International Journal of Computer Networks and Communications Security

VOL. 2, NO. 9, SEPTEMBER 2014, 308–317
Available online at: www.ijcncs.org
ISSN 2308-9830

A Comprehensive View of Hadoop MapReduce Scheduling Algorithms

Seyed Reza Pakize

Department of Computer, Islamic Azad University, Yazd Branch, Yazd, Iran

E-mail: s.rezapakize@gmail.com

ABSTRACT

Hadoop is a Java-based programming framework that supports the storing and processing of large data sets in a distributed computing environment and it is very much appropriate for high volume of data. it's using HDFS for data storing and using MapReduce to processing that data. MapReduce is a popular programming model to support data-intensive applications using shared-nothing clusters. the main objective of MapReduce programming model is to parallelize the job execution across multiple nodes for execution. nowadays, all focus of the researchers and companies toward to Hadoop. due this, many scheduling algorithms have been proposed in the past decades. there are three important scheduling issues in MapReduce such as locality, synchronization and fairness. The most common objective of scheduling algorithms is to minimize the completion time of a parallel application and also achieve to these issues. in this paper, we describe the overview of Hadoop MapReduce and their scheduling issues and problems. then, we have studies of most popular scheduling algorithms in this field. finally, highlighting the implementation Idea, advantages and disadvantage of these algorithms.

Keywords: *Hadoop, Map Reduce, Locality, Scheduling algorithm, Synchronization, Fairness.*

1 INTRODUCTION

Hadoop is much more than a highly available, massive data storage engine. One of the main advantages of using Hadoop is that you can combine data storage and processing [1]. it can provide much needed robustness and scalability option to a distributed system as Hadoop provides inexpensive and reliable storage. Hadoop using HDFS for data storing and using MapReduce to processing that data. HDFS is Hadoop's implementation of a distributed filesystem. It is designed to hold a large amount of data, and provide access to this data to many clients distributed across a network [2]. MapReduce is an excellent model for distributed computing, introduced by Google in 2004. Each MapReduce job is composed of a certain number of map and reduce tasks. The MapReduce model for serving multiple jobs consists of a processor sharing queue for the Map Tasks and a multi-server queue for the Reduce Tasks [3]. Hadoop allows the user to configure the job, submit it, control its execution, and query the state. Every job consists of independent tasks, and all the tasks need to have a system slot to run [2]. In Hadoop all scheduling and allocation decisions are made on a task and node slot level for both the map and reduce phases[4]. There are three important scheduling issues in MapReduce such as locality, synchronization and fairness. Locality is defined as the distance between the input data node and task-assigned node. Synchronization is the process of transferring the intermediate output of the map processes to the reduce processes as input is also consider as a factor which affects the performance [5]. Fairness finiteness have trade-offs between the locality and dependency between the map and reduce phases. due to the important issues and many more problems in scheduling of MapReduce, the Scheduling is one of the most critical aspects of MapReduce. There are many algorithms to solve this issue with different techniques and approaches. Some of them get focus to improvement data locality and some of them implements to provide Synchronization processing. also, many of them have been designed to minimizing the total completion time.

The rest of this article is organized as follows. Section 2 a brief introduction of Hadoop, introduces an overview of the MapReduce programming model and will describe main issues of scheduling in MapReduce. Section 3 some of the more popular Scheduling algorithm in Hadoop MapReduce will be described. in Section 4 we have discussion and analysis of these algorithm and shows the Taxonomy, Idea to implementation, advantages and disadvantage in the form of table. Section 5 concludes the article.

2 BACKGROUND

2.1 Overview of Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project [6], [7]. Hadoop is a, Java-based programming framework that supports the processing of large data sets in a distributed computing environment and is part of the Apache project sponsored by the Apache Software Foundation. it can provide much needed robustness and scalability option to a distributed system as Hadoop provides inexpensive and reliable storage. The Apache Hadoop software library can detect and handle failures at the application layer, and can deliver a highly-available service on top of a cluster of computers, each of which may be prone to failures [1]. Hadoop enables applications to work with thousands of nodes and terabyte of data, without concerning the user with too much detail on the allocation and distribution of data and calculation. Hadoop is much more than a highly available, massive data storage engine. One of the main advantages of using Hadoop is that you can combine data storage and processing. Hadoop using HDFS for data storing and using MapReduce to processing that data. HDFS is Hadoop's implementation of a distributed file system. It is designed to hold a large amount of data, and provide access to this data to many clients distributed across a network. HDFS consists of multiple DataNodes for storing data and a master node called NameNode for monitoring DataNodes and maintaining all the meta-data.

2.2 MapReduce Overview

MapReduce [5] is a popular programming framework to support data-intensive applications using shared-nothing clusters. MapReduce was introduced to solve large-data computational problems, and is specifically designed to run on commodity hardware and its dependent on divide-and-conquer principles. In MapReduce, input data are represented as key-value pairs. Several functional programming primitives including Map and Reduce are introduced to process the data. in the MapReduce Programming Model [7], Map algorithm include three steps: first, Hadoop and MapReduce framework produce a map task for each Input Split, and each Input Split is generated by the Input Format of job. Each <Key, Value> corresponds to a map task. in second step, Execute Map task, process the input <key, value> to form a new <key, value>. and in last step, Mapper's output is sorted to be allocated to each Reducer. Reducer algorithm also included three steps [7]: first, MapReduce will assign related block for each Reducer (Shuffle). Next step, the input of reducer is grouped according to the key (sorting step). and finally step is Secondary Sort. If the key grouping rule in the intermediate process is different from its rule before reduce. As shown in Figure 1, a mapper takes input in a form of key/value pairs (k1, v1) and transforms them into another key/value pair (k2,v2). The MapReduce framework sorts a mapper's output key/value pairs and combines each unique key with all its values (k2, {v2, v2,…}). These key/value combinations are delivered to reducers, which translate them into yet another key/value pair (k3, v3)[8].

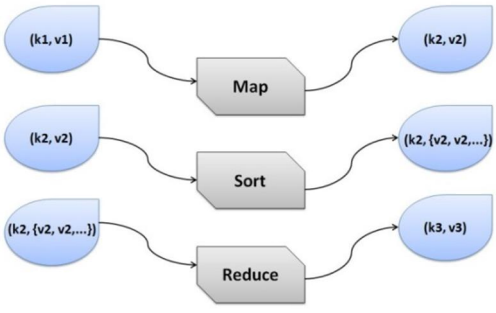

Fig. 1. The functionality of mappers and reducers [8]

2.3 Hadoop MapReduce Scheduling and Important Issues

Hadoop allows the user to configure the job, submit it, control its execution, and query the state. Every job consists of independent tasks, and all the tasks need to have a system slot to run [2]. In Hadoop all scheduling and allocation decisions are made on a task and node slot level for both the map and reduce phases [4]. The Hadoop scheduling model is a Master/Slave (Master/Worker) cluster structure. The master node (JobTracker) coordinates the worker machines (TaskTracker). JobTracker is a process which manages jobs, and TaskTracker is a process which manages tasks on the corresponding nodes. The scheduler resides in the Jobrtacker and allocates TaskTracker resources to running tasks: Map and Reduce tasks are granted independent slots on each machine [9]. In the fact, MapReduce Scheduling system takes on in six steps [5]: first, User program divides the MapReduce job. Second, master node distributes MapTasks and ReduceTasks to different workers. Third, MapTasks reads in the data splits, and runs map function on the data which is read in. fourth, MapTasks write intermediate results into local disk. Then, ReduceTasks read the intermediate results remotely, and run reduce function on the intermediate results which are read in. finally, These ReduceTasks write the final results into the output files. Figure 2 illustrate this step.

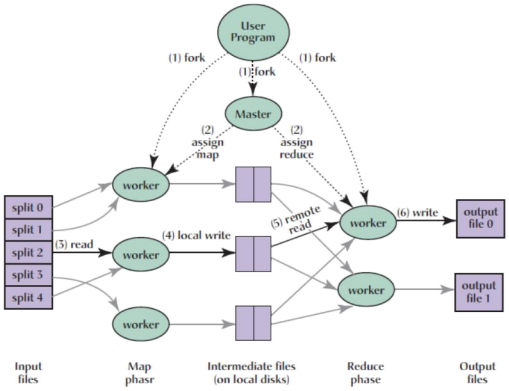

Fig.2. Overview of a MapReduce job [10]

There are three important scheduling issues in MapReduce such as locality, synchronization and fairness. Locality is a very crucial issue affecting performance in a shared cluster environment, due to limited network bisection bandwidth [5]. it is defined as the distance between the input data node and task-assigned node. The concept of data locality in MapReduce is a time when scheduler tries to assign Map tasks to slots available on machines in which the underlying storage layer holds the input intended to be processed [12]. Synchronization is the process of transferring the intermediate output of the map processes to the reduce processes as input is also consider as a factor which affects the performance. In generally, reduce process will be start a time when Map Process is absolutely finished. Due to this dependency, a single node can slow down the whole process, causing the other nodes to wait until it is finished. So, Synchronization of these two phases (Map and Reduce phase), made up essential help to get overall performance of MapReduce Model. A map-reduce job with a heavy workload may dominate utilization of the shared clusters, so some short computation jobs may not have the desired response time. The demands of the workload can be elastic, so fair workload to each job sharing cluster should be considered. Fairness finiteness have trade-offs between the locality and dependency between the map and reduce phases. When each MapReduce job has approximately an equal share of the nodes and the input files are spread in distributed file system, some map processes have to load data from the networks. This causes a great degradation in throughput and response time. Synchronization overhead could affect the fairness [11], [12], [13]. To solve these important issues many scheduling algorithms have been proposed in the past decades. in the next section we describe these algorithms.

3 SCHEDULING ALGORITMS

Due to the important issues that we described, and many more problems in scheduling of MapReduce, the Scheduling is one of the most critical aspects of MapReduce. There are many algorithms to address these issues with different techniques and approaches. Some of them get focus to improvement data locality and some of them implements to provide Synchronization processing. Also, many of them have been designed to minimizing the total completion time. in this section we describe briefly some of these algorithm.

3.1 FIFO scheduling algorithm

FIFO is the default Hadoop scheduler. The main objective of FIFO scheduler to schedule jobs based on their priorities in first-come first-out of first serve order. FIFO stands for first in first out which in it Job Tracker pulls oldest job first from job queue and it doesn't consider about priority or size of the job [14]. FIFO scheduler have many limitations such as: poor response times for short

jobs compared to large jobs, Low performance when run multiple types of jobs and it give good result only for single type of job. to address these problems scheduling algorithms such as Fair and Capacity was introducing.

3.2 Fair Scheduling Algorithm

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time [15]. If there is a single job running, the job uses the entire cluster. When other jobs are submitted, free task slots are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. It lets short jobs complete within a reasonable time while not starving long jobs [16]. The objective of Fair scheduling algorithm is to do a equal distribution of compute resources among the users/jobs in the system [17], [12]. The scheduler actually organizes jobs by resource pool, and shares resources fairly between these pools. By default, there is a separate pool for each user. The Fair Scheduler can limit the number of concurrent running jobs per user and per pool. Also, it can limit the number of concurrent running tasks per pool. The traditional algorithms have high data transfer and the execution time of jobs. Tao et al. [18] introduced an improved FAIR scheduling algorithm, which takes into account job characteristics and data locality, which decreases both data transfer and the execution time of jobs. Thus, Fair scheduling can covers some limitation of FIFO such as: it can works well in both small and large clusters and less complex. Fair scheduling algorithm does not consider the job weight of each node, which this is an important disadvantage of it.

3.3 Capacity scheduler

The design of capacity scheduling algorithm is very similar to fair scheduling. But used of queues instead of pool. Each queue is assigned to an organization and resources are divided among these queues. Scilicet, Capacity scheduling algorithm puts jobs into multiple queues in accordance with the conditions, and allocates certain system capacity for each queue. If a queue has heavy load, it seeks unallocated resources, then makes redundant resources allocated evenly to each job [19], [6]. For maximizing resource utilization, it allows re-allocation of resources of free queue to queues using their full capacity. When jobs arrive in that queue, running tasks are completed and resources are given back to original queue. It also allows priority based scheduling of jobs in an organization queue [20]. The capacity scheduler allows users or organization to simulate a separate MapReduce cluster with FIFO scheduling for each user or organization [21]. Generally, capacity scheduling algorithm addresses the FIFO's disadvantage such as the low utilization rate of resources. The most complex among three schedulers is an important problem in capacity algorithm.

3.4 hybrid scheduler based on dynamic priority

Nguyen et al. [22] propose a Hybrid Scheduler algorithm based on dynamic priority in order to reduce the delay for variable length concurrent jobs, and relax the order of jobs to maintain data locality. Also its provides a user-defined service level value for QoS. This algorithm is designed for data intensive workloads and tries to maintain data locality during job execution [11]. their believes, average response time for the workloads approximately 2.1x faster over the Hadoop Fairs with a standard deviation of 1.4x. it achieves this improved response time by means of relaxing the strict proportional fairness with a simple exponential policy model. This algorithm is a fast and flexible scheduler that improves response time for multi-user Hadoop environments.

3.5 Longest Approximate Time to End (LATE)

Zaharia et al. [23] proposed LATE (Longest Approximate Time to End) scheduler to robustly improve performance by reducing overhead of speculation execution tasks. The technique works for MapReduce in heterogeneous environment. LATE scheduling algorithm tries to improve Hadoop by attempting to find real slow tasks by computing remaining time of all the tasks. it ranks tasks by estimated time remaining and starts a copy of the highest ranked task that has a progress rate lower than the Slow Task Threshold[16 23]. LATE is based on three principles: prioritizing tasks to speculate, selecting fast nodes to run on, and capping speculative tasks to prevent thrashing. the advantage of the LATE algorithm is robustness to node heterogeneity, since only some of the slowest speculative tasks are restarted. This method does not break the synchronization phase between the map and reduce phases, but only takes action on appropriate slow tasks.

3.6 Self-Adaptive MapReduce (SAMR)

LATE scheduling algorithm does not compute the remaining time for tasks correctly, and cannot find real slow tasks in the end. to address this limitation, Chen et al. [24] propose a Self-Adaptive MapReduce scheduling algorithm which is another

approach for heterogeneous environments. The SAMR scheduling algorithm improves MapReduce by saving execution time and system resources. it defined fast nodes and slow nodes to be nodes which can finish a task in a shorter time and longer time than most other nodes. On MapReduce, slow tasks prolong the execution time of an entire job. In heterogeneous clusters, nodes require different times to accomplish the same tasks due to their differences, such as computation capacities, communication, architectures, memory, and power. SAMR scheduling algorithm could be further improved in term of in three aspects. First, it will focus on how to account for data locality when launching backup tasks, because data locality may remarkably accelerate the data load and store. Second, it considering a mechanism to incorporate that tune the parameters should be added. at last, it will be evaluated on various platforms by first evaluated on rented Cloud Computing platform [24]. SAMR decreases the time of the execution up to 25% compared with Hadoop's scheduler and 14% compared with LATE scheduler [25].

3.7 delay scheduling

Zaharia et al. [26] proposed a simple algorithm which named delay scheduling To address the conflict between locality and fairness. In delay scheduling when a node requests a task, if the head-of-line job cannot launch a local task, we skip it and look at subsequent jobs. However, if a job has been skipped long enough, we start allowing it to launch non- local tasks, to avoid starvation [26]. Delay scheduling is a solution that temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When a node requests a task, if the head-of-line job cannot launch a local task, it is skipped and looked at subsequent jobs. However, if a job has been skipped long enough, non-local tasks are allowed to launch to avoid starvation [21].

3.8 Maestro

Ibrahim et al. [27] developed a scheduling algorithm called Maestro to avoid the non-local Map tasks execution problem that relies on replica-aware execution of Map tasks. To address this, Maestro keeps track of the chunks and replica locations, along with the number of other chunks hosted by each node. This way, Maestro can schedule Map tasks with low impact on other nodes' local Map tasks execution by calculating the probabilities of executing all the hosted chunks locally [11]. Maestro keeps track of the chunks locations along with their replicas locations and the number of other chunks hosted by each node. So that it can efficiently schedule the map task on a data local node which causes minimal impacts on other nodes local map tasks executions. It does map task scheduling in two waves. initially, it fills the empty slots of each data node based on the number of hosted map tasks and on the replication scheme for their input data; and second, runtime scheduling takes into account the probability of scheduling a map task on a given machine depending on the replicas of the task's input data [27]. Maestro with use of these two waves, provide a higher locality in the execution of map tasks and more balanced intermediate data distribution for the shuffling phase.

3.9 Combination Re-Execution Scheduling Technology (CREST)

Lei et al. [28] propose a novel approach, CREST (Combination Re-Execution Scheduling Technology), which can achieve the optimal running time for speculative Map tasks and decrease the response time of MapReduce jobs. To mitigate the impact of straggler tasks, it is common to run a speculative copy of the straggler task. The main idea is that re-executing a combination of tasks on a group of computing nodes may progress faster than directly speculating the straggler task on target node, due to data locality. The evaluation conducted demonstrates that CREST can reduce the running time of speculative Map tasks by 70% on the best cases and 50% on average, compared to LATE.

3.10 Context-aware Scheduler

Kumar et al. [29] propose a context-aware scheduler. The proposed algorithm uses the existing heterogeneity of most clusters and the workload mix, proposing optimizations for jobs using the same dataset. Although still in a simulation stage, this approach seeks performance gains by using the best of each node on the cluster. The design is based on two key insights. First, a large percentage of MapReduce jobs are run periodically and roughly have the same characteristics regarding CPU, network, and disk requirements. Second, the nodes in a Hadoop cluster become heterogeneous over time due to failures, when newer nodes replace old ones. The proposed scheduler is designed to tackle this, taking into account job characteristics and the available resources within cluster nodes. The scheduler uses then three steps to accomplish its objective: classify jobs as CPU or I/O bound; classify nodes as Computational or I/O good; and

map the tasks of a job with different demands to the nodes that can fulfill the demands.

3.11 Locality-Aware Reduce Task Scheduler

Hammoud and Sakr [30] present another approach discussing the data locality problem. It deals specifically with Reduce tasks. The Reduce phase scheduling is modified to become aware of partitions, locations, and size, to decrease network traffic. The scheduler, named Locality-Aware Reduce Task Scheduler (LARTS), uses a practical strategy that leverages network locations and sizes of partitions to exploit data locality. LARTS attempts to schedule Reducers as close as possible to their maximum amount of input data and conservatively switches to a relaxation strategy seeking a balance among scheduling delay, scheduling skew, system utilization, and parallelism.

3.12 Center-of-Gravity Reduce Scheduler

Hammoud et al. [31] propose another approach named Center-of-Gravity Reduce Scheduler (CoGRS). The work designs a locality-aware, skew-aware Reduce task scheduler for saving MapReduce network traffic. The proposed scheduler attempts to schedule every Reduce task at its center-of-gravity node deter-mined by the network locations. By scheduling Reducers at their center-of-gravity nodes, they argue for decreased network traffic, which may possibly allow more MapReduce jobs to co-exist on the same system.

3.13 MapReduce with communication overlap

Ahmad et al. [32] proposed MaRCO (MapReduce with communication overlap), which is directed to the overlapping of the Shuffle with the Reduce computation. The original Hadoop data flow was modified allowing the operation of Reduce tasks on partial data. MaRCO breaks Reduce into many smaller invocations on partial data from some map tasks, and a final reducing step re-reduces all the partial reduce outputs to produce the final output.

3.14 COSSH

Rasooli and Down [33] introduce a new scheduling system that called COSHH, which is designed and implemented for Hadoop, it considers heterogeneity at both application and cluster levels. The main approach in COSHH scheduling system is to use system information to make better scheduling decisions, which leads to improving the performance. COSHH consists of two main processes, where each process is triggered by receiving one of these messages. Upon receiving a new job, the scheduler performs the queuing process to store the incoming job in an appropriate queue. Upon receiving a heartbeat message, the scheduler triggers the routing process to assign a job to the current free resource. COSHH is proposed to improve the mean completion time of jobs.

3.15 SARS algorithm

Tang et al. [34] proposed an optimal reduce scheduling policy for reduce tasks start time in Hadoop, which called SARS. This algorithm works by delaying the reduce processes, the reduce tasks are scheduled when part but not all of the map tasks finished. The purpose of the scheduling algorithm SARS is to shorten the copy duration of the reduce process, decrease the task complete time, and save the reduce slots resources. due to The experimental results in [p18], a time when comparing SARS with the FIFO, the reduce completion time is decreased sharply. And it is also proved that the average response time are decreased 11% to 29% when SARS algorithms are applied traditional job scheduling algorithm: FIFO, Fair Scheduler and Capacity Scheduler. the limitation of this algorithm is that only focus on reduce process.

4 DISCUSSION AND ANALYSIS

due to our research and result of many articles [1],[2],[11],[13],[16],[20],[21],[22], we analysis scheduling algorithms and shows the Taxonomy, Idea to implementation, advantages and disadvantage in the form of table. Table 1 shows this. in the taxonomy column, Based on runtime flexibility of their algorithms we classified In tow adaptive and non-adaptive categories. An adaptive scheduling algorithm uses the previous, current and/or future values of parameters to make scheduling decisions. A non-adaptive scheduling algorithm, does not take into consideration the changes taking place in environment and schedules job/tasks as per a predefine policy/order. Generally, these algorithms implemented based on different ideas, some of them have been achieved to its purpose and any other cannot achieve to their objective. In the Idea implementation column listed the proposed of authors of each algorithm. as you can see, in the disadvantage column, some of these algorithm have null value. Because, they can achieved to their proposed and due to result of many articles [22], [27], [28], [30], [31], [33] we believe they don’t have any disadvantage that able

to reduce their abilities and performances. All of these algorithm proposed to address one or two problems and none of them is not suitable to all of our objectives. (see table 1 in the next page)

Table 1: Analysis of scheduling algorithm based of Taxonomy, advantages and disadvantages

Scheduling Algorithm	Taxonomy	Idea to Implementation	Advantages	Dis-advantages
FIFO	Non-adaptive	schedule jobs based on their priorities in first-come first-out.	1. cost of entire cluster scheduling process is less. 2. simple to implement and efficient.	1. designed only for single type of job. 2. Low performance when run multiple types of jobs. 3. poor response times for short jobs compared to large jobs.
Fair Scheduling	adaptive	do a equal distribution of compute resources among the users/jobs in the system.	1. less complex. 2. works well when both small and large clusters. 3. it can provide fast response times for small jobs mixed with larger jobs.	1. does not consider the job weight of each node.
Capacity	adaptive	Maximization the resource utilization and throughput in multi-tenant cluster environment.	1. ensure guaranteed access with the potential to reuse unused capacity and prioritize jobs within queues over large cluster .	1. The most complex among three schedulers.
hybrid scheduler based on dynamic priority	adaptive	designed for data intensive workloads and tries to maintain data locality during job execution	1. is a fast and flexible scheduler. 2. improves response time for multi-user Hadoop environments.	-
LATE	adaptive	Fault Tolerance	1. robustness to node heterogeneity. 2. address the problem of how to robustly perform speculative execution to maximize performance.	1. only takes action on appropriate slow tasks. 2. However it does not compute the remaining time for tasks correctly, and cannot find real slow tasks in the end. 3. poor performance due to the static manner in computing the progress of the tasks.
SAMR	adaptive	To improve MapReduce in terms of saving the time of the execution and the system's resources.	1. decreases the execution time of map reduce job. 2. improve the overall MapReduce performance in the heterogeneous environments.	1. do not consider the data locality management for launching backup tasks.
delay scheduling	adaptive	To address the conflict between locality and fairness.	1. Simplicity of scheduling	1. No particular
Maestro	adaptive	Proposed for map tasks, to improve the overall performance of the MapReduce computation.	1. provide a higher locality in the execution of map tasks. 2. provide more balanced intermediate data distribution for the shuffling phase.	-

CREST	adaptive	re-executing a combination of tasks on a group of computing nodes.	1. decrease the response time of MapReduce jobs. 2. optimal running time for speculative map tasks.	-
context-aware scheduler	adaptive	To optimizations for jobs using the same dataset	1. optimizations for jobs using the same dataset.	-
LARTS	adaptive	decrease network traffic	1. improvement data Locality	-
CoGRS	adaptive	proposed scheduler attempts to schedule every Reduce task at its center-of-gravity node deter-mined by the network locations.	1. decreased network traffic. 2. saving MapReduce network traffic.	-
MaRCO	adaptive	achieve nearly full overlap via the novel idea of including the reduce in the overlap.	1. reduce computation time. 2. improve performance for the important class of shuffle-heavy Map Reductions .	-
COSHH	adaptive	proposed to improve the mean completion time of jobs	1. improve the overall system performance. 2. addresses the fairness and the minimum share requirements.	-
SARS	adaptive	shorten the copy duration of the reduce process, decrease the task complete time, and save the reduce slots resources	1. reduce completion time. 2. decreased the response time.	1. only focus on reduce process.

5 CONLUSION

In this paper, we described the overview of Hadoop MapReduce and their scheduling issues. Then, we explained fifteen popular scheduling algorithms in this field. Then, we Analyzed these algorithm based on taxonomy, Idea to implementation, advantages and disadvantages. we can see that most of these schedulers discussed in this paper, addresses one or more problem. And choice of this scheduler for a particular job is up to the user. Therefore, none of these algorithm cannot plays all of our requires. So, to improve the overall MapReduce performance in the heterogeneous environments we can use of COSHH and SAMR algorithms. to decreased network traffic and saving MapReduce network traffic, CoGRS is the best cases of them. Also, to improvement data Locality LARTS is the best case. if you need a fast and flexible scheduler, hybrid scheduler based on dynamic priority is more better than anyone. I believe it now we need universal scheduling algorithm to get best result from MapReduce model.

6 REFERENCES

[1] A. N Nandakumar and Y. Nandita, " A Survey on Data Mining Algorithms on Apache Hadoop Platform", International Journal of Emerging Technology and Advanced Engineering, Vol. 4, NO. 1, January 2014, pp. 563-566.

[2] Z. Tang, L. Jiang, J. Zhou, K. Li, and K. Li, " A self-adaptive scheduling algorithm for reduce start time ", Future Generation Computer Systems, 2014.

[3] K. Morton, M. Balazinska and D. Grossman, " Paratimer: a progress indicator for MapReduce DAGs", In Proceedings of the 2010 international conference on Management of data, 2010, pp.507–518.

[4] Lu, Wei, et al. "Efficient processing of k nearest neighbor joins using MapReduce ", Proceedings of the VLDB Endowment, Vol. 5, NO. 10, 2012, pp. 1016-1027.

[5] J. Dean and S. Ghemawat, "Mapreduce: simplied data processing on large clusters", in OSDI 2004: Proceedings of 6th Symposium on Operating System Design and Implemention, (New York), ACM Press, 2004, pp. 137–150.

[6] "Hadoop home page." http://hadoop.apache.org/.

[7] White, T., 2012," Hadoop: The Definitive Guide ",ed. Third, Tokyo: Yahoo press.

[8] B. Lublinsky, K.T. Smith, A. Yakubovich, 2013," Professional Hadoop Solutions ",ed. first, John Wiley & Sons, Inc., Indianapolis, Indiana

[9] hfsp

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", In Communications of the ACM, Vol.51, NO. 1, 2008, pp. 107-113.

[11] Polato I, et al. " A comprehensive view of Hadoop research—A systematic literature review ", Journal of Network and Computer Applications (2014), http://dx.doi.org/10.1016/j.jnca.2014.07.022.

[12] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico and P. Michiardi, " Practical size-based scheduling for MapReduce workloads", CoRR, vol. abs/1302.2749.

[13] D. Yoo and K.M Sim, " A comparative review of job scheduling for MapReduce", In Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on IEEE, September 2011, pp. 353-358.

[14] Hadoop, "Hadoop home page." http://hadoop.apache.org/.

[15] Hadoop's Fair Scheduler. https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.

[16] B.P Andrews and A. Binu, " Survey on Job Schedulers in Hadoop Cluster ", IOSR Journal of Computer Engineering, Vol.15, NO. 1, Sep. - Oct. 2013, pp. 46-50.

[17] The Apache Hadoop Project. http://www.hadoop.org.

[18] Y. Tao Y, Q. Zhang, L. Shi and P. Chen, " Job scheduling optimization for multi-user MapReduce clusters ", In: The fourth international symposium on parallel architectures, algorithms and programming. IEEE; 2011. p. 213–17.

[19] J. Chen, D. Wang and W. Zhao, " A Task Scheduling Algorithm for Hadoop Platform ", JOURNAL OF COMPUTERS, VOL. 8, NO. 4, APRIL 2013, pp. 929-936.

[20] N. Tiwari, " Scheduling and Energy Efficiency Improvement Techniques for Hadoop Map-reduce: State of Art and Directions for Future Research (Doctoral dissertation, Indian Institute of Technology, Bombay Mumbai).

[21] A. P. Kulkarni and M. Khandewal, " Survey on Hadoop and Introduction to YARN ", International Journal of Emerging Technology and Advanced Engineering, Vol.4, NO. 5, May 2014, pp. 82-87.

[22] P. Nguyen, T. Simon, M. Halem, D. Chapman and Q. Le, " A hybrid scheduling algorithm for data intensive workloads in aMapReduce environment", In: Proceedings of the 2012 IEEE/ ACM fifth international conference on utility and cloud computing. Washington, DC, USA: IEEE computer society; UCC'12, 2012, p. 161-168.

[23] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz and I. Stoica, "Improving MapReduce performance in heterogeneous environments " In: OSDI 2008: 8th USENIX Symposium on Operating Systems Design and Implementation 2008.

[24] Q. Chen, D. Zhang, M. Guo, Q. Deng Q and S. Guo, " SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment " , In: The 10th international conference on computer and information technology. IEEE; 2010. p. 2736–43.

[25] S. Khalil, S.A. Salem, S. Nassar and E.M. Saad, " Mapreduce Performance in Heterogeneous Environments: A Review ", International Journal of Scientific & Engineering Research, Vol. 4, NO. 4, April - 2013, pp. 410-416.

[26] 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 fifth European conference on computer systems. New York, NY, USA: ACM; 2010, p. 265–278.

[27] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu and S. Wu, " Maestro: replica-aware map scheduling for MapReduce", In: The 12th international symposium on cluster, cloud and grid computing. IEEE/ACM; 2012, p. 435–477.

[28] L. Lei, T. Wo and C. Hu, " CREST: Towards fast speculation of straggler tasks in MapReduce", In: The eighth international conference on e-business engineering. IEEE; 2011, pp. 311-316

[29] K.A Kumar, V.K Konishetty, K. Voruganti and G. Rao, " CASH: context aware scheduler for Hadoop", In: Proceedings of the international conference on advances in computing, communications and informatics. New York, NY, USA: ACM; 2012. p. 52–61.

[30] M. Hammoud and M. Sakr, " Locality-aware reduce task scheduling for MapReduce", In: The third international conference on cloud

computing technology and science. IEEE, 2011, p. 570–576.

[31] M. Hammoud, M. Rehman and M. Sakr, " Center-of-Gravity reduce task scheduling to lower MapReduce network traffic ", In: International conference on cloud computing. IEEE, 2012, p. 49–58.

[32] F. Ahmad, S. Lee, M. Thottethodi and T. Vijaykumar, " MapReduce with communication overlap (MARCO) ", J Parallel Distrib Comput, Vol. 73, NO. 5, 2013, pp. 608–628.

[33] A. Rasooli and D.G. Down, " COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems", Future Generation Computer Systems, 36, 2014, pp. 1-15.

[34] Z. Tang, L. Jiang, J. Zhou, K. Li, and K. Li. "A self-adaptive scheduling algorithm for reduce start time." Future Generation Computer Systems, 2014.