A Novel Approach of Fair Scheduling to Enhance Performance of Hadoop Distributed File System

Rubayet Hussain
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology
Gazipur, Bangladesh
* Email: erahat.cse@duet.ac.bd

Khawja Imran Masud
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology
Gazipur, Bangladesh
Email: eimran.cse@gmail.com

Md. Nasim Akhtar
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology
Gazipur, Bangladesh
Email: drnasim@duet.ac.bd

Mostafijur Rahman
Department of Software Engineering
Daffodil International University, Bangladesh
Dhaka, Bangladesh
Email: mostafijur.cse@gmail.com

Sheikh Md Roky
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology
Gazipur, Bangladesh
Email: eroky.cse@gmail.com

Tanjila Akter Tarin
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology
Gazipur, Bangladesh
Email: tanjilaaktertarin46@gmail.com

***Abstract*— Nowadays, big data is one of the most challenging issues for managing big amounts of data more effectively and efficiently. It widely used in E-commerce, social media, online business and such types of applications. Hadoop Distributed File System (HDFS) is one of the widely used frameworks which can easily handle and store large amounts of data set frequently. For HDFS job scheduling is more challenging because it plays an important role in time optimization in big data. For resolving this issue in this paper, we introduce a job scheduling algorithm which is more time efficient and accurate than existing fair job scheduling algorithm. We optimize the time cycle of fair scheduling by minimizing iteration. We have accelerated with the different number of jobs in the existing algorithm and proposed an algorithm for experimentally proving the time complexity and time measurement. It is observed that the proposed method is computationally efficient than the existing one and our algorithm has reduced the number of iterations and improved the time efficiency on average 26.719%.**

***Keywords*— Big Data, HDFS, Map Reduce, Scheduling**

I. Introduction

In information technology, big data is the very popular term. Big data [7] can manage large range of multiple data structure. It is difficult to manage big data by conventional system. It can be used in many ways. In recent analysis big dataset are increasing exponentially. These datasets are very difficult and complicated. In several fields data has increased in large scale. Data comes from different sources like media sites, business transactions, online processing, engineering, astronomy, social activities, medical science, banking and many other fields.

Hadoop Distributed File System (HDFS) is the most popular file system framework for handling datasets. It's improved the reliability, efficiency, and dependability of datasets. The MapReduce is esoteric programming models of HDFS. The MapReduce functions work through two modules called Map function and Reduce function. The Map function generates the pairs of key value depend on inputs pairs of keys. The Reduce function merges all similar key pairs into one. In Map reduce model there are Task Tracker, Job Tracker, Name Node, Data Node etc. In task Tracker part various task like process and execute with the arrival job can be done. Job Tracker used in various scheduling algorithm send task to the Task Tracker. Job scheduling is the most important issues to achieve the better performance. For this reason, we are motivated to improve the performance of job scheduling algorithm.

II. Related Work

A. *Big Data*

Big data [1,18] refers to large and complex data sets. In traditional data management application, it's difficult to managing, processing and scheduling with a very large amount of data. Storage, analysis, searching, sharing, and visualization of large amount of data is almost challenging. Big data is also similar as small data but important thing is that it's bigger. Big data has three V's which means volume, variety, and velocity. Big data has structure and unstructured data format where the unstructured data is not stored in systematic and well-defined manner and structured data are well organized. For example Wikipedia, Google, Facebook, amazon etc. has unstructured data formats and e-commerce sites having structured data formats. The data set is increased in day by day. The conventional system is very challenging to handle unstructured data. For that reason, it's very important to manage unstructured data. To handled unstructured data Apache Hadoop [9-12] is one of the most popular open source framework. Hadoop have two components, one is Hadoop Distributed file system (HDFS) and another is MapReduce.

B. Hadoop Distributed File System (HDFS)

HDFS [2,4,14-17] is a Distributed File System (DFS) where distributed files are store in storage system by following HDFS nodes. Here big data are divided into blocks which size is basically 64MB or 128MB. For each block it creates three copies and these copies of block is stored in three different machines. HDFS has two main parts. One of them is NameNode and another one is DataNode. The architecture of HDFS based on master-slave architecture. The master part is called single NameNode which store metadata and the slave node is called DataNode which store application data.

NameNode is managed by the file system name-space and file are controlled by NameNode which is access by clients. The file system operation (opening, closing and renaming files and directories) are executed by NameNode. Data Node is responsible for reserving read and write request from the file system's clients and also determine the mapping of blocks from data nodes.

C. MAPREDUCE

MapReduce [5] is a programming design pattern which is used for data intensive computing in the distributed computing environment. This section discusses the architecture of MapReduce and also how scheduling of jobs is completed in MapReduce. The processing technique of MapReduce is based distributed computing. MapReduce introduce a programming model which consist of two important functions. [3, 6]. One of the functions is Map which take a set of data key-value pairs as an input and generate the output as another set of intermediate key value pairs . The another one is Reduce function receives this intermediate key value pair generated by the Map function as an input and reduces this set of intermediate key value pairs into a smaller set of pairs. MapReduce is parallelized on top of HDFS and processes the data which resides on HDFS. In a typical MapReduce job, input files are read from the HDFS and each Map task is preferably scheduled on a machine that contains a replica of the corresponding input data.

The replication feature is provided by HDFS and it is used by MapReduce for improving the performance. The architecture of MapReduce consists of one Master node (Job Tracker) and many Slave nodes (Task Trackers). A Job Tracker manages a number of slave nodes, known as the Task Trackers. The Name Node, present at a master node, manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree. A Data Node, usually one per node, stores HDFS data in its local file system. The DataNodes store and retrieves blocks when they get instructions from clients or the Name Node and reports back to the Name Node periodically with lists of blocks.

III. JOB SCHEDULING ALGORITHM IN HADOOP

A. FAIR SCHEDULING

Fair Scheduling algorithm controls the allocation of resources each job of processes. In this scheme all resources are among jobs fairly. Fair Scheduling is developed by Facebook [1,6,5,3]. Map Reduce is the important function of Hadoop. Facebook has developed Fair Scheduling to manage the cluster of Map Reduce in Hadoop. In Map Reduce all cluster are managed by Scheduling. The main purpose of Fair Scheduling is to assigned every users fairly share of the cluster capacity over an amount of time. Users allocated the jobs in to job Pools, using less amounts of Map and Reduce slots. Fair Scheduling supports the preemption. This scheduling allows the slots to the pool processing over capacity. Fair Scheduling is similar to capacity scheduling with small differences. In capacityscheduling there has been used multiple queues. Also Fair Scheduling used multiple queues terms as Pools. When the jobs are collected from the Pools and also given their parts of the resources. Assume that another job arrive to the Pool. In this case the capacity scheduling would execute it FIFO scheduling. As a result high priority process will be waiting for long amount of time. This problem can be improves in Fair Scheduling Algorithm.

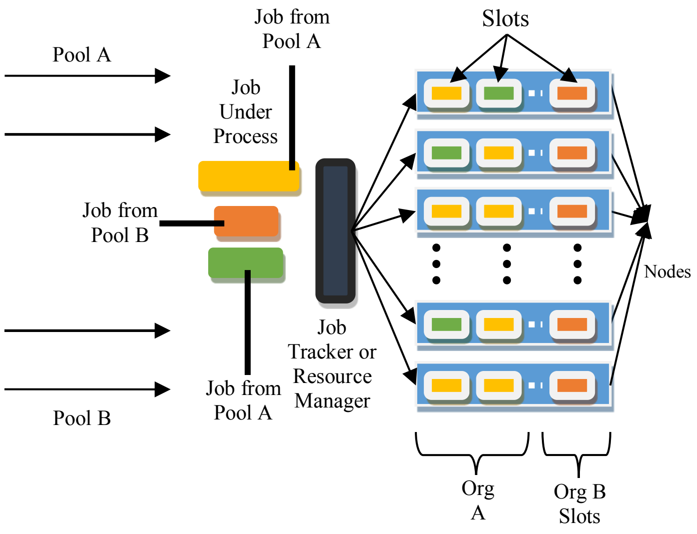

Fig. 1. Job Execution Architecture of Fair Scheduling. [6]

The jobs are collected which is waiting in the queue for long time. Then it would be execute in parallelly. Fair scheduling improved the performance in Hadoop distributed file system by proper utilization of fairly sharing the resources for all jobs. It allocates resources equally to each job to achieve a good output via using available resources. It generates groups of jobs stand on the configurable attributes like user name called Pools. Defaults Pools have shares resources equally. If there are any Pools which is unfilled then it has been filled by other Pools Slot. If Pools send many more jobs, the Fair Scheduling scheme detects those jobs and identifying those jobs marked as a not executable job. This Fair Scheduling algorithm performs good in small and large data sets.

A.1 NON-WEIGHTED MAX-MIN FAIR SHARING SCHEDULING

The Fair Scheduling Algorithm implementation is very complicated for its configuration. In Non-Weighted Max-Min Sharing Algorithm shared resources count the weight of allocated all resources equally to all users, unless sometimes all users need not share whole space. If any user doesn't need the share whole space fully then the rest part of the resources is not used by other processes. In this algorithm scheme if any users need not share whole space the remaining unused unit capacity has been shared by other job by dividing equally among the remaining large job in recursive method [13].

The main idea of the Non-Weighted Max-Min Sharing Algorithm is all users are given an equal share of the total resources. The Non-Weighted Max-Min Sharing Algorithm decreases the task rates in a fair way. The demanded capacity is equal to its total processor capacity. Since the input all jobs is equally important so that initially divides all units of processor capacity for equal sharing resources for all jobs. As a result, all jobs get shared resources within limited unit capacity which is taken by dividing total processor capacity by its total task. The remaining unit capacities are divided by remaining jobs. So that the remaining all jobs equally sharing the unit capacity. At this time, it has been used the resources efficiently. There has been no unit time are unused. All unit time has been used by all jobs.

In Non-Weighted Max-Min Sharing Algorithm does not count the weight of each job. Which has been affect the Non-Weighted Max-Min Sharing Algorithm, so that when any job has been finished before the quantum unit of capacity the reduce space of unit capacity will not be used. In the meantime, if it will be used by other job, all jobs will probably execute less time. On the other side it does not count the weight of each job, so that the performance rate is unbalanced. Non-Weighted Max-Min Sharing Algorithm has been used only for non-weighted job processes.

The Non-Weighted Max-Min Sharing Algorithm idea has been described by an example, where four jobs arrive as request services with the rates are 10, 5,7,17 respectively. Assume that the processor capacity offered equal 32 units. So, we get the total demanded capacity are (10+5+7+17) =39 units, see that the demanded capacity is larger than total offered processor capacity. At this situation this Non-Weighted Max-Min Scheduling decrease the job rate in a fair method such that the demanded job rate is equal with the offered processor capacity rate. For this approach the processor capacity is divided by total no of jobs which means that there has same priority for all job's execution. So, the algorithm initially divides processor capacity 32 by total requested jobs 4. That is 32/4=8 units.

We see that the second and third job request has been completed in first iteration because of the input rates are less than 8 units (5 and 7 respectively), so that the second and third request are getting the rate.

The other two requests like first and fourth request are not completely getting rate because of the rates of both request greater than 8 units (10 and 17 respectively), therefore the first iteration of this algorithm the 8 units of rates are assign for both requests respectively. At the last moment of first iteration the residue will be (32-28=4) [where (8+5+7+8) =28)] where 8,5,7,8 are completed job rates of first, second, third and fourth job respectively. In the next iteration the remaining of 4 units are equally shared among first and fourth job request, where the actual rates of first demanded jobs are less than the remaining units. That is 4/2=2 units. Since the first job request is equal than 10[8+2] units.

TABLE I. AN EXAMPLE OF NON-WEIGHTED MAX-MIN FAIR SHARING SCHEDULING WITH PROCESSOR CAPACITY 39

Demanded Rate	First Iteration	Second Iteration
10	8	2
5	5	0
7	7	0
17	8	2
Residue	32-28=4	0

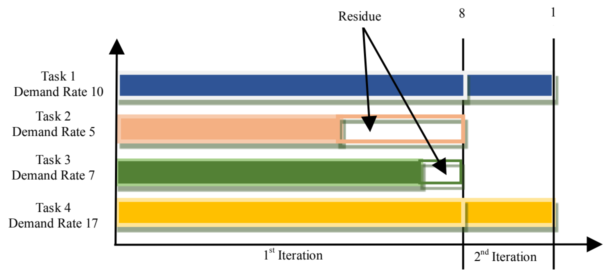

Fig. 2. A graphical conceptualization of the Weighted Max-Min Fair Scheduling Algorithm example [13]

As a result, the first job has been getting the rates to complete the request. On the other side the fourth job request also be executed in this 10 [8+2] units at last moment of the second iteration. So, the Non-Weighted Max-Min Sharing Scheduling has been completed the X_i rates of Z_i job request in the iteration. In this scheduling when the second and third task has been completed, residue unused units are not used by other jobs. So that it cannot provide the best utilization of resources also need more iteration to complete all task

A.2 *WEIGHTED MAX-MIN FAIR SHARING SCHEDULING*

The Non-Weighted Max-Min Sharing Scheduling is considered that all jobs or task are equal [6]. The Weighted Max-Min Fair Sharing Scheduling considers the weight of each job because of users may have various kinds of task which have different priorities. Assume that each task assigned an integer weight. Each task is assigned weight value depend on their priority to share resources by all job. Assume that first and fourth task weight is 1. The second and third task is assigned weight as 2. The core idea of Weighted Max-Min Sharing Scheduling is shared resources to all jobs depend on the weight so that it will helps to consider the weight and also maintain the job priority. By using the weight of each job efficiency and execution time will be decreased. The Weighted Max-Min Fair Sharing Scheduling can be describing by an example which has been described before through the Non Weighted Max-Min Fair Sharing Scheduling. Consider that the total weights of all task (1+2+2+1) =6. The demanded capacity of all task is (10+5+7+17) =39 units and the offered processor capacity is 32 units.

TABLE II. AN EXAMPLE OF WEIGHTED MAX-MIN FAIR SHARING SCHEDULING WITH PROCESSOR CAPACITY 39.

Demanded Rate	Weights	First Iteration	Second Iteration
10	1	5.33	4.66
5	2	5	0
7	2	7	0
17	1	5.33	4.66
Residue		32-22.68=9.32	0

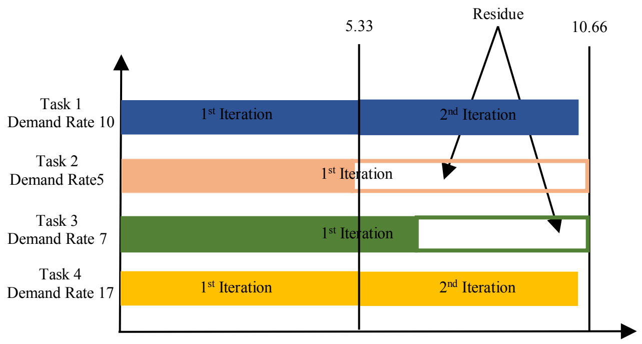

Fig. 3. A graphical representation of the Weighted Max-Min Fair Scheduling Algorithm example [13].

The Weighted Max-Min Fair Sharing Scheduling algorithm works by dividing the total processor capacity units by the total weight of task. That is (32/6) = 5.33 units. For each weight of each task is assigned 5.33 units. According to this scheduling algorithm in first iteration, first and fourth task are assigned in 5.33 units because of their weight is 1. Other two job (second and third) are assigned in (5.33×2) = 10.66 units. However, demanded capacity for second job is 5 units, which is less than assign rate of 10.66 units. Instead, a remaining of 5.66 units is obtained. Also the third job demanded capacity is 7 units which is less than assign rate of 10.66 units. Also remaining of 3.66 units is obtained. The demanded rates of one and fourth tasks are greater than initial assign of capacity. However, there has no residual capacity. The total remaining capacity of second and third task is (3.66+5.66) = 9.32 units. The first and fourth job can get additional capacity dividing the total remaining capacity by their weights. That is 9.32/2 = 4.66 units. In this reason, the first task will get (5.33+4.66) = 9.99 units. The fourth task will get (5.33+4.66) = 9.99 units. Thus, the algorithm has been terminated in second iteration.

IV. FORMAL PROBLEM DEFINATION AND PROPOSED SYSTEM

The proposed system is modifying the Max- Min weighted fair sharing scheduling algorithm to improve the performance. In job scheduling, performance is depend on execution time & space. Max- Min weighted fair sharing scheduling algorithm, the processor capacity is shared among job depend on their weight. In existing system, it needs more iteration and also there has some remaining capacity. In this case, remaining capacity is divided into other incomplete job. At this point, this residue capacity is used by other jobs but it needs more execution time.

A. *PROPOSED SYSTEM ALGORITHM*

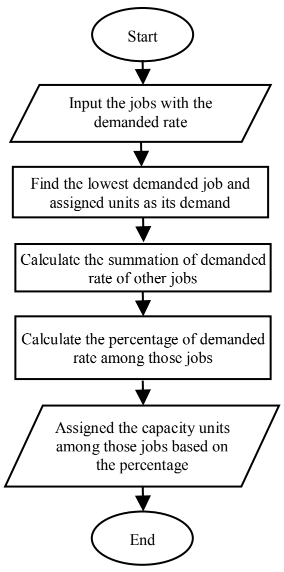
Start
Input the jobs with the demanded rate
Find the lowest demanded job and assigned units as its demand
Calculate the summation of demanded rate of other jobs
Calculate the percentage of demanded rate among those jobs
Assigned the capacity units among those jobs based on the percentage
End

Fig. 4. Algorithm of proposed system.

B. *IMPLEMENTATION*

The proposed system is very efficient and it’s possible to solve this existing problem that has mentioned. We solved this problem based on demanded rate of percentage (%) calculation. The core idea is, in this proposed system all resources capacity is shared by the all jobs according to percentages of their demand rate.

TABLE III. AN EXAMPLE OF THE PROPOSED SYSTEM WITH PROCESSOR CAPACITY 39

Demanded Rate	Assigned Capacity for lowest demanded job	PRCPJ	Assigned Capacity Unit
10	-	29.41	7.94
5	5	-	-
7	-	20.58	5.55
17	-	50.00	13.5
Residue	32-5 = 27		27-27=0

For reducing the iterations, we have considered these two equations which can helps to find out the percentage of the capacity.

1. Calculate the Percentage of required capacity unit per job within total demand (PRCPJ)

$$= \frac{\textbf{\textit{Job Demanded Rate}}}{\textbf{\textit{Total Demanded Rate}}} \times 100 \,.$$

2. Required capacity for each job within Processor Capacity

$$= \frac{\textbf{\textit{PRCPJ}} \times \textbf{\textit{Remaining Processor Capacity}}}{\textbf{100}} \,.$$

The proposed system has been released the processor capacity at first for this job which is needed minimum numbers of capacity. Further, the remaining processor capacity has been shared to all jobs according to amount of percentages. This remaining capacity has been calculated using equation (1) & (2) which written in above. As a result the residue processor capacity is properly utilized. All processes are executed with in one iteration only. As a result the job completion time is reduced.

The idea of the proposed system can be explained by the following example. Assume that, there are four jobs which are requested with demand rates 10, 5, 7 and 17 respectively. Then, total demand rate would be (10+5+7+17) = 39 units. The available processor capacity would be 32 units. At first, the smallest demanded rate of job is allocated from resources. In this example the smallest demanded job rate is 5. It will share the resources according to its demand. After sharing the demanded resource of second job remaining processor capacity would (32-5) =27 units. So, remaining demanded rate would be (10+7+17) = 34 units. Now we have calculated the first job Percentage of required capacity unit per job within total demand (PRCPJ) by equation-1 and the resultant value is 29.41%. For demand rate 7 and 17 we have PRCPJ is 20.58% and 50.00% respectively.

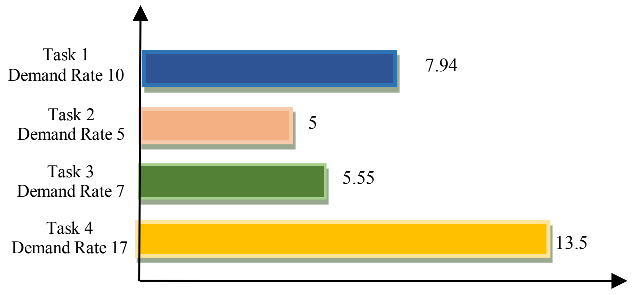

Fig. 5. A graphical conceptualization of the proposed algorithm example

Now we need to calculated the required capacity for each job within processor capacity by equation-2. For demand rate 10, 7 and 17 it has 7.94, 5.55 and 13.5 units respectively. By following this idea, we have calculated all of this calculation in only one iteration. If this same example is solved by Weighted Max-Min Fair Sharing Scheduling algorithm, it needed two iterations. So, we concluded that our proposed system is reduced the sharing time, efficiency and increased reliability then existing one.

V. RESULT AND DISCUSSION

A. *COMMPLEXITY*

Complexity is an important criterion for an algorithm. The time complexity of an algorithm quantifies the amount of time taken by an algorithm. We have analysis the current system time complexity and also evaluate our proposed system time complexity. We know that the time complexity of fair scheduling algorithm is O (logn) [18]. We have concluded that our proposed systems complexity is O(1).

B. *COMPARISON OF RESULTS*

For our experimental proof, we simulated our proposed algorithm with different number of jobs. We calculate the execution time for 50, 100, 150, 200, 250 and 300 jobs for both existing fair job scheduling algorithm and our proposed job scheduling algorithm. We have accelerated that our proposed system is good enough then previous one and execution time is also reduced. Details of execution time for different number of jobs is given below in Table-4. We observe that our proposed method is computationally efficient than the existing method and our algorithm has reduced the number of iterations and improved the time efficiency on average 26.719%.

TABLE IV. EXPERIMENTAL RESULT TABLE

Number of Jobs	**Time (ms)**	
	Fair Scheduling	***Proposed Method***
50	15.9878	11.3455
100	29.2628	18.3346
150	31.8067	27.5487
200	49.2724	32.7156
250	68.9059	39.861
300	82.7516	46.2981

C. *GRAPHICAL ILLUSTRATION OF EXPERIMENTAL RESULT*

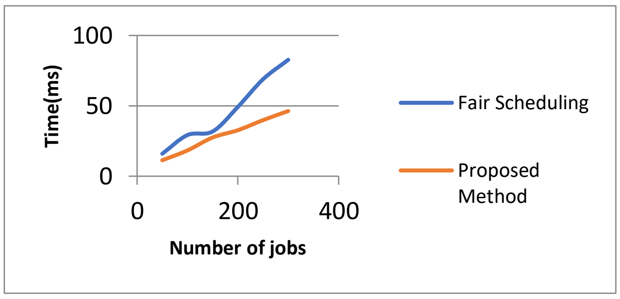

Fig. 6. Graphical view of time measurments of existing and proposed system

VI. CONCLUSION

In this paper, we have analysis different job scheduling algorithms which are used by MapReduce techniques. Among them we have selected to modify the weighted Max-Min Fair Scheduling Algorithm. We overcome the drawback of this algorithm. Our modification is reduced the iteration and time among previous one. We proved mathematically as well as practically that proposed algorithm executes faster than the existing one.

REFERENCES

[1] S. Dhingra," Scheduling Algorithms in Big Data: A Survey", International Journal of Engineering and Computer Science, ISSN: 2319-7242, Vol.-5, no .18, pp.17737-17743, 2016.

[2] I. Abaker, T. Hashem, I.Yaqoob, N. Anuar, S. Mokhtar, AbdullahGani,S. UllahKhan,"The Rise of The Big Data on

Cloud Computing: Review and Open Research Issues", Vol. 47 , pp. 98-115, 2015.

[3] D. Pan & Y. Yang. "A Constant Complexity Fair Scheduler with O (log N) Delay Guarantee". ITC'19, Proceedings of Poster Beijing University of Posts and Telecommunications Press, pp. 2401-2410,2019.

[4] B. J. Mathiya and V. L. Desai, "Apache Hadoop Yarn Parameter Configuration Challenges and Optimization", 2015 International Conference on Soft-Computing and Networks Security (ICSNS), Coimbatore, pp.1-6. 2015.

[5] Scheduling in Hadoop, Available in: https://developer.ibm.com/articles/os-hadoop-scheduling/, accessed on 15, February 2018.

[6] M. Usama, M. Liuiu, M. Chen, "Job Schedulers for Big Data Processing in Hadoop Environment: Testing Real-Life Schedulers using Benchmark Programs", vol. 3, No. 4, p.p. 260-273, 2017.

[7] J. V. Gautam, H. B. Prajapati, V. K. Dabhi and S. Chaudhary, "A Survey on Job Scheduling Algorithms in Big Data Processing," IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), pp. 1-11, 2015.

[8] S. Bende, R. Shedge," Dealing with Small Files Problem in Hadoop Distributed File System", Procedia Computer Science, Vol. 79, pp. 1001-1012, 2016.

[9] Aache Software Foundation, Hadoop Apache, Accessed on 10th April, 2018. URL: https ://hadoop.apache.org/.

[10] C. Sreedhar, N. Kasiviswanath and P. C. Reddy. "A Survey on Big Data Management and Job Scheduling", International Journal of Computer Applications, 130(13), p.p. 41-49, 2015.

[11] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling". In Proceedings of the 5th European conference on Computer systems (EuroSys '10). ACM, USA, pp. 265-278. dio: http://dx.doi.org/10.114 5/1755913. 1755940.

[12] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, pp. 1-10, 2010,

[13] J. Dean, S. Ghemawat," MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, 2008.

[14] S. Shaikh and D. Vora, "YARN versus Map Reduce: A comparative study", 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, pp. 1294-1297, 2016.

[15] K. Aziz, D. Zaidouni and M. Bellafkih, "Real-time dataanalysis using Spark and Hadoop", 4th International Conference on Optimization and Applications (ICOA), Mohammedia, 2018, pp. 1-6. doi: 10.1109/ICOA.2018.8370593

[16] X. Dai and B. Bensaou, "Scheduling for response time in Hadoop MapReduce," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016, pp. 1-6.doi: 10.1109/ICC.2016.7511252

[17] N. Doulamis, E. Varvarigos1 and T. Varvarigou, "Fair Scheduling Algorithms in Grids", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 11, pp. 1630-1648, 2007.

[18] X. Li, T. Jiang, R. Ruiz," Heuristics for Periodical Batch Job Scheduling in A Map reduce Computing Framework", Vol-326, pp. 119-133, 2016.