to perform parallel tasks. Due to the large volume of data processing, tasks must be performed in parallel.
- Some methods can only schedule jobs on homogeneous clusters with the same systems. The efficiency of these schedulers on heterogeneous systems is very low.
- Estimating execution time is one of the most important challenges of scheduling. Estimating execution time on heterogeneous systems with different performance is very complex. Some methods are not able to estimate execution time. These methods assume that execution time estimates are available for each job. Another group of schedulers use probabilistic methods to estimate execution time. These methods require high calculations or they are not very accurate. Another category of schedulers using job histories. These methods maintain a large volume of records.
- Some methods have a high workload on scheduling servers. Increasing scheduling performance increases the workload on scheduling servers.
- For some applications, a minimum share for each job or user must be guaranteed. Many schedulers do not guarantee a minimum share. Some existing methods guarantee a minimum share. But these methods give the minimum share to the job or the user, not to both.
- Some methods do not consider fairness. Observing fairness among jobs or users increases efficiency.
- Many methods only deal with jobs and do not pay attention to the distribution of job data between Hadoop clusters. Good data distribution increases efficiency and locality.

To solve the above, an architectural model is presented in this article. This model is designed to:

- This model executes jobs in parallel.
- It can be run on homogeneous and heterogeneous clusters. This model uses a base unit and considers heterogeneous clusters as a homogeneous cluster. This reduces the complexity of runtime estimation and scheduling.
- It defines a basic system. This system is used to estimate the execution time of jobs. Each computational system is given a performance factor. To do this, a category containing one or more specific jobs is selected. Each system must run a batch before being added to the Hadoop cluster. The coefficient of each system is calculated based on the average execution time obtained. Only execution time is maintained based on the base system to store job histories. Therefore, there is no need to maintain the number of jobs in the system and the status of systems within the cluster for job histories. This greatly reduces the space required to store job histories.

- It reduces the workload of scheduling servers by performing part of the scheduling on the user's system.
- It can guarantee a minimum share for each job or user based on system requirements.
- It considers fairness in scheduling.
- Job scheduling is consistent with the distribution of data jobs across data nodes.

The main purpose of designing this architectural model is to increase performance and locality. The architectural model is designed in such a way that any scheduling policy can be implemented with a slight change. The designed architectural model is expressed in Sect. 5.

3 Preliminaries and Hadoop system definition

The main objective of designing Hadoop is to quickly store and process a large amount of information. For this purpose, a storage section called Hadoop Distributed File System (HDFS) and a processing section (namely, Map/Reduce) have been developed. For quick information storage and processing, Hadoop additionally uses a cluster comprised of n computational nodes. In Hadoop, computational nodes are also called data nodes. A cluster with n data nodes is thus displayed as follows:

$$\text{Cluster} = \left\{\text{DN}_1, \ldots, \text{DN}_n\right\}$$

In most Hadoop systems, each CPU core in each data node is recognized as a slot. Therefore, each data node has one storage unit and a set of slots:

$$\text{DN}_j = \left(\text{slts}_j, \text{mem}_j\right)$$

In case s is the number of slots inside data node DN_j, the set of slots pertaining to data node DN_j are represented as follows:

$$\text{slts}_j = \left\{\text{slt}_1^j, \ldots, \text{slt}_s^j\right\}$$

Besides, each slot has an execution rate (ER) and the ERs of the slots in a data node are equal.

mem_j refers to the memory unit of data node DN_j and has two capacity and data retrieval rate (RR) properties. Data RR of a data node is the speed at which the data is read from the storage unit of that data node. If f is the number of files in the Hadoop system, its set of files is represented as follows:

$$\text{Files} = \left\{F_1, \ldots, F_f\right\}$$

Hadoop also breaks files down into big blocks (called slices) and consequently distributes them among the data nodes of the cluster. Each file is then divided into equally sized slices.

In a Hadoop system, the size of data blocks is fixed and pre-defined. Therefore, the size of each slice is equal to that of the data blocks in the Hadoop system (Slc. size = Bsize). The number of slices (value l) for each file is correspondingly obtained as follows:

$$F_i = \left\{ \mathrm{slc}_1^i, \ldots, \mathrm{slc}_l^i \right\}$$

Fsize^i is the size of file F_i and Bsize denotes the size of data blocks in the Hadoop system.

$$l = \frac{\mathrm{Fsize}^i}{\mathrm{Bsize}} \tag{1}$$

Likewise, the set of users who use a Hadoop system is represented as follows:

$$\mathrm{Usrs} = \left\{ U_1, \ldots, U_N \right\}$$

where N represents the number of users. The set of jobs of user i is also called Jobs^i,

$$\mathrm{Jobs}^i = \left\{ j_1^i, \cdots, j_m^i \right\}$$

wherein m is the number of jobs of user i and $j_d{}^i$ shows the dth job of user i.

Data$_d{}^i$ also represents a set of files known as the required data for the execution of dth job of user i. Average ER of job j_d on data node DN_j is further obtained from the reverse of its mean execution time.

$$\mathrm{MER}\left(j_d \text{ on } \mathrm{DN}_j\right) = \frac{1}{\mathrm{ME}\left(j_d \text{ on } \mathrm{DN}_j\right)} \tag{2}$$

Schedulers in Hadoop systems to schedule jobs use parameters like priority, min-share, locality, etc. In this respect, priority is an integer number given to a job by the user or the scheduler and shows the importance ratio of that job. Moreover, the number of slots assigned to each job at any point in time is determined by the scheduler and based on its policy. The minimum number of slots that must be assigned by the system to user i for job d at any given point in time is called the min-share and is represented as minshr$_d{}^i$. Typically, the set of user's jobs in Hadoop systems, which are currently in use, is dynamic. This means that the set of jobs belonging to user i at time t_1 is different from time t_2. For data processing, the Map/Reduce section of the Hadoop system is also utilized. This section performs processing in parallel. In Map/Reduce, each job is also executed during two Map and Reduce steps. Therefore, each job includes sets of Map tasks and Reduce tasks.

$$j_d^i = j_d^i \cdot \mathrm{Maps} \cup j_d^i \cdot \mathrm{Reduces}$$

If the Map tasks set of job d of user i has x tasks, it is represented as follows:

$$j_d^i \cdot \mathrm{Maps} = \left\{ j_d^i \cdot mt_1, \ldots, j_d^i \cdot mt_x \right\}$$

in addition, if the Reduce tasks set of job d of user i includes y tasks, it is characterized as follows:

$$j_d^i \cdot \text{Reduces} = \left\{ j_d^i \cdot rt_1, \dots, j_d^i \cdot rt_y \right\}$$

each Reduce task of job j_d is also executed after Map tasks of job j_d and it uses some of their results.

4 Importance and base unit definition

One of the challenges facing distributed computing systems is heterogeneity of computing systems in clusters, which has caught a lot of attention in scheduling Hadoop systems. Since jobs are executed in parallel and on several systems with different performances in Hadoop systems, calculating execution time of a job becomes a difficult task. In case the scheduling policy permits the execution of various jobs in parallel, the situation becomes much more complicated because the execution time of a job changes as the system status and the number of free slots vary at the time of its execution, such that a different execution time is obtained at each execution of a job. In case of a good estimation for the execution time of the jobs, scheduling jobs becomes more efficient. For this reason, in many scheduling algorithms presented in this line, there have been efforts to estimate the execution time of the jobs. Most proposed methods have been thus unique to their investigated system. To solve this problem, a method is presented in this paper so that the estimation of job execution times becomes independent of the heterogeneity of the systems and their free capacity when the jobs are executed.

Here, a base unit is defined for execution time and the execution time of each job is calculated accordingly. For this reason, the execution power of a system in a time unit is considered as the base unit for execution time. Since slots are the smallest processing units in Hadoop systems, one slot is considered as the base. Now, any slot with a different performance can be measured relative to it. For instance, in case a user's job takes two-time units on the base slot and one-time unit on another slot, it can be stated that the new slot is equivalent to two base slots.

Having access to the number of slots used for a job, performance coefficient of the slots relative to the base slot, as well as execution time of the job on each slot, the execution time of the job can be now calculated relative to the base slot (as the base unit). This execution time based on the base slot will not change by varying the members of the utilized set of slots or their performance coefficient (it is assumed that only the time when each slot is busy is considered). Contrary to this method, converting the execution time of a job based on the base slot into the time it takes for that job to be executed on an arbitrary set of slots is possible by having the performance coefficient of those slots. In the following, the execution time of a job according to the base slot as the base unit is considered and the execution time of a job is estimated accordingly. This base unit is used in the presented scheduling.

5 Proposed Hadoop scheduling system

The high-level architecture of the proposed scheduling system is illustrated in Fig. 1. First, the workflow in the high-level architecture of the proposed system is described, and later, the four components in this architecture are explained.

5.1 Workflow in high-level architecture of proposed system

First, the job scheduler unit on the user's system obtains the estimated execution time of the new job from the user process component. Then, it sends the new job request to the job tracker, which subsequently declares the desire to receive the new job by sending an identification code for the new job. After receiving the identification code, the job scheduler unit on user's system sends the new job along with its estimated time to the job tracker. The job tracker also puts the new job in the job queue and sends the acknowledgment message of the new job to the user. After the acknowledgment message is received by the user's system, a copy of the data needed for executing the new job is sent from the user's system to the HDFS. Next, the HDFS executes the data partitioning process component. Moreover, the HDFS distributes the received files between the nodes based on the results of the data partitioning process component. Task trackers also periodically report the state of the slots to the job tracker. Therefore, the job tracker knows the number of free and busy slots in the nodes (i.e., Map and Reduce slots). With the arrival of each new job or the termination of a job, the job tracker executes the job scheduling process component. By executing this component, each share of each job in the queue from all the slots (namely, Map and Reduce slots) is determined. Once the quota of the jobs in the queue is specified, the job scheduling process component assigns a new slot quota to the jobs in the queue by calling the task scheduling process component. In case the data required for the execution of a task is not available to the slot executing that task, data request is sent to the HDFS and it sends the required data to the slot. Once job execution is done, the job tracker sends the job termination message to the user's system. This message contains the job execution time according to the

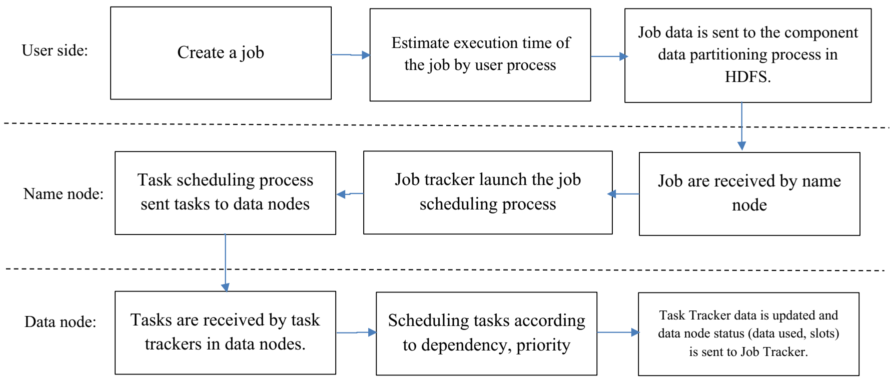
User side:
Create a job
Estimate execution time of the job by user process
Job data is sent to the component data partitioning process in HDFS.
Name node:
Job are received by name node
Job tracker launch the job scheduling process
Task scheduling process sent tasks to data nodes
Data node:
Tasks are received by task trackers in data nodes.
Scheduling tasks according to dependency, priority
Task Tracker data is updated and data node status (data used, slots) is sent to Job Tracker.

Fig. 1 Workflow in high-level architecture of proposed system

base slot. Completed job information is ultimately saved as a new record in the user process component.

5.2 User process

On user's system, the estimated execution time of the job is also calculated before being sent to the job tracker. As well, the job execution time estimate is obtained using the algorithm for finding the nearest neighbors admissible. Algorithm 1 presents the algorithm for finding the nearest neighbors admissible. This algorithm is located in the user process component, estimating the execution time of each job through a data set consisting of the execution time of the finished jobs. To simplify the algorithm, the number of different types of jobs is considered to be fixed and predefined. This algorithm also creates a table for each different type of job and puts the data pertaining to each type in its specific table. Moreover, each table contains two columns with respect to the base slot: file size and execution time. For estimating the execution time of a new job, this algorithm firstly finds a table with the job of the same type as the new job. Then, it selects k samples from the table, which is a fixed number and is specified by the user by default. To increase the accuracy of the execution time estimation, the algorithm does not consider the values with differences higher than the threshold value, as a positive number. A sample can only participate in the job execution time estimation if its maximum size difference with the new job is not more than the difference threshold value. Therefore, k samples with the closest size to that of the new job and the size difference of each one with that of the new job, not more than the difference threshold value, are chosen from the table. In case the number of the selected samples is zero ($k=0$), the algorithm chooses a sample with the closest size to that of the new job (without considering the difference threshold value) and obtains the estimated time value using Eq. 3.

$$\mathrm{EET}_{\mathrm{new}} = \frac{\mathrm{Data}_{\mathrm{new}}}{\mathrm{Data}_i} * \mathrm{ET}_i \tag{3}$$

Data_i is the size of sample i such that sample i is of the same type as the new job and has the closest size to that of the new job among available samples and ET_i refers to the execution time of sample i. Otherwise, the number of the selected samples is more than zero ($k>0$). Therefore, the average execution time of the selected samples is employed for execution time estimation. Equation 4 presents how execution time estimation of the new job is carried out using k samples.

$$\mathrm{EET}_{\mathrm{new\ job}} = \frac{\sum_{i=0}^{k} \frac{\mathrm{Data}_{\mathrm{new\ job}}}{\mathrm{Data}_i} * \mathrm{ET}_i}{k} \tag{4}$$

After the execution of each job, the job tracker sends the execution time of that job with respect to the base slot to the users. Users also save the size of the file utilized by the job and its execution time with regard to the base slot in their tables with maximum size values. In case the table is filled, user's system chooses two records with the least size difference and removes them from the table before a new record is added. Then, it calculates their average execution time using Eq. 4. It should be noted that puts the calculated value as the execution time of the new record and their average size as the size of the new record in the table.

Algorithm 1 Algorithm for finding admissible nearest neighbors

```
TD: The Threshold of Difference.
K: Number of selected samples with the least difference.
JT: Number of job types.
Tables[JT]: An array of tables where each table contains samples of one type of job.
When a new job (say $j_{new}$) arrives do:
Find a table from the Tables[JT] (say $T_j$) for $j_{new}$ so that the type of samples in the table and the
$j_{new}$ are the same.
Select K samples from $T_j$ that has the nearest volume to $j_{new}$ and consider as a set A.
Select a sample from $T_j$ that has the nearest volume to $j_{new}$ and consider as min.
k=K.
For each sample within the set A do
  If |$j_{new}$.size-sample.size|>TD then
      Remove it from A.
      k=k-1.
  End if
End for
If k=0 then
    Estimate the execution time of the new job using Formula 3 and min.
Else
    Estimate the execution time of the new job using Formula 4 and set A.
End if
```

5.3 Data partitioning process

This component distributes the data required by the jobs between the data nodes. Each user's data is also distributed according to its execution rate on each data node and its min-share. As an example, if a data node has a higher ER for job *d*, more of the data of job *d* will be assigned to it. In case the data node memory is full, priority is with the job that has a higher min-share. To this end, if job *d* wants to store its data in a data node and the memory of that data node is full, the data selects a job with the lowest min-share and removes it from the data node. Then, the data of job *d* is stored in the data node. Data node also gives the data pertaining to the removed job to the HDFS, so that it can be re-distributed among the data nodes.

Algorithm 2 depicts data partitioning process. This algorithm firstly calculates the execution time of each job on each one of the data nodes using Eq. 5.

$$\mathrm{ET}\left(j_d^b \text{ on } \mathrm{DN}_i\right) = \frac{\mathrm{EET}_d^b}{\mathrm{Cslts}_i * \mathrm{Nslts}_i} \tag{5}$$

where DN_i is the data node i and $\mathrm{EET}_b{}^d$ shows the execution time, estimate of job j_d calculated by user U_b. Cslts_i and Nslts_i are also the coefficients and the number of slots of data node *i*, respectively. Then, the affinity value of each data node is calculated using Eq. 6.

$$\mathrm{Aff}\left(j_d^b \text{ on } \mathrm{DN}_i\right) = w_t * \frac{1}{\mathrm{ET}\left(j_d^b \text{ on } \mathrm{DN}_i\right)} + w_s * \mathrm{minshr}_d^b \tag{6}$$

wherein w_t is the time weight and w_s represents the min-share weight. If it is assumed that $w_s = 0$ and $w_t = 1$, affinity becomes the execution rate. minshr_d^b is also the min-share of job *d* of user *b*.

In case there are *n* data nodes, the total affinity of the data nodes for j_d^b is obtained from Eq. 7.

$$\mathrm{TAff}\left(j_d^b\right) = \sum_{i=1}^{n} \mathrm{Aff}\left(j_d^b \text{ on } \mathrm{DN}_i\right) \tag{7}$$

The amount of data for each data node is further calculated as the ratio of the affinity value of that data node relative to other data nodes.

$$\mathrm{DN}_i \cdot \mathrm{Data}\left(j_d^b\right) = \mathrm{Data}_d^b * \frac{\mathrm{Aff}\left(j_d^b \text{ on } \mathrm{DN}_i\right)}{\mathrm{TAff}\left(j_d^b\right)} \tag{8}$$

wherein Data_d^b refers to the amount of data required for executing job *d* of user *b*. Besides, $\mathrm{DN}_i \cdot \mathrm{Data}(j_d^b)$ specifies the amount of data from job *d* that must be stored on data node *i*. Note that the min-share of a user on data nodes is identical. Therefore, if the memory of data nodes does not get full, the affinity used in Eq. 8 distributes the data of a job among data nodes only according to the execution rate. If the memory of data nodes gets full, the affinity of the jobs is compared with other ones and the min-share affects selecting the priority of the jobs in that case (refer to Algorithm 2).

Algorithm 2 Data partitioning process

```
$j_d^b$: $j_d$ of the user b.
When data from the $j_d^b$ arrives do:
Calculate the execution time of the $j_d^b$ on all data nodes using Formula 5.
Calculate the affinity value of the $j_d^b$ on all data nodes using Formula 6.
Calculates the sum of affinities of the $j_d^b$ on all data nodes using Formula 7.
For each data node do
  Calculate The amount of data that $j_d^b$ must store in the data node using Formula 8. (say $d_{new}$)
  If the data node has free space to store $d_{new}$ then
      Save the $d_{new}$ in the data node.
  Else
      Find the data in the data node whose job has the lowest affinity. (say $d_{selected}$)
      Send $d_{selected}$ to HDFS for distribution between data nodes.
      Replace $d_{new}$ with the $d_{selected}$.
  End if
End for
```

5.4 Job scheduling process

The job tracker uses the execution and waiting queues for scheduling jobs. The execution queue contains executing jobs, and the waiting queue is comprised of the jobs waiting to be executed. The maximum size of the execution queue is also equal to the total number of available slots in the system. Considering each job is received by the job tracker and the execution queue is full or the sum of the min-share of the jobs in the execution queue with the min-share of the new job is more than the system's total number of slots with respect to the base slot, the job is placed in the waiting queue. Otherwise, the job is added to the end of the execution queue. Each job is also removed from the execution queue after being executed. The slots are subsequently assigned to all the jobs in the execution queue. Upon assigning the slots to each job at the Map and Reduce steps, the job's min-share parameter and the ratio of the execution rate of the job relative to the ER of other jobs in the execution queue at that time are used. To simplify scheduling, Map and Reduce slots are simultaneously assigned to the jobs in the execution queue. Therefore, for each job, Eqs. 14 and 16 are run for Map and Reduce slots. In the following, the required calculations are explained and the queue here means the execution queue.

Algorithm 3 presents the job-scheduling process. With the arrival of each new job or its termination, this algorithm is executed. The job also enters the waiting state if a new job is received and the number of jobs in the queue is more than the total number of slots. Waiting state might further occur if sum of the min-share of the jobs in the execution queue with the mini-share of the new job is more than the total number of the slots in the system with respect to the base slot. The total number of the slots with regard to the base slot is calculated using Eq. 9.

$$\mathrm{TNslts} = \sum_{i=1}^{n} \mathrm{Cslts}_i * \mathrm{Nslts}_i \tag{9}$$

n is the number of data nodes, Cslts_i refers to the coefficient of the slots of data node i, and Nslts_i represents the number of slots of data node i. Otherwise, the job is added to the end of the execution queue. Then, the ER of the jobs in the queue is calculated using Eq. 10.

$$\mathrm{ER}_d = \frac{1}{\mathrm{EET}_d} \tag{10}$$

EET_d refers to the estimated execution time of job d calculated on user's system employing the user process components. If there are p jobs in the queue at moment t, the total ER of the jobs in the queue at moment t is calculated using Eq. 11.

$$\mathrm{TER}^t = \sum_{i=1}^{p} \frac{1}{\mathrm{EET}_i} \tag{11}$$

For each job in the queue, the job tracker calculates the number of slots that must be assigned to that job. To do this, it firstly calculates the sum of the min-shares of the jobs in the queue using Eq. 12.

$$\mathrm{Tminshr}^t = \sum_{i=1}^{p} \mathrm{minshr}_i \tag{12}$$

$\mathrm{min\_shr}_i$ shows the min-share of job i, and $\mathrm{Tminshr}^t$ is the sum of the min-share of the jobs in the queue at moment t. Then, it subtracts the sum of the min-share of the jobs from the total number of the slots with respect to the base slot, so that the number of the remaining slots from all the slots is determined (Eq. 13).

$$\mathrm{RNslts} = \mathrm{TNslts} - \mathrm{Tminshr}^t \tag{13}$$

The number of the slots of each job is calculated using Eq. 14.

$$\mathrm{ANslts}_d^t = \mathrm{RNslts} * \frac{\mathrm{ER}_d}{\mathrm{TER}^t} + \mathrm{minshr}_d \tag{14}$$

$\mathrm{ANslts}_d{}^t$ is the number of slots that must be assigned to job d at moment t.

Upon the termination of one job and its removal from the queue of jobs, the job tracker divides the released slots between the jobs present in the queue if there are no other jobs in the queue. The number of the free slots of the data nodes at time t is calculated using Eq. 15.

$$\mathrm{TFNslts}^t = \sum_{i=1}^{n} \mathrm{Cslts}_i * \mathrm{FNslts}_i^t \tag{15}$$

$\mathrm{FNslts}_i{}^t$ represents the number of free slots in system i at time t. Then, the share of each job in the queue from the free slots is determined and added to the number of the allocated slots of each job.

$$\mathrm{ANslts}_d^{t+1} = \mathrm{TFNslts}^t * \frac{\mathrm{ER}_d}{\mathrm{TER}^t} + \mathrm{ANslts}_d^t \tag{16}$$

$\mathrm{ANslts}_d{}^{t+1}$ is the number of slots that must be assigned to job d at time $t+1$. $\mathrm{TFNslts}^t$ refers to the total number of the free slots of the data nodes at time t. ER_d shows the ER of job d.

Algorithm 3 Job scheduling process

```
p: Number of jobs in the execution queue.
TNS: Total number of slots in the system.
TNslts: Total number of slots in terms of the base slot in the system.
$minshr_i$: Minimum share of job i.
When a new job arrives or finish executing a job do:
If a new job (say $j_{new}$) arrives then
    If p+1> TNS OR $Tminshr^t+minshr_{new}>$ TNslts then
      Add $j_{new}$ to waiting queue.
      Return.
    Else
      Add $j_{new}$ to execution queue.
      p=p+1.
      Calculate the execution rate of the jobs in the execution queue using Formula 10.
      Calculate the total execution rate of the jobs in the execution queue using Formula 11.
      Calculate the sum of the minimum shares of the jobs in the execution queue using Formula 12.
      Calculate the number of remaining slots after assigning the minimum number of slots
      required in the execution queue using Formula 13.
      For each job in the execution queue do
          Calculate the number of slots that must be assigned to the job using Formula 14.
      End for
      Call Task scheduling process algorithm to assigns slots to the jobs in the execution queue.
    End if
Else
    // finish a job (say $j_{completed}$) in the execution queue.
    Remove $j_{completed}$ from the execution queue.
    p=p-1.
    If waiting queue isn't NULL then
      Remove a job from the waiting queue and set it as $j_{new}$.
      Call job scheduling process algorithm.
      Return.
    Else
      Calculate The number of free slots of data nodes using Formula 15.
      For each job in the execution queue do
          Calculate the number of slots that must be assigned to the job using Formula 16.
      End for
      Call Task scheduling process algorithm to assigns slots to the jobs in the execution queue.
    End if
End if
```

Algorithm 4 Task scheduling process

```
newslts_i: The number of new slots to be assigned to job i.
lastslts_i: The number of last slots assigned to job i.
Fslt: set of free slots. It is computed by Algorithm 3.
Aslts_i: set of slots allocated to the job i.
When this algorithm is called do:
Select the jobs in the execution queue with a minimum share greater than zero and set as g.
Select the jobs in the execution queue with a minimum share equal to zero and set as e.
For each job in the g and then for each job in the e do
    $dif = newslts_i - lastslts_i$.
    Remove unexecuted tasks from the queue of slots in the set Aslts_i and set as B.
    If $dif > 0$ then
        Select a number of free slots that have the closest coefficient to dif and add them to the set
        of previously assigned slots (Aslts_i).
        Assign unexecuted tasks from set B to slots in the set Aslts_i.
    Else
        Select a number of slots in set  Aslts_i that have the closest coefficient to dif and remove
them
        from this set.
        Assign unexecuted tasks from set B to slots in the set Aslts_i.
    End if
End for
```

5.5 Task scheduling process

After specifying the number of the slots that must be assigned to each job in the queue, the job tracker calls the task-scheduling process component, which assigns the slots to the jobs in the queue according to the values calculated by the job tracker. Algorithm 4 presents the task scheduling process.

First, the ID of the slots that must be assigned to each job is determined. This component gives the slots to the jobs with the least min-share. Then, the remaining slots are divided between the jobs without any min-share. This guarantees the min-share being assigned to the jobs. To select the slots of the job among the available slots, the slot with the minimum difference with the job's share is mainly selected. Afterward, the remaining slots are divided between the jobs without any min-share. This guarantees the min-share being assigned to the jobs. For selecting the slots of a job among available slots, the slot with the minimum difference with the job's share is selected first. Next, the ID of the slot is saved for the job and the slot is removed from the set of available ones. The slot coefficient value is then subtracted from the job's share value. This action continues until the job's share becomes zero, meaning that if job i needs three base slots, it selects the slots from the available ones whose coefficients add up to three or the sum of their coefficients is the closest number bigger than 3. After specifying the IDs of the slots of each job, the tasks are assigned to the jobs. For the new job, its tasks are sent to the queue of the slots specified for it. The tasks of each job are also divided into its slots according to the coefficient of each slot relative to others, i.e., the faster the slot, the more the tasks assigned to it.

For each job being executed, the tasks in the queues of new and previous slots are accordingly edited. For instance, imagine at time *t1*, job scheduling gives slots 1, 2 and 3 to job *x*. At time *t2*, job *y* enters the queue and job scheduling is re-executed and specifies the share of each job. Upon the execution of job scheduling, slot 2 must be taken from job *x* and then given to job *y*. Therefore, task scheduling removes the unexecuted tasks of job *x* from the queue of slot 2 and divides them between slots 1 and 3 in accordance with their coefficients and puts the tasks of job *y* in the queue of slot 2. When the number of the slots of a job is increased by the job tracker, task scheduling can gather the unexecuted tasks from the queues of previous slots and distribute them between the old and new slots after reprocessing.

6 Experimental results

In this section, the results of implementing the proposed method and its comparison with conventional methods in simulated and real Hadoop environments are presented. First, the properties of the simulation environment and its results are expressed. Then, the results and the evaluation circumstances of the scheduling system in the real environment are delineated. FIFO and FAIR scheduler have been used to evaluate the proposed method. FIFO and FAIR are standard schedulers in the Hadoop. FIFO is default scheduler in the Hadoop. This scheduler has been implemented in the Hadoop. It is used in some applications. It is a standard to evaluating proposed schedulers in the Hadoop [8, 10, 14, 17]. The FAIR Scheduler was developed by Facebook as they needed to share the clusters between multiple users. Facebook uses this scheduler. Due to the similarity of our proposed method with the FAIR Scheduler, this Scheduler has been used for better evaluation.

6.1 Simulation environment

For evaluation, the Map/Reduce Simulator MRSIM [24] has been thus far used to simulate a Hadoop cluster, which is based on discrete-event simulation and models the Hadoop environment very well. In this paper, this simulator was extended for measuring the proposed method. One component for users, one component for scheduling based on the method and one component for sending jobs to the scheduler were also added to the MRSIM architecture. The JobTracker component in the MRSIM was also altered.

For the simulation environment, a cluster including five heterogeneous data nodes is defined. The properties of the data nodes are presented in Table 1. The bandwidth between network components is 1Gbps. For generating workload, the Hadoop Map/Reduce trace created in [21] is used. Table 2 outlines the results of this trace over six months from May to October 2009. The number of users has been considered five such that each job is sequentially assigned to a user, i.e., job 1 is labeled for user 1, job 2 is labeled for user 2, etc. Accordingly, 100 jobs are sent to the system and the types of jobs are the ones in Table 2. The count of jobs from each type based on its ratio to all other jobs is calculated in Table 2. As well, Table 3 shows the min-share

Table 1 Experimental data nodes

Data nodes	Slots number	Slots Exec rate	Mem capacity	Mem retrieve rate
DN1	2	400	4 TB	9 Gbps
DN2	2	5	400 KB	40 Kbps
DN3	2	400	4 TB	9 Gbps
DN4	4	400	4 TB	9 Gbps
DN5	2	5	400 KB	40 Kbps

Table 2 Job categories in Facebook trace. Map time and Reduce time are in task seconds, e.g., 2 tasks of 10 s each are 20 task seconds [21]

Job categories	Duration (s)	Job	Input	Shuffle	Output	Map time	Reduce time
Facebook trace							
Small jobs	32	126	21 KB	0	871 KB	20	0
Fast data load	1260	25	381 kB	0	1.9 GB	6079	0
Slow data load	6600	3	10 kB	0	4.2 GB	26321	0
Large data load	4200	10	405 kB	0	447 GB	66657	0
Huge data load	18300	3	446 kB	0	1.1 TB	125662	0
Fast aggregate	900	10	230 GB	8.8 GB	491 MB	104338	66760
Aggregate and expand	1800	6	1.9 TB	502 MB	2.6 GB	348942	76736
Expand and aggregate	5100	2	418 GB	2.5 TB	45 GB	1076089	974395
Data transform	2100	14	255 GB	788 GB	1.6 GB	384562	338050
Data summary	3300	1	7.6 TB	51 GB	104 kB	4843452	853911

Table 3 Minimum share of job types

Job types	min_share (num slots)
J1	1
J2	0
J3	0
J4	2
J5	2
J6	0
J7	1
J8	1
J9	0
J10	2

of each job in Table 2. For all scheduling algorithms (FIFO, FAIR and the proposed algorithm), the size of data blocks in Hadoop systems is set to 128 megabytes, which is its default value in Hadoop 1.2.1. For the number of data replications, the default system value of 3 is used. For the user process component, the size of the table on user's system is set to 10 jobs and the difference threshold is 500 MB and $k=4$. For w_t and w_s, as the time weight and the min-share weight, values of 0.3 and 0.7, are set, respectively.

6.2 Simulation results

The results used in the charts are calculated from the average of ten outputs resulting from simulations. At each run, the proposed method is compared with the FIFO and FAIR algorithms (the version introduced in [9]). Figure 2 presents the average job execution times for the algorithms based on the number of the completed jobs. The results show that the proposed method has better average execution time compared with other algorithms. This superiority is because the proposed method, unlike the other two algorithms, can select the slots with higher performance from available slots for the execution of the jobs. Another reason is how the data are stored. The proposed method can distribute the data among data nodes based on their performance and lead to a decrease in the time the data required by a job are not available in the data node. Therefore, the execution time of the job reduces. As presented in Fig. 2, the proposed algorithm and the FAIR algorithm can perform better than the FIFO algorithm because of the parallel execution of the jobs. While there are small jobs in the system, the difference between the average execution time of the FAIR algorithm and the FIFO algorithm is high. With the arrival of big jobs and waiting for small jobs due to the long-term allocation of the slots by big jobs, the execution time of short times increases. Thus, the average execution time of the FAIR algorithm rises and approaches the FIFO one. However, due to parallel executions, it can perform better. Because of assigning the slots to the jobs based on their ER, the proposed method can solve the problem of the FAIR algorithm. For this reason, small jobs are answered quickly and this can significantly affect the average execution time.

Figure 3 presents the average execution time of the schedulers. Considering that the FIFO algorithm considers a simple method in scheduling jobs, it produces a lower execution time. The time difference between the proposed method and the

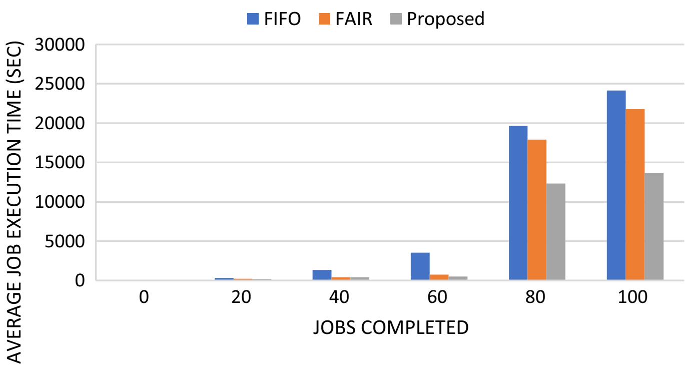

Fig. 2 Average job execution time for FIFO, FAIR *n* proposed method

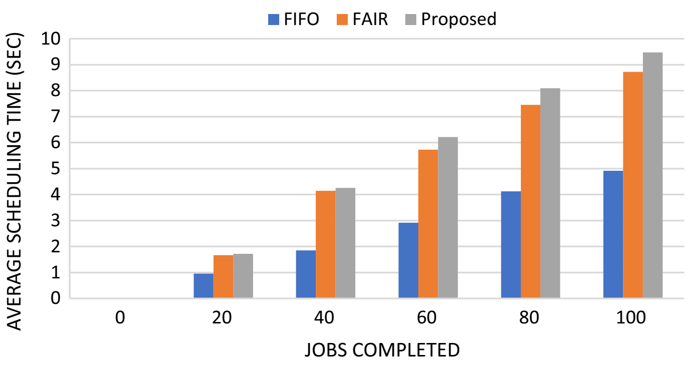

Fig. 3 Average scheduling time for FIFO, FAIR *n* proposed method

FAIR algorithm is not very significant, and the FAIR algorithm presents a lower time. The proposed method calculates the execution time of each job and sends the job to the name node. In the name node, jobs are scheduled and slots are assigned to them. The time shown in Fig. 3 includes all of these times. Given that one part of the scheduling operation in the proposed method takes place on user's system and here the time of that operation is calculated, it can be stated that the proposed method presents a lower time for the job tracker compared with the FAIR algorithm. In Fig. 4, only the schedule performed in the name node is calculated. Due to the fact that FAIR and FIFO algorithms do all the scheduling in the name node, their time has not changed, but the proposed method has been reduced due to doing part of the scheduling in the name node. Figure 5 shows the locality of the algorithms. The locality of the proposed method can improve the FIFO algorithm by 3.3% on average. As the proposed method divides the data among data nodes based on their performance and uses this policy to assign jobs to data nodes, it presents a higher locality.

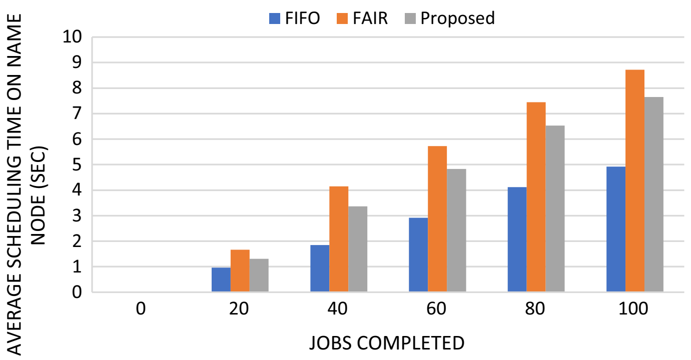

Fig. 4 Average scheduling time on name node for FIFO, FAIR *n* proposed method

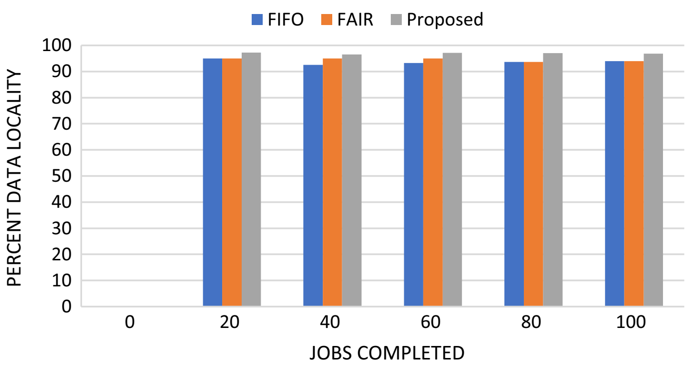

Fig. 5 Percent data locality for FIFO, FAIR *n* proposed method

6.3 Real Hadoop system environment

In this section, the performance of the proposed method is evaluated via some experiments on a Hadoop cluster and use of a real workload. Considering that there are various limitations for evaluating a real Hadoop cluster, the results of this section must be considered as a basis for verifying the practicality of the solution in real systems. For evaluation, a small local Hadoop cluster with six nodes has been thus far used. This cluster can be converted into a medium-sized one; however, six nodes are used in this paper due to some limitations. The local cluster includes a master node and five slave ones. The components are also connected to each other through 1 Gbps Ethernet. The information regarding the systems in the cluster is presented in Table 4. The Micro-Benchmark workload has been so far used in the Hadoop system, comprised of WordCount, Sort and TeraSort. These workloads are also widely applied in Hadoop research, and they are compatible with the Map/Reduce system. The data of the Micro-Benchmark workload has been similarly generated using the

Table 4 Hadoop evaluation environment

Node type	Hardware configuration	Hadoop configuration
Master node	Intel core i7-6700HQ CPU @ 2.60 GHz * 8, 16 GB RAM, 1000 GB Disk space	8 map slots and 4 reduce slots
Slave node 1	Intel core i5-2430 M CPU @ 2.40 GHz * 4, 4 GB RAM, 500 GB Disk space	4map slots and 2 reduce slots
Slave node 2	Intel core i3-4160 CPU @ 3.60 GHz * 4, 8 GB RAM, 1000 GB Disk space	4 map slots and 2 reduce slots
Slave node 3	Intel core i7-6700HQ CPU @ 2.60 GHz * 8, 16 GB RAM, 1000 GB Disk space	8 map slots and 4 reduce slots
Slave node 4	Intel core i5-2430 M CPU @ 2.40 GHz * 4, 4 GB RAM, 500 GB Disk space	4 map slots and 2 reduce slots
Slave node 5	Intel core i3-4160 CPU @ 3.60 GHz * 4, 8 GB RAM, 1000 GB Disk space	4 map slots and 2 reduce slots

RandomTextWriter in the Hadoop system version 1.2.1. The size of the generated files varies from 10 k to 4G. For the size of data blocks and the number of data replications, default Hadoop values, i.e., 128 MB and 3, have been also utilized. For evaluation, the Java Development Kit (JDK) version 8 has been used. The output of the diagrams is the average of ten experiments. In scheduling systems presented in Hadoop, min-share is given to either the job or the user. In this section, min-share is given to the users (unlike the previous section). Assigning min-share to the user or the job is also based on the policy of the organization or the company providing the system (e.g., Facebook Inc. gives the equal min-share to users). Table 5 presents the min-share for the users. For the parameters defined in Sect. 5.2 (that is, user process component), the size of the tables on user's systems is set to 10 jobs and the difference threshold to 500 MB and $k = 4$. In the beginning, a sample job is selected from all three types of jobs. Then, the selected jobs are run on the system, considered as the base one. It is better to reflect on the weakest system as the base unit, so that fraction coefficients are not less than one. In the systems in Table 4, the weakest system in terms of performance (namely, slave node 1) is taken into account as the base. To find the weakest system, a sample job can be executed on them and their execution times can be compared. This will determine the slot coefficient of each system and the weakest one. The execution time of the jobs with respect to the base slot is calculated using Eq. 17.

$$\mathrm{ET} = \mathrm{ET}_i * \mathrm{Nslts}_i * \mathrm{Cslts}_i \tag{17}$$

wherein ET is the execution time with respect to the base slot and Nslts_i shows the number of slots available in system i. As well, ET_i is the execution time of the job on system i. Cslts_i also represents the coefficient of the slots of system i with respect to the base slot (since system i is the base and coefficient of its slots is one). The size of these files and their calculated execution time is placed in user's tables as the first record. Values $w_t = 0.3$ and $w_s = 0.7$ are also set.

6.4 Real environment results

Figure 6 illustrates the average job execution time for the algorithms in a real Hadoop system based on the job types. According to Fig. 6, the proposed method has on average executed WordCount, Sort and TeraSort jobs by 45.58%, 45.39% and 41.33% faster than the FIFO algorithm and the FAIR algorithm has been equal to 5.22%, 7.64% and 6.16% faster than the FIFO algorithm, respectively. Figure 7 shows the average

Table 5 Minimum share of users

Users	Min-share
User 1	1
User 2	0
User 3	2
User 4	1
User 5	1

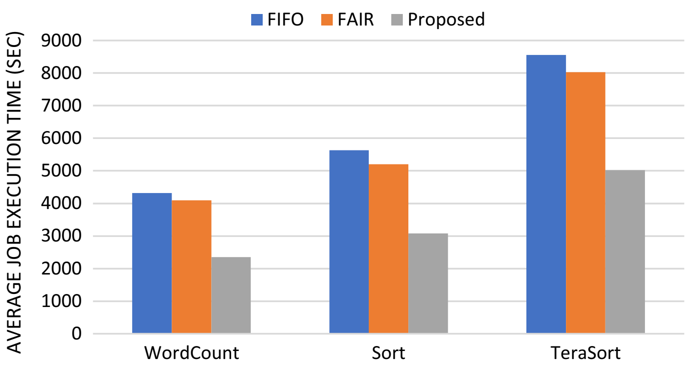

Fig. 6 Average job execution time for Micro-Benchmarks

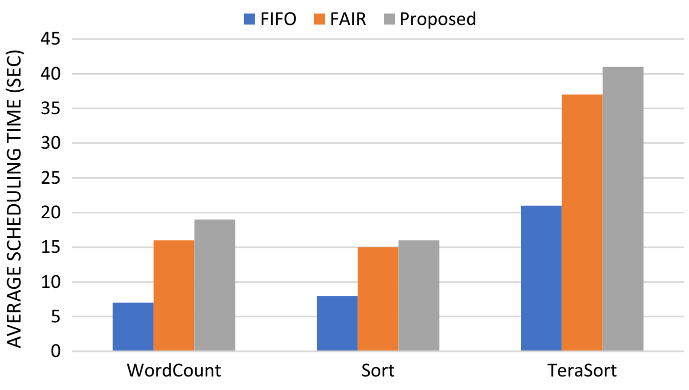

Fig. 7 Average scheduling time for Micro-Benchmarks

execution time of the schedulers. The complexity of the proposed method and the FAIR algorithm has similarly led to a rise in their scheduling execution time compared with the FIFO algorithm. Figure 7 shows the execution time of the proposed method in the user's system and the name node. Figure 8 shows it in the name node.

Figure 9 depicts the locality of the algorithms. The proposed method has improved the locality of WordCount, Sort and TeraSort jobs by 10%, 10% and 13.5% on average compared with the FAIR algorithm and 19%, 21% and 24% compared with the FIFO algorithm, respectively. Taking the performance of the data nodes into consideration while assigning the jobs and distributing the data of the jobs based on the performance of the nodes in the proposed method can also lead to a difference in its results compared with those of the FIFO and FAIR algorithms.

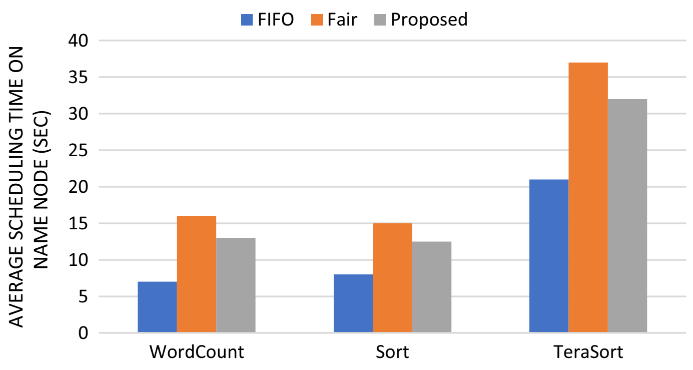

Fig. 8 Average scheduling time on name node for Micro-Benchmarks

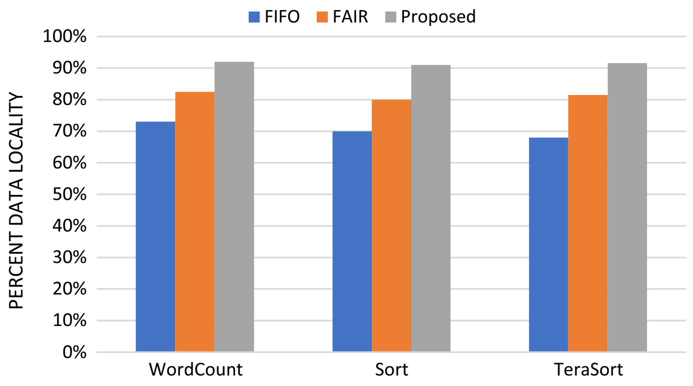

Fig. 9 Percent data locality for Micro-Benchmarks

7 Summary and findings

In this article, first, the most famous scheduling algorithms in Hadoop systems and their features are briefly described. Then, a scheduler was proposed to improve FAIR scheduler. The proposed scheduler was compared with the FIFO and FAIR scheduler in real and simulated environments. The results show that the proposed scheduler works well. Table 6 shows a comparison of various Hadoop schedulers and proposed scheduler. To prepare this table, the results obtained in [12, 25, 26] and [27] and the results obtained in this article have been used. According to the table, the proposed scheduler allocates resources dynamically and considers job priority. Due to the use of the base unit, it shows high performance in heterogeneous environments. There is a fair distribution of resources between the user or jobs. Due to part of the schedule in the user system, the name node overload has been reduced.

Table 6 Comparison of various Hadoop schedulers and proposed scheduler in big data

Scheduler	FIFO	Fair	Capacity	LATE	Delay	Matchmaking	Deadline constraints	Resource aware	Proposed
Allocation	Static	Static	Static	Static	Static	Static	Dynamic	Dynamic	Dynamic
Resources sharing	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Priority in job queue	No	Yes	By default no	Yes	Yes	Yes	Yes	Yes	Yes
Environment homogeneous/heterogeneous	Homo	Homo	Homo	Both	Homo	Homo	Both	Both	Both
Fairness/fair sharing	No	Yes, for users	Yes, for jobs	Yes	Less fairness than fair scheduler	No	Yes	Yes	Yes, for users and jobs
Name node overload	Low	High	High	High	High	Medium	High	High	Medium
Performance	Low	Medium	High	Medium	Medium	Medium	High	Medium	High
Locality	Yes	Yes, for small jobs	Yes	Yes	Improved compared to fair scheduler	Yes	Yes, for small jobs	Improved compared to fair scheduler	Yes
Mode	Non-preemptive	Preemptive	Non-preemptive	Preemptive	Preemptive	Non-preemptive	Preemptive	Preemptive	Preemptive
Taxonomy	Non-adaptive	Adaptive	Adaptive	Adaptive	Adaptive	Adaptive	Adaptive	Adaptive	Adaptive

8 Conclusion

A major challenge in scheduling algorithms in Hadoop systems is heterogeneous clusters. This challenge is also observed in all systems with distributed computing. To deal with this challenge, various methods have been thus far presented. Another challenge of scheduling algorithm is the estimation of the execution time of the jobs. Here, by considering the base unit, the complexity of estimating the execution time for heterogeneous clusters can be decreased. Using this method, existing scheduling algorithms can be further extended. Reflecting on the base slot for the scheduling algorithm is thus better than the memory size or CPU power, since the base unit shows the performance of the systems instead of their apparent specifications. Here, considering the needs of Map/Reduce, the base unit is only defined for system hardware. This unit can be developed to cover software.

In the method presented here, a portion of the scheduling operation is done on user's system. Running some scheduling components on user's system can accordingly reduce the workload of name nodes. Developing scheduling algorithms that move some calculations to user's system can be also studied in the future.

As the costs of scheduling algorithms in Hadoop systems have received little attention from researchers, many companies nowadays do many of their computational processing over the Internet and on user's systems (in exchange for paying money), so there is a need for cost-based scheduling algorithms.

As a part of this research work in the future, it is suggested to do a comprehensive survey on scheduling algorithms in different distributed systems and compare them with the scheduling algorithms presented for Hadoop systems. Moreover, there will be attempts to use the findings in this paper.

Acknowledgements This paper has been extracted from a PhD thesis entitled "Improvement of scheduling in Hadoop clusters" with the supervision of Dr Yaghoubyan, Dr BagheriFard, Dr Nejatian, Dr Parvin.

References

1. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mob Netw Appl 19(2):171–209
2. Breur T (2016) Statistical power analysis and the contemporary "crisis" in social sciences. J Mark Anal 4(2–3):61–65
3. Zhou S, Xie J, Du N, Pang Y (2018) A random-keys genetic algorithm for scheduling unrelated parallel batch processing machines with different capacities and arbitrary job sizes. Appl Math Comput 334:254–268
4. Cheng B, Cai J, Yang S, Hu X (2014) Algorithms for scheduling incompatible job families on single batching machine with limited capacity. Comput Ind Eng 75:116–120
5. Hu Y, Zhou H, de Laat C, Zhao Z (2020) Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Future Gener Comput Syst 102:562–573
6. Osorio-Valenzuela L, Pereira J, Quezada F, Vásquez ÓC (2019) Minimizing the number of machines with limited workload capacity for scheduling jobs with interval constraints. Appl Math Model 74:512–527
7. Moon Y-H, Youn C-H (2015) Multihybrid job scheduling for fault-tolerant distributed computing in policy-constrained resource networks. Comput Netw 82:81–95

8. Varga M, Petrescu-Nita A, Pop F (2018) Deadline scheduling algorithm for sustainable computing in Hadoop environment. Comput Secur 76:354–366
9. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I (2009) Job scheduling for multi-user mapreduce clusters. In: EECS Department, University of California, Berkeley
10. Yildiz O, Ibrahim S, Antoniu G (2017) Enabling fast failure recovery in shared Hadoop clusters: towards failure-aware scheduling. Future Gener Comput Syst 74:208–219
11. Suresh S, Gopalan NP (2014) An optimal task selection scheme for Hadoop scheduling. IERI Proced 10:70–75
12. Usama M, Liu M, Chen M (2017) Job schedulers for Big data processing in Hadoop environment: testing real-life schedulers using benchmark programs. Digit Commun Netw 3(4):260–273
13. Guoa Y, Wu L, Yuc W, Wud B, Wange X (2015) The improved job scheduling algorithm of Hadoop platform.pdf. arXiv e-prints
14. Gupta S, Fritz C, Price B, Hoover R, Dekleer J, Witteveen C (2013) Throughputscheduler: learning to schedule on heterogeneous Hadoop clusters. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC'13), pp 159–165.
15. Naik NS, Negi A, BR TB, Anitha R, (2019) A data locality based scheduler to enhance MapReduce performance in heterogeneous environments. Future Gener Comput Syst 90:423–434
16. Xie J, Meng F, Wang H, Pan H, Cheng J, Qin X (2013) Research on scheduling scheme for Hadoop clusters. Proced Comput Sci 18:2468–2471
17. Rasooli A, Down DG (2014) COSHH: a classification and optimization based scheduler for heterogeneous Hadoop systems. Future Gener Comput Syst 36:1–15
18. Liang W, Chen Y, Liu J, An H (2019) CARS: a contention-aware scheduler for efficient resource management of HPC storage systems. Parallel Comput 87:25–34
19. Brahmwar M, Kumar M, Sikka G (2016) Tolhit: a scheduling algorithm for Hadoop cluster. Proced Comput Sci 89:203–208
20. Zhang H, Stafman L, Or A, Freedman MJ (2018) SLAQ: quality-driven scheduling for distributed machine learning. arXiv e-prints. arXiv:1802.04819
21. Chen Y, Ganapathi A, Griffith R, Katz R (2011) The case for evaluating mapreduce performance using workload suites. In: Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems
22. Malik M, Neshatpour K, Rafatirad S, Joshi RV, Mohsenin T, Ghasemzadeh H, Homayoun H (2019) Big vs little core for energy-efficient Hadoop computing. J Parallel Distrib Comput 129:110–124
23. Islam MT, Srirama SN, Karunasekera S, Buyya R (2020) Cost-efficient dynamic scheduling of big data applications in apache spark on cloud. J Syst Softw 162:110515
24. Hammoud S, Li M, Liu Y, Alham NK, Liu Z (2010) MRSim: a discrete event based mapreduce simulator. In: Proceedings of the Seventh International Conference on Fuzzy Systems and Knowledge Discovery. IEEE. pp 2993–2997.
25. Hv A, Sebastian S (2017) Comparative study of job schedulers in Hadoop environment. Int J Adv Res Comput Sci 8(3).
26. Bahel E, Trudeau C (2019) Stability and fairness in the job scheduling problem. Games Econ Behav 117:1–14
27. Hamad F (2018) An overview of Hadoop scheduler algorithms. Mod Appl Sci 12:69

Affiliations

Abdol Karim Javanmardi[1] · S. Hadi Yaghoubyan[1,3] · Karamollah BagheriFard[1,3] · Samad Nejatian[2,3] · Hamid Parvin[4,5,6]

Abdol Karim Javanmardi
ab_karim_j@yahoo.com

Karamollah BagheriFard
k.bagheri@iauyasooj.ac.ir

Samad Nejatian
samad.nej.2007@gmail.com

Hamid Parvin
parvin@iust.ac.ir

[1] Department of Computer Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran

[2] Department of Electrical Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran

[3] Young Researchers and Elite Club, Yasooj Branch, Islamic Azad University, Yasooj, Iran

[4] Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam

[5] Faculty of Information Technology, Duy Tan University, Da Nang 550000, Vietnam

[6] Department of Computer Engineering, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran