Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

On the optimization of Hadoop MapReduce default job scheduling through dynamic job prioritization

Narges Peyravi*[1] and Ali Moeini†[2]

[1]Department of Computer Engineering and Information Technology, Faculty of Engineering, University of Qom.

[2]Department of Algorithms and Computation, School of Engineering Science, College of Engineering, University of Tehran.

ABSTRACT

One of the most popular frameworks for big data processing is Apache Hadoop MapReduce. The default Hadoop scheduler uses queue system. However, it does not consider any specific priority for the jobs required for MapReduce programming model. In this paper, a new dynamic score is developed to improve the performance of the default Hadoop MapReduce scheduler. This dynamic priority score is computed based on effective factors such as job runtime estimation, input data size, waiting time, and length or bustle of the waiting queue. The implementation of the proposed scheduling method, based on this dynamic score, not only improves CPU and memory performance, but also reduced waiting time and average turnaround time by approximately 45% and 40% respectively, compared to the default Hadoop scheduler.

Keyword: Hadoop MapReduce, Job scheduling, prioritization, dynamic priority score.

AMS subject Classification: 05C78.

ARTICLE INFO

Article history:
Received 12 February 2020
Received in revised form 22 October 2020
Accepted 11 November 2020
Available online 30 December 2020
Research paper

*narges.peyravi@gmail.com

†Corresponding author: A. Moeini E-mail: moeini@ut.ac.ir

1 Introduction

Nowadays, various new systems such as wireless sensor networks, remote health monitoring, smart home, IOT (The Internet of Things), social networks, and stock markets, generate growing data known as the big data. Management, storage, retrieval, processing, and analysis of this data type requires special infrastructure systems and algorithms, due to the volume, velocity, variety, and veracity. One of the well-known open-source frameworks used in storing and processing big data is Apache Hadoop [11]. Hadoop is scalable and reliable framework for big data storage and process. It divides the big input data into fixed-size pieces which is known as splits. These splits are stored and processed on cluster of machines. In Hadoop 2, the default size of each split is 128MB. In order to manage error and fault tolerance, each split is copied three times and transferred to different machines.
Distributed Hadoop file system -HDFS- uses MapReduce programming system which in turn exploits parallelism in cluster computing [2, 11, 17, 22]. For job scheduling Hadoop uses queue system. Considering priority in this queue is a challenging issue, in order to determine which job takes possession of the resources. Since proper prioritized scheduling has significant impact on system performance, such as CPU and memory consumption, load balancing, shorter response time, shorter waiting time, and fairness, this paper proposes a prioritizing jobs module for the default scheduler Hadoop. In this module, some effective parameters like runtime, input data size, entry time, waiting time, and the length or bustle of the waiting queue are considered to compute a score for each job. Based on this score the priority of each job is determined. A higher score yields a higher priority.
The remaining of this paper is as follows: Section 2 describes the statement of the problem; Section 3 presents an overview of earlier researches; Section 4 describes the proposed method. Section 5 states the evaluation and finally, the conclusion is discussed in Section 6.

2 Statement of the problem

Scheduling is a policy in which a system deals with job selection and system resource assignment and allocation. Due to processing and resource storage limitation, applications that work on massive datasets will often wait to run. It is the labor of the scheduler to allocate resources to applications according to some defined policy [22]. By default, Hadoop has three types of job scheduling policies: FIFO, Fair, and Capacity [2, 11, 17, 22]. In FIFO (First In First Out) scheduling model, executable jobs are in the waiting queue, based on entry time. As soon as the system resources are released, they execute in order of priority. Although FIFO has been considered a simple algorithm, it fails to meet the fairness among jobs and it does not consider any priority more than the order in which the job is entered. So, short jobs are sacrificed for longer jobs; so, runtime and waiting time will be long and unacceptable. However, FIFO has been deprecated and is no longer directly used.

In Fair scheduling model, all jobs or applications get an equal share of resources during their course of running. In this model, one or more pools are defined in order that the existing resources are distributed equitably among pools so that several jobs can be run simultaneously. It can run small jobs first without starving the big job. It is simple to configure. However, when more jobs enter, each job's runtime and turnaround time is increased tremendously.
In Capacity scheduling model, multiple hierarchical queuing techniques are used. Each queue is assigned to a user, group, or organization. In other word, the system resource usage percentage is predefined by the administrator in each system. By default, FIFO techniques are utilized in each queue to prioritize jobs. However, there is no priority sequence and none of the jobs have any priority over the other, then, there are FIFO challenges in each queue.
The aim of this paper is to propose a new method to prioritize the jobs and improve the performance of the default scheduler (Fair and Capacity) for Hadoop. The main advantage of this method is to reduce average waiting and turnaround time for the jobs and reduce CPU and Memory consumption. To do this, we need some effective factors like runtime, input data size, waiting time, the length or the bustle of the waiting queue to determine a score for each job. The highest score shows the most prominent priority.

3 Review of Related Work

The matter of scheduling, is a challenging problem that has been tackled by researchers from different aspects. Some researchers try to manage and optimize the resources; others try to shorten the waiting and turnaround times.
A multi-queue scheduling approach of heterogeneous jobs in a hybrid cloud environment is proposed by Li Chunlin et al. [5]. They applied genetic algorithm, and neural networks for Map and Reduce tasks to predict the runtime of a job. High priority tasks with minimum finishing time are allocated to the resources.
Chen He [9] suggested a real-time scheduling g algorithm based on QoS in heterogeneous Hadoop MapReduce clusters. His aim is to energy minimization, data locality, and QoS control. He developed a matchmaking scheduling algorithm for improving the data locality of MapReduce applications. The author achieved a higher cluster utilization without a deadline missing.
Fei Teng et al. [20] used the Paused Rate Monotonic algorithm for real-time tasks prioritized scheduling on Hadoop MapReduce. This prioritized algorithm schedules jobs according to their time constraints, and the pauses between the Map and Reduce stages.
HAT- history-based auto-tuning- MapReduce in heterogeneous environments is proposed by Quan Chen et al. [3, 4]. This scheduler works on historical information saved on every node and it divided slow nodes into Map and Reduce tasks. Once a Map or Reduce task finishes on a node, proper parameters are updated according to the new values. During the run of a MapReduce application, HAT computes the progress scores of all the running jobs periodically.

Chun Lin et al. [14] introduced four scheduling-policy combinations derived from the default schedulers and proposed three different queue-structures: Fair-DRF, Fair-FIFO, and Capacity-FIFO. The experiments show that employing the merged-queue scenario is the best choice because it enables almost all combinations to gain high workload completion rates and shorten workload turnaround time.
Size-based scheduling for Hadoop, schedules tasks by shortening completion times [13, 16]. It also implements an aging policy, where the cost of a job in the queue becomes gradually decremented as it waits for resources. The technique is called the Shortest Remaining Virtual Time-SRVT. At the benefit of virtual error elimination and starvation in the queue, SRVT results in a slight increase in the average throughout time.
Leveraging size patterns scheduler, calculates the number of slots and resources of each based on the history of the job completion time [24]. The completion time will be predicted in regard to the job size. The Job Tracker continually sorts users instead of jobs, and allows the most efficient users to receive the most slots, without starving other users. If new users enter jobs to the cluster, the similar job profiles is assigned to these new users based on some similarity criteria among users.
Wenhong Tian et al. [21] designed a job scheduler that aims to minimize the makespan of a set of MapReduce jobs. The authors combine the features of both Johnson's classical algorithm and MapReduce to minimize the runtime for both online and offline jobs. They considered the Offline and Online HScheduler with the best-known constant ratio for decreasing the runtime.
Abaker Targio Hashem et al. [7] investigated the field of scheduling in big data platforms, which was a comparison among Hadoop, Mesos, and Corona frameworks. The proposed scheduling algorithms have covered the strategies, resources, workload, optimization approaches, requirements, and speculative execution.
Y. Yao et al. [23] proposed two new schedulers named HaSTE and HaSTE-A, for Hadoop YARN systems. Their schema can improve the usage of resources and reduce the runtime of MapReduce jobs based on each task's fitness and urgency. HaSTE dynamically schedules jobs for running when resources become available. By further considering each job's alignment, HaSTE-A addresses the long issue caused by iterative jobs.
Zhou et al. [25] proposed a fine-grained method to improve the allocation granularity of resource. This method estimates resource information and divides jobs into execution stages according to the actual requirements. Then, job resource requirements are matched with the available server resources. They considered two aspects of allocation granularity: duration and quantity.
Praveen M. Dhulavvagol et al. [6] discussed different scheduling techniques and their performance on multimode clusters. They considered some parameters for performance evaluation: CPU time, physical memory, and virtual memory. Their results showed that a capacity-based scheduling algorithm is more efficient than FIFO and FAIR in terms of CPU cycles, physical and virtual memory utilization.
Akhtar et al. [1] have presented a comparative study of job scheduling algorithms which could be used in various big data-based image processing application. They have also proposed a tipping point scheduling algorithm to optimize the workflow for job execution

on multiple nodes. Their algorithm considered job execution time, CPU usage, and workflow optimization. Another purpose of their work is to determine the essentialities of job scheduling and resource allocation.
Hashem, I. A. T. et al. [8] proposed multi-objective MapReduce job scheduling based on the reduction of the completion of time and cost. Their scheduling algorithm considered the earliest finishing time to resource allocation. They compered experimental results with other well-known schedulers, such as FIFO and Fair in different scenarios.
In this paper, the prioritization of jobs is done dynamically by considering effective parameters, which have not been addressed in previous researches. The goal is to reduce waiting and turnaround time, resulting in optimal resource consumption.

4 The proposed method - dynamic job prioritization

In Hadoop platform, several jobs which may be shown as a vector J can enter into the cluster at the same or different times $J = \{j_1, j_2, j_3, \cdots, j_n\}$. Each job j_i is divided into some tasks, the size of each task depends on the size of the splits in Hadoop. In Hadoop 2, the default value size of each split is 128 MB, so each job j_i is a set of several tasks or splits: $j_i = \{t_1, t_2, t_3, \cdots, t_m\}$. Tasks are performed on a cluster of machines like $M = \{m_1, m_2, m_3, \cdots, m_t\}$ and each machine can run them simultaneously [2, 11, 17, 22]. The default Hadoop's scheduling algorithm does not consider any precedence over the selection of jobs. Here, we present a new method to assign dynamic prioritized score to jobs which are ready to run. To calculate this score for job i shown as $P_t(i)$, many parameters which affect the priority are considered. These parameters are: waiting time for job i shown as $T_w(i)$, runtime for job i shown as $T_r(i)$, the number of jobs in the waiting queue or the bustle of the waiting queue shown as l. When the queue is crowded and the length of it is increased, the priority of the earlier jobs should be increased. It means that the score of job i is proportional to $(l-i)k_1$, where k_1 is a constant coefficient between zero and one that diminishes the impact of $(l-i)$.
$T_r(i)$ is the runtime of job i. Considering the fact that big jobs should not be starved and small jobs should not be sacrificed for big jobs, we can conclude that the score of job i is proportional to the ratio of the sum of its runtime and waiting time to its runtime divided by a constant coefficient k_2 which is the number of splits of each i. The discussion above can be summarized in Equation (1) as follows:

$$p_t(i) = \big((l-i) * k_1\big) + \frac{T_r(i) + T_w(i)}{T_r(i) * k_2} \quad \text{or} \quad p_t(i) = \big((l-i) * k_1\big) + \frac{1}{k_2} * \left(1 + \frac{T_w(i)}{T_r(i)}\right) \quad (1)$$

In Equation (1), the waiting time for each job i at time t can be calculated from Equation (2).

$$T_w(i) = T_c(i) - T_e(i) \quad (2)$$

In Equation (2), $T_w(i)$ is the waiting time of job i at time t, $T_c(i)$ is the system's current time, and $T_e(i)$ is the time which job i enters the system.
The algorithm (1) shows the pseudo-code to calculate the score of jobs.

Algorithm (1)- pseudo-code of the proposed job scheduling policy

Input: Job set J //J is a set of jobs
Output: scheduled jobs
1. **For each** application do
2. $L \longleftarrow 0$; // L is the length of queue
3. **For each** job do
4. $id \longleftarrow$ Job's id; //each Job has a unique id
5. $D_{input} \longleftarrow$ *the size of input data*;
6. $N_m \longleftarrow D_{input}/D_{split}$; // D_{split} is the default value of each split (128MB)
7. **Call** EstimatingRunTime // EstimatingRunTime is a function for estimating runtime of a Job
8. **Return** (T_r) ;
9. $J_i \longleftarrow (id, D_{input}, N_m, T_r)$;
10. **ADD** J_i to queue
11. $T_e(J_i) \longleftarrow Time()$; // $T_e(J_i)$ is the time that a Job enters in a queue
12. $L \longleftarrow L + 1$;
13. **For each** job in queue do
14. $T_c \longleftarrow Time()$; // T_c is the current time
15. $T_w(J_i) \longleftarrow T_c - T_e(J_i)$; // $T_w(Ji)$ is waiting time of job i
16. $k_1 \longleftarrow 0.3$; // k_1 is Constant coefficient ($k_1 = 0.3$ is considered experimentally tested)
17. $k_2 = N_m$; // k_2 is The coefficient is proportional to the number of splits
18. $P_t(J_i) \longleftarrow \big[(L-i) * k_1\big] + \big[(T_r(J_i) + T_w(J_i))/(T_r(J_i) * k2)\big]$; // $P_t(J_i)$ is priority score of job i at time t
19. **Then**
20. **Send for Running** $(Max(P_t(J_i))$; // Run a Job with Maximum value of P_t

In Algorithm (1), each job has a unique *id*. In line 5, the size of the input data is computed and is placed into D_{input}. In line 6, the number of splits are obtained by dividing D_{input} to D_{split}. The value of D_{split} is 128MB by default. In line 7, the *EstimatingRunTime* function is called to return the estimate runtime of a job as T_r. This function works according to the job processing anatomy in Hadoop MapReduce. Two cases are considered: when a job runs for the first time or a job has been previously run. In the first case, based on the Hadoop execution pipeline, each phase formulates and runtime is calculated. In the second case, by referring to the profile or the history of the job presented in the database, and by using a weighting mechanism the runtime is estimated. For more explanations on the runtime estimation of a job in Hadoop MapReduce refer to our other paper [18].

J_i includes id, D_{input}, N_m, and T_r of each job that enters the queue for execution. When a job is added to the queue, the length of the queue (L) is increased. The time that a job enters a queue is assigned to $T_e(J_i)$.
T_c shows the current time. For each job in the queue, waiting time ($T_w(J_i)$) is obtained from the difference between T_c and $T_e(J_i)$. k_1 is a constant coefficient that is obtained through several experiments. The results of those experiments revealed that the value should be 0.3. Also, k_2 is a coefficient that is proportional to the number of splits.
Based on Equation (1), the priority score of each job ($P_t(J_i)$) is determined.

5 Evaluation

For clarity, the evaluation is done in two parts. First, we justify the proposed method theoretically with hypothetical values, then test our method in a Hadoop cluster environment and compare it with the default state.

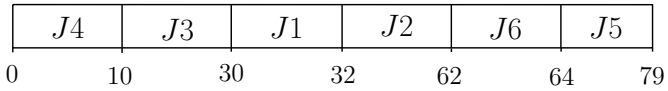
5.1 Theoretical evaluation of the proposed scheduling method

Since both default Hadoop scheduling algorithms (Fair, Capacity) use the FIFO method to enforce prioritization in their queues, first, we compare the proposed algorithm with FIFO. Assuming that six jobs with their respective information presented in Table 1 enter a Hadoop cluster where their arrival times and run times are displayed in seconds in columns 2 and 3, respectively.

Table 1: jobs information

Jobs	Arrival Time(S)	Run Time(s)
J_1	6	2
J_2	10	30
J_3	3	20
J_4	0	10
J_5	12	15
J_6	11	2

In FIFO method, the Gantt chart will be:



Therefore, the average waiting time and the turnaround time in FIFO method can be obtained from the formula below respectively:

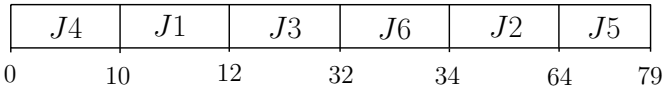
$$Avg_{wait} = \frac{24 + 22 + 7 + 0 + 52 + 51}{6} = 26$$

$$Avg_{turnaround} = \frac{26 + 52 + 27 + 10 + 67 + 53}{6} = 39.16$$

In the proposed method, the priority of each job is determined dynamically based on Equation (1).
At zero time, the only available work is $J4$, which runs and engages the system resources. At time 10; $J3$, $J1$, and $J2$ have competition together. The priority of each one may be computed as follows: (For simplicity in this example, we consider $k_1 = 1$)

$$J1 : 1 + \frac{2+4}{2} = 4 \qquad J2 : 0 + \frac{30+0}{450} = 0.066 \qquad J3 : 2 + \frac{20+7}{200} = 2.135$$

The highest score belongs to $J1$, so it has the highest priority. Similarly, the priority of other jobs is determined according to the following Gantt chart:



The average waiting time and the average turnaround time in the proposed method are:

$$Avg_{wait} = \frac{4 + 24 + 9 + 0 + 52 + 21}{6} = 18.33$$
$$Avg_{turnaround} = \frac{6 + 54 + 29 + 10 + 67 + 23}{6} = 31.5$$

Similar to the example above, the other five-set are theoretically tested. The results are shown in Figures 1 and 2. According to Fig. 1 and Fig. 2, the proposed method is better than FIFO both in average waiting time and turnaround time.

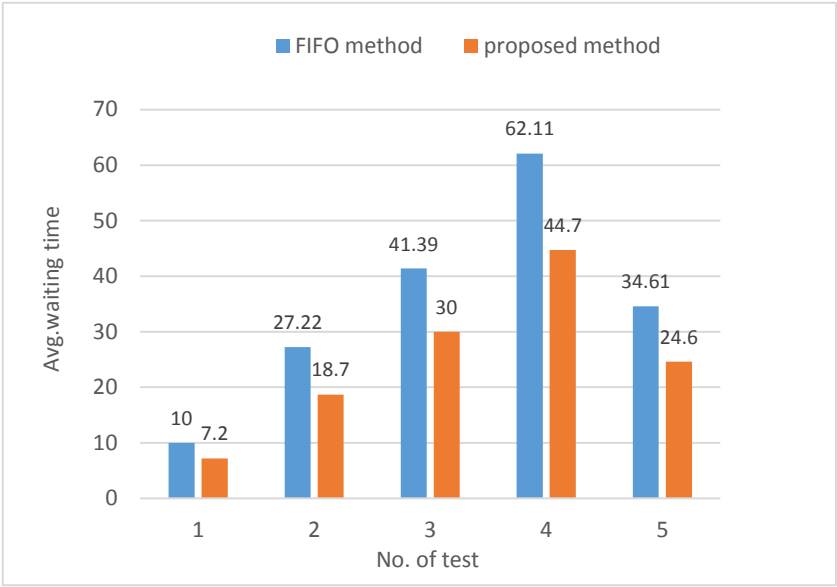


Figure 1: average waiting time

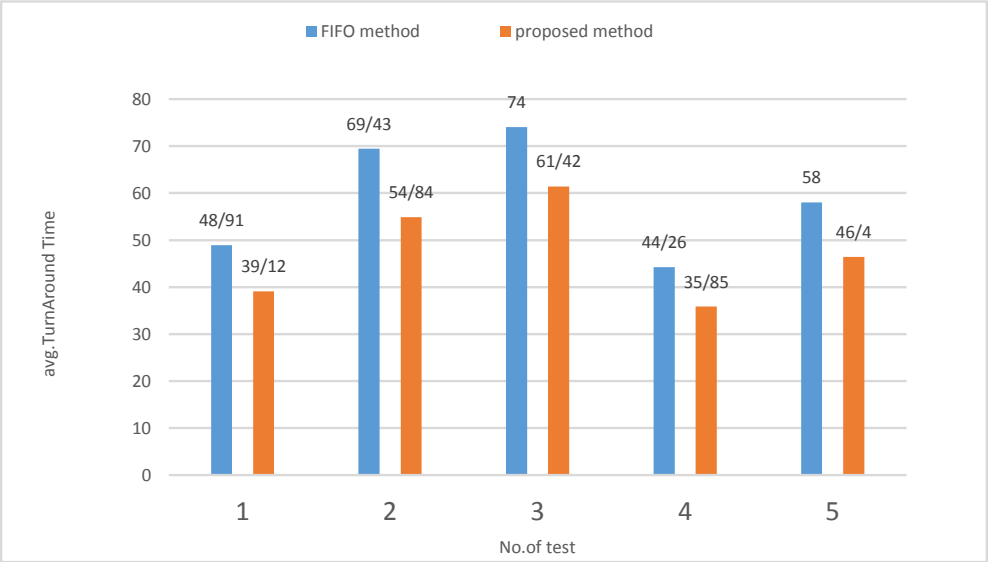


Figure 2: average turnaround time

5.2 Practical evaluation of the proposed scheduling method

After investigating the proposed method theoretically, for a more precise evaluation, we tested our method in our lab with the specifications mentioned below in Table 2.

Table 2: Test environment specifications

Motherboard	Giga p85
CPU	Intel ®core i3-3240 2*2 cores 3.40 GHz
RAM	4 GB
HDD	500 GB
OS	Ubuntu 17.04
Band Width	100 Mbps
Hadoop	2.9.1
JDK	1.8.1

In our lab, there are 13 systems as a cluster of Hadoop with the specifications in Table 2. Out of the thirteen systems, one system is Master, and twelve systems are considered as Slaves. Hadoop 2.9.1 is installed on the systems. Four conventional Hadoop benchmarks [10] were used to produce jobs. TeraGen initially produces input data sizes of 1G, 2G,

3G, 5G, 10G, and 20GB. WordCount is a memory-intensive job for counting the words within a text. TeraSort is a CPU-intensive job for sorting the input file. Inverted Index is a CPU limited job for looking up the set of documents contain a given word or a term. Since the purpose of this study is to improve the performance of the Hadoop default scheduler (Fair and Capacity) the tests have been done with and without the proposed method by using default scheduling algorithms in terms of the average waiting and turnaround times.

By varying the number of jobs that entered the cluster at the same or different times, four sets of tests have been performed. Each set of the test are scheduled with four methods: Fair, Capacity, new Fair, and new Capacity. In the new_Fair (new_Capacity) method, first, we ran our proposed method then we utilized the Fair (Capacity) method. Fig. 3 shows the comparison of the average waiting time, and Fig. 4 shows the comparison of the average turnaround time in WordCount application. Fig. 5 and 6 show the comparison for TeraSort application with average waiting time and turnaround time respectively. Fig. 7 and Fig. 8 show the average waiting time and turnaround time for Inverted index application.

Figure 3: Average waiting time for WordCount

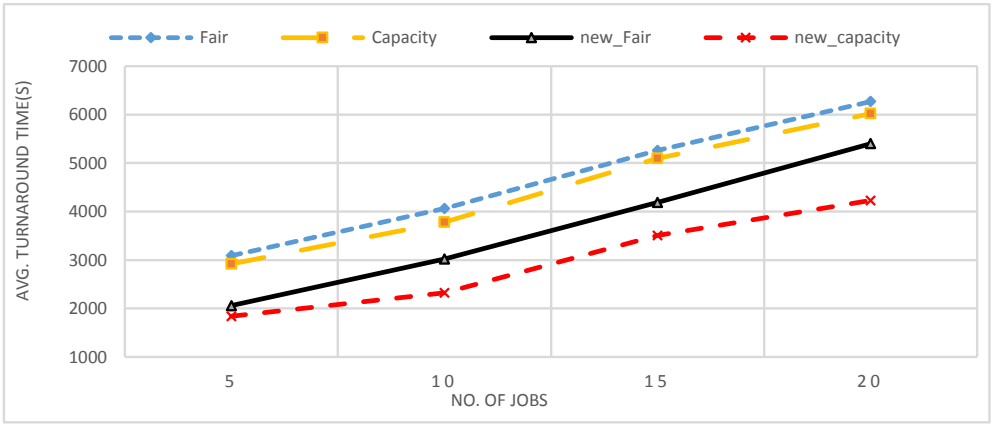


Figure 4: Average turnaround time for WordCount

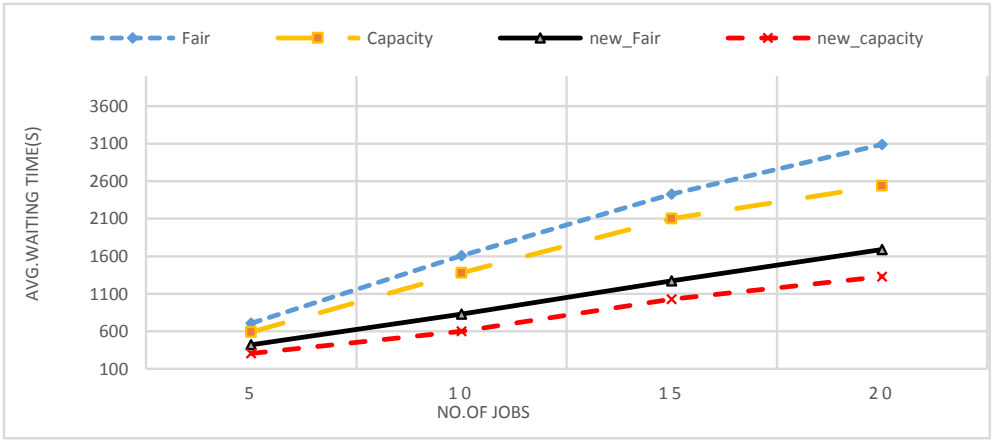


Figure 5: Average waiting time for TeraSort

Figure 6: Average turnaround time for TeraSort

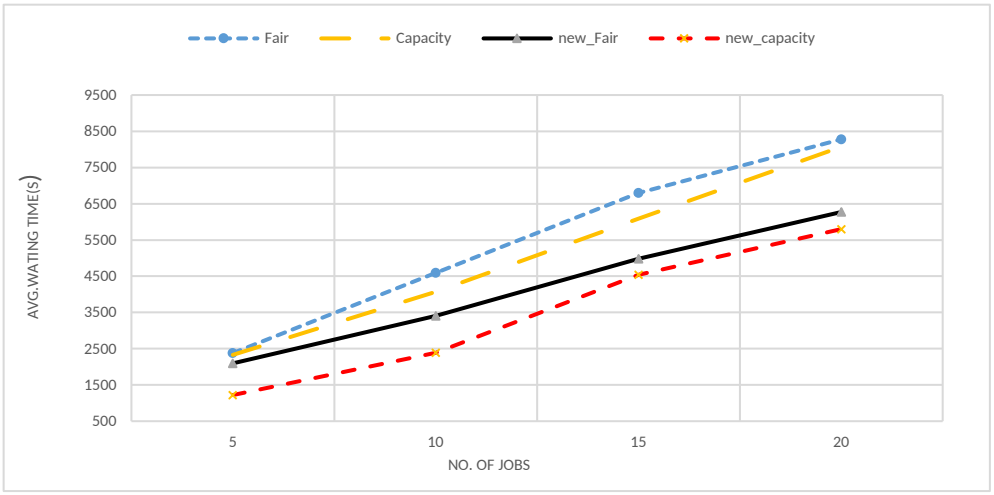


Figure 7: Average waiting time for Inverted index

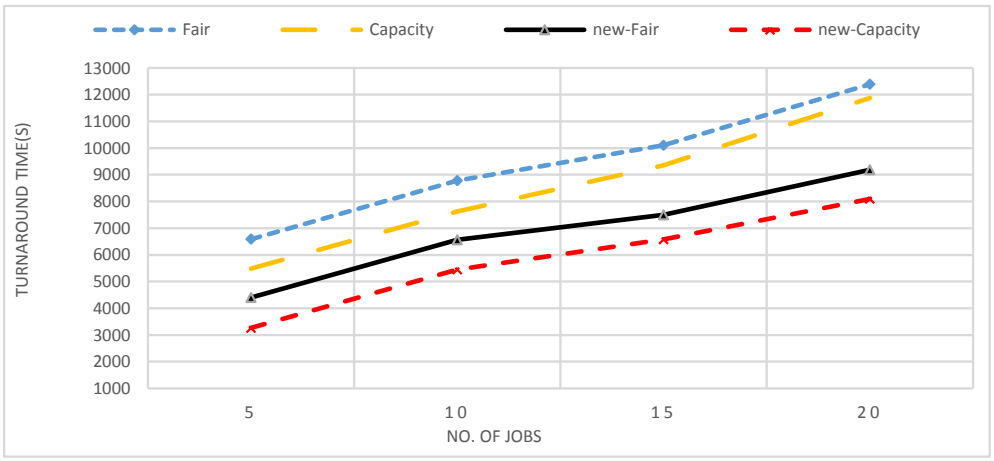


Figure 8: Average turnaround time for Inverted index

As the results show, new_Capacity shows better results than Capacity. Also, new_Fair is better than the Fair. It means, when the system uses the proposed method for scheduling jobs and determines their priority, the average waiting and turnaround times are decreased. Additionally, the higher the number of jobs, the better the results achieved. After running several similar tests, the investigation of the results shows an approximately 45% reduction in average waiting time and a 40% reduction in average turnaround time compared to the Hadoop default algorithms.

After evaluating the proposed method in prioritizing jobs, we used the Ganglia monitoring tool to investigate CPU and Memory performances. Ganglia is a distributed and scalable monitoring tool for high-performance computing systems, clusters, and networks. This software is used to view live or recorded statistics from criteria such as average CPU load, network usage, and memory [12, 15, 19].

For this purpose, two groups of experiments were taken. In the first set of tests, about 12GB jobs were given to the system and the default Hadoop scheduler was activated. Fig. 9 shows the CPU performance and consumed memory. The second experiment was repeated with the previous set of jobs, but in this experiment, the proposed method was used to prioritize the tasks. Fig. 10 shows the amount of memory consumption and CPU performance. In Figures 9 and 10, CPU performance is indicated by the diagram on the right. In these diagrams, the x-axis shows the CPU operating time and the y-axis the percentage of CPU usage. According to Figures 9 and 10, if Hadoop's default scheduler is enabled, the average CPU usage percentage is 9% and the average CPU wait is 20.4%, and the jobs are done in interval 15:50 to 16:50. If the proposed method is used to prioritize tasks, the average CPU usage percentage will increase to 12% and the average CPU wait will decrease to 15.5%, and the jobs are done in interval 5 to 10:45. This is a significant reduction in time.

In terms of memory consumption, using the proposed method involves a shorter amount of memory to perform tasks.

We repeated the experiment with the conditions above. Fig. 11 shows the CPU performance in the case where the default Hadoop scheduler is used, and Fig. 12 shows the CPU performance in the case that our proposed method is used. According to these diagrams, if we use the default scheduler, the average percentage of CPU usage is 7.2%, and if the proposed method is used, this increases to 13.1%. Also, using the proposed method, the average CPU wait is reduced from 21.2% to 17.4%. Similarly, the time interval is reduced from 17:22 - 18:21 to 12:21 - 13: 7.

According to Figures 13 and 14, the memory time usage is shorter if the proposed method is used. Reviewing the performance of other jobs showed similar results.

Figure 9: CPU and memory usage in Hadoop default scheduler

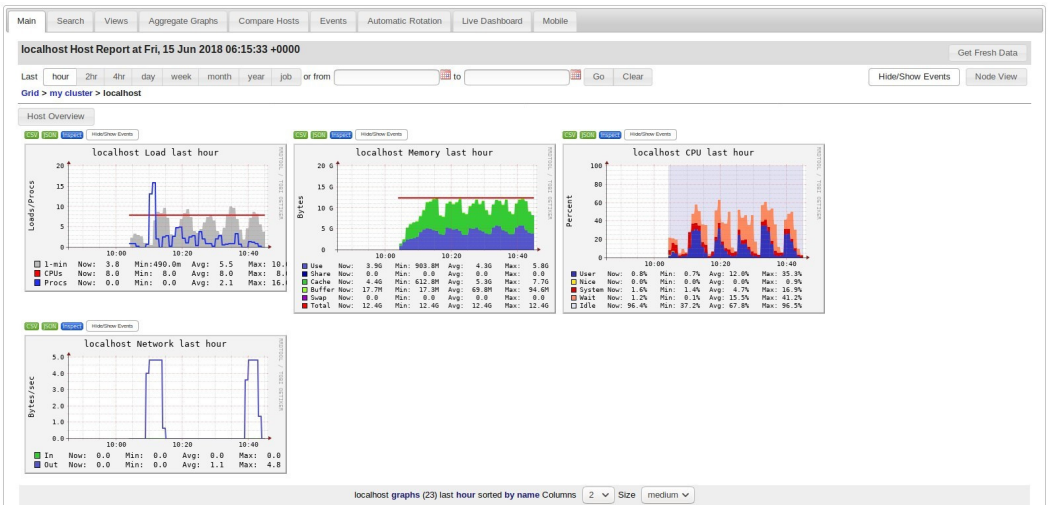


Figure 10: CPU and memory usage in the proposed method

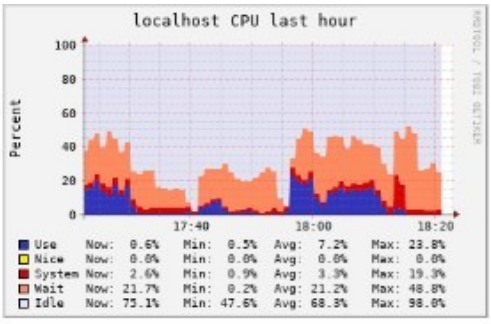


Figure 11: CPU performance using the default Hadoop scheduler

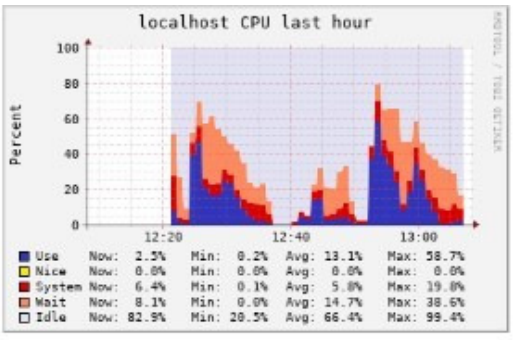


Figure 12: CPU performance in using the proposed method

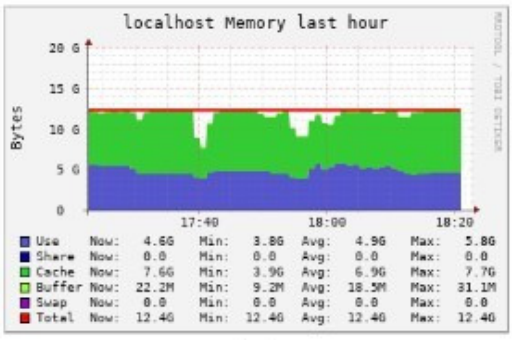


Figure 13: Memory performance using the default Hadoop scheduler

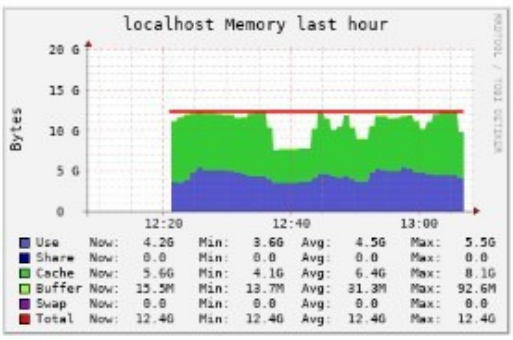


Figure 14: Memory performance using the default Hadoop scheduler

We compared our proposed method with other studies [3, 4, 16, 23, 25] that had a similar approach to ours. They also intended to improve Hadoop default job scheduling and focus on reducing the average turnaround or job completion times. Fig. 15 shows the result of this comparison.

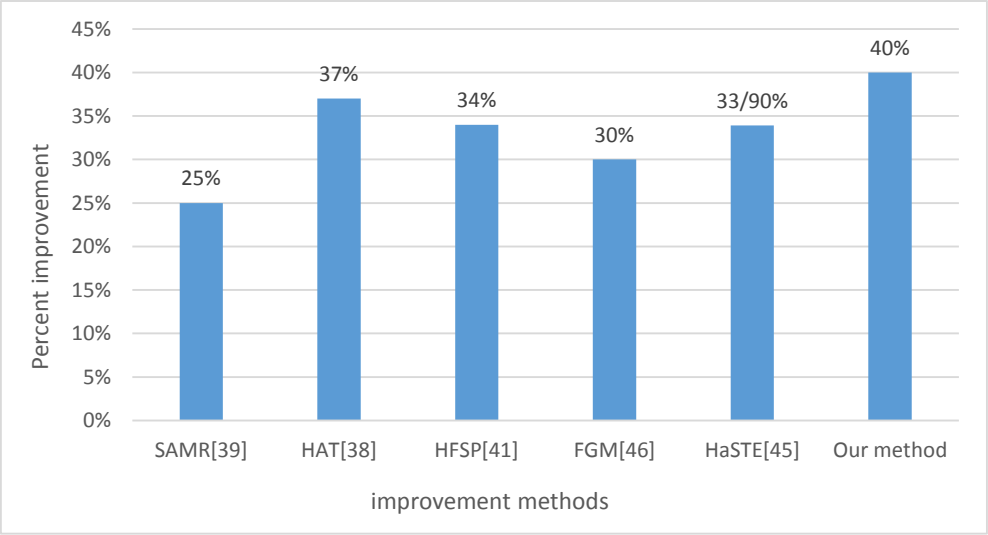


Figure 15: The percentage improvement different methods versus in comparison Hadoop default scheduling

According to Fig. 15, our method has a higher improvement percentage than other methods in job completion times.

6 Conclusion and future works

In this paper, a novel dynamic method has been presented to improve the default job scheduling in Hadoop MapReduce. This method determines the priority of execution by allocating a score to each job. The execution priority is assigned to a job that has the highest score among the jobs. To calculate this score, parameters such as waiting time, the runtime of each job, the input data size, and the number of jobs in the waiting queue are considered. The experiments show the proposed priority method improved the default Hadoop job scheduler by approximately 45% in the average waiting time and by 40% in the average turnaround time. Also, by using the proposed method, not only CPU performance improved but memory usage decreased as well.
In future work, we plan to extend the proposed scheme by using machine learning techniques and optimize Hadoop parameter values by using metaheuristic algorithms to reduce runtime. Also, we intend to work on Spark and Flink scheduling algorithms to process streaming data.

References

[1] Akhtar MN, Saleh JM, Awais H, Bakar EA. Map-Reduce based tipping point scheduler for parallel image processing. Expert Systems with Applications. 2020 Jan 1;139:112848.

[2] Alapati SR. Expert Hadoop administration: managing, tuning, and securing spark, YARN, and HDFS. Addison-Wesley Professional; 2016 Nov 29.

[3] Chen Q, Guo M, Deng Q, Zheng L, Guo S, Shen Y. HAT: history-based auto-tuning MapReduce in heterogeneous environments. The Journal of Supercomputing. 2013 Jun 1;64(3):1038-54..

[4] Chen Q, Zhang D, Guo M, Deng Q, Guo S. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In2010 10th IEEE International Conference on Computer and Information Technology 2010 Jun 29 (pp. 2736-2743). IEEE.

[5] Chunlin L, Jianhang T, Youlong L. Multi-queue scheduling of heterogeneous jobs in hybrid geo-distributed cloud environment. The Journal of Supercomputing. 2018 Oct 1;74(10):5263-92.

[6] Dhulavvagol PM, Totad SG, Sourabh S. Performance Analysis of Job Scheduling Algorithms on Hadoop Multi-cluster Environment. InEmerging Research in Electronics, Computer Science and Technology 2019 (pp. 457-470). Springer, Singapore..

[7] Hashem IA, Anuar NB, Marjani M, Ahmed E, Chiroma H, Firdaus A, Abdullah MT, Alotaibi F, Ali WK, Yaqoob I, Gani A. MapReduce scheduling algorithms: a review. The Journal of Supercomputing. 2020 Jul;76(7):4915-45.

[8] Hashem IA, Anuar NB, Marjani M, Gani A, Sangaiah AK, Sakariyah AK. Multi-objective scheduling of MapReduce jobs in big data processing. Multimedia Tools and Applications. 2018 Apr 1;77(8):9979-94.

[9] He C. Scheduling in Mapreduce Clusters. Computer Science and Engineering: Theses, Dissertations, and Student Research. 148; The University of Nebraska - Lincoln, 2018; https://digitalcommons.unl.edu/computerscidiss/148

[10] https://engineering.purdue.edu/ puma/pumabenchmarks.htm, last visited: August 2020

[11] http://hadoop.apache.org/docs/r2.9.1/index.html, last visited: May 2020

[12] http://ganglia.sourceforge.net/; last visited: March 2020

[13] Johannessen R, Yazidi A, Feng B. Hadoop MapReduce scheduling paradigms. In2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA) 2017 Apr 28 (pp. 175-179). IEEE.

[14] Lin JC, Lee MC. Performance evaluation of job schedulers on Hadoop YARN. Concurrency and Computation: Practice and Experience. 2016 Jun 25;28(9):2711-28.

[15] Massie M, Li B, Nicholes B, Vuksan V, Alexander R, Buchbinder J, Costa F, Dean A, Josephsen D, Phaal P, Pocock D. Monitoring with Ganglia: tracking dynamic host and application metrics at scale. " O'Reilly Media, Inc."; 2012 Nov 9.

[16] Pastorelli M, Carra D, Dell'Amico M, Michiardi P. HFSP: bringing size-based scheduling to Hadoop. IEEE Transactions on Cloud Computing. 2015 Jan 23;5(1):43-56.

[17] Perera S. Hadoop MapReduce Cookbook. Packt Publishing Ltd; 2013..

[18] Peyravi N, Moeini A. Estimating runtime of a job in Hadoop MapReduce. Journal of Big Data. 2020 Dec;7(1):1-8.

[19] Prashant Bhamidipati; GANGLIA INSTALLATION GUIDE; http://www-hep.uta.edu/hep_notes/computing/computing_0025.pdf

[20] Teng F, Magoulès F, Yu L, Li T. A novel real-time scheduling algorithm and performance analysis of a MapReduce-based cloud. The Journal of Supercomputing. 2014 Aug 1;69(2):739-65.

[21] Tian W, Li G, Yang W, Buyya R. HScheduler: an optimal approach to minimize the makespan of multiple MapReduce jobs. The Journal of Supercomputing. 2016 Jun 1;72(6):2376-93.

[22] White, T., Hadoop: The definitive guide, Fourth Edition, 2015;Published by OReilly Media, Inc.

[23] Yao Y, Gao H, Wang J, Sheng B, Mi N. New scheduling algorithms for improving performance and resource utilization in hadoop YARN clusters. IEEE Transactions on Cloud Computing. 2019 Jan 23.

[24] Yao Y, Tai J, Sheng B, Mi N. LsPS: A job size-based scheduler for efficient task assignments in Hadoop. IEEE Transactions on Cloud Computing. 2014 Jul 30;3(4):411-24.

[25] Zhou M, Dong X, Chen H, Zhang X. Fine-grained scheduling in multi-resource clusters. The Journal of Supercomputing. 2020 Mar;76(3):1931-58.