A Comprehensive View of MapReduce Aware Scheduling Algorithms in Cloud Environments

Hadi Yazdanpanah
Computer Student,
Department of Computer,
Iaub, Bushehr, Iran

Amin Shouraki
Computer Student,
Department of Computer,
Iaub, Bushehr, Iran

Abbas Ali Abshirini
Computer Student,
Department of Computer,
Azad University, Dehdasht, Iran

ABSTRACT
Cloud computing has emerged as a model that harnesses massive capacities of data centers to host services in a cost-effective manner. MapReduce has been widely used as a Big Data processing platform, proposed by Google in 2004 and has become a popular parallel computing framework for large-scale data processing since then. It is best suited for embarrassingly parallel and data-intensive tasks. It is designed to read large amount of data stored in a distributed file system such as Google File System (GFS), process the data in parallel, aggregate and store the results back to the distributed file system. Scheduling is one of the most critical aspects of MapReduce. Also three important scheduling issues in MapReduce such as locality, synchronization and fairness exist. This paper tries to illustrate and analyze the overview of thirteen different aware scheduling algorithms with different techniques and approaches for MapReduce in Hadoop and their scheduling issues and problems. At the end, Advantages and disadvantages of these algorithms are identified.

Keywords
Cloud Computing, MapReduce, Scheduling algorithms

1. INTRODUCTION
Nowadays data are becoming larger and larger in every field. Cloud Computing is new style of computing which is getting progress constantly. Also Cloud Computing includes computational and storage services as pay you go model. To provide proficient resources, Cloud computing is been pioneered. Many organizations have their own private cloud, but when there is need for extra resources they go for public cloud where they have been outlaid for their use. In such "pay-per-use", workflow execution cost must be considered during scheduling based on users' QoS constraints. As a popular programming model in cloud-based data processing environment, MapReduce and Hadoop [1] is Apache's open source implementation of the MapReduce framework, are widely used both in industry and academic researches. MapReduce [2] is proposed by Google in 2004 and has become a popular parallel computing framework for large-scale data processing since then. It is best suited for processing parallel and data-intensive tasks. It is designed to read large amount of data stored in a distributed file system such as Google File System (GFS) [3], process the data in parallel, aggregate and store the results back to the distributed file system. In a typical MapReduce job, the master divides the input files into multiple map tasks, and then schedules both map tasks and reduce tasks to worker nodes in a cluster to achieve parallel processing. The two major performance metrics in MapReduce are job execution time and cluster throughput.

Many cloud applications assume a homogeneous environment. For example, Hadoop [4] assumes that all nodes participating in the cluster have the same processing power. A Hadoop job is consists of a number of tasks that run on nodes concurrently. When Hadoop schedules a task of a job, it assumes that it takes about the same time to process a task regardless of where it runs. It considers network connectivity by giving preference to tasks that access local data over these access remote data, but does not consider the difference of computing capability of nodes. Further, in a heterogeneous environment, some tasks run faster on a particular node than others. In addition, it is not straight forward to guarantee fairness among multiple jobs in heterogeneous environments. The aim of task scheduling in Hadoop is to move computation towards data. Scheduling is one of the important factors in MapRduce. In order to achieve good performance a MapReduce scheduler must avoid unnecessary data transmission. The JobTracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack. A TaskTracker is a node in the cluster that accepts tasks Map, Reduce and Shuffle operations from a JobTracker. In MapReduce framework, each TaskTracker sends frequent heartbeats to the job tracker which contains the number of free map and reduce slots on that slave node. The JobTracker then assigns a task to the TaskTracker having free slots according to the configured scheduling policy.

The MapReduce scheduling algorithms mainly include FIFO (First Input First Output), LATE (Longest Approximate Time to End), Fair Scheduler and Capacity Scheduler. FIFO [4] 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. LATE scheduler [5] try to improve performance by reducing overhead of speculation execution tasks. The fair scheduler [6] was developed by Facebook. The goal of the fair scheduler is to provide fast response times for small jobs and QoS for production jobs. The fair scheduler has three basic concepts: 1) Jobs are grouped into pools. 2) Each pool is assigned a guaranteed minimum share. 3) Excess capacity is split between jobs. By default, jobs that are uncategorized go into a default pool. Pools have to specify the minimum number of map slots, reduce slots, and a limit on the number of running jobs. The capacity scheduler [7] was developed by Yahoo. The capacity scheduler supports several features that are similar to the fair scheduler. 1) Queues are allocated a fraction of the total resource capacity. 2) Free resources are allocated to queues beyond their total capacity. 3) Within a queue a job with a high level of priority has access to the queue's resources.

The rest of the paper is organized as follows: Section 2 provides a background on Hadoop and MapReduce Mechanisms. Sections 3 introduce the MapReduce Aware Scheduling Algorithms. Section 4 describe analyze and consider advantage and disadvantage in the form of table. In Section 5 conclude the paper.

2. BACKGROUND

This section briefly describes how a Hadoop and MapReduce work.

2.1 Hadoop

Hadoop is Java base open source implementation of the MapReduce platform and distributed file system. Hadoop runs over a distributed file system called Hadoop Distributed File System (HDFS) which has the same architecture as Google File System [8]. HDFS has master/slave architecture. HDFS consists of one the master server, called NameNode and there are a number of slaves, called DataNodes. NameNode which controls several DataNodes, and the DataNodes store actual data. Namenode supervises metadata such as information of directories, access log from users, detail of data location, and system logs. Datanode keeps data in Blocks. A Block is a basic unit for data storing in HDFS. Figure 1 briefly describes the Hadoop Architecture.

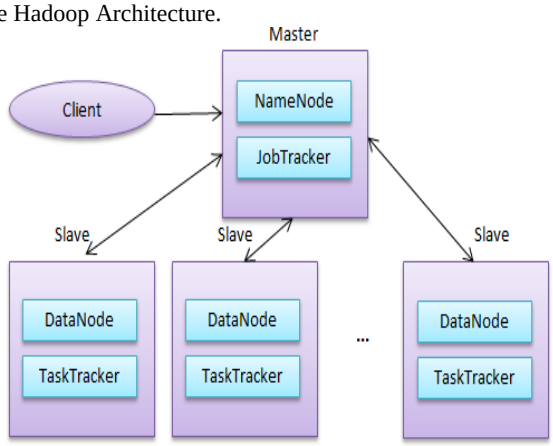

Fig 1. Hadoop Architecture.

2.2 MapReduce

MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. MapReduce programs are written in a particular style influenced by functional programming constructs, specifically idioms for processing lists of data. As a distributed computing framework on commercial computer, one of the MapReduce most significant advantages is that it provides an abstraction that hides many system level details from programmer. It processes data by dividing the progress into two phases: Map and Reduce. Each Map function takes a split file as its input data, which locates in the distributed file system and contains the key/value data. The split file can be co-location with the Map function or not. If the split file and the Map function don't in the same node, then the system will transfer the split file to the Map function. The Reduce function is applied to all values that associated with the same intermediate key and generates output key/value pairs as the final result. The MapReduce framework has master/slave architecture. It has a single master server or JobTracker and several slave servers or TaskTrackers, one per node in the cluster. The JobTracker is the point of interaction between users and the framework. Users submit map/reduce jobs to the JobTracker, which puts them in a queue of pending jobs and executes them on a first-come/first-served basis. The JobTracker manages the assignment of map and reduce tasks to the TaskTrackers. The TaskTrackers execute tasks upon instruction from the JobTracker and also handle data motion between the map and reduce phases. Figure 2 briefly describes how the MapReduce model works.

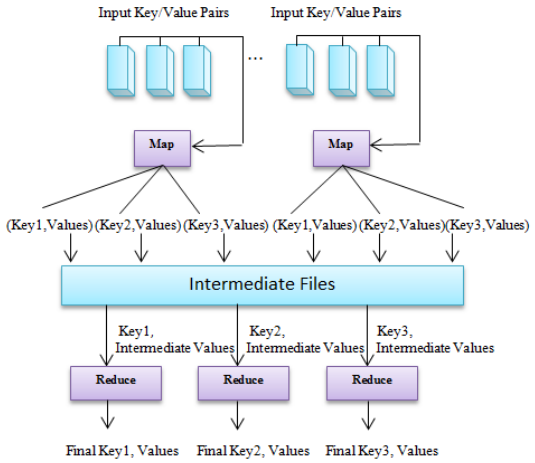

Fig 2. MapReduce Model

3. MAPREDUCE AWARE SCHEDULING ALGORITHMS

The Scheduling is one of the most critical aspects of MapReduce. There are many Aware Scheduling algorithms to address these issues with different techniques and approaches.

3.1 Center-of-Gravity Reduce Scheduling

In [9], the authors propose another approach named Center-of-Gravity Reduce Scheduler (COGRS). COGRS is a locality-aware skew-aware reduce task scheduler for saving MapReduce network traffic. This scheduler tries to schedule every reduce task at its center of- gravity node determined by the network locations of that task's feeding nodes and the skew in the sizes of that task's partitions. By scheduling reducers at their center-of-gravity nodes, they argue for reduced network traffic which can possibly allow more MapReduce jobs to co-exist on the same system.

3.2 Context Aware Scheduling

In [10], the authors proposed a model for smarter services that combines techniques of context awareness and adaptive job scheduling. The proposed model works by adjusting the priorities of the server-based jobs in response or pro-actively to variations of the end-user local context. It aims at providing delay-tolerant job execution required in mobile environment, while reducing the resource wastage by properly scheduling jobs in the Cloud. That is, by being able to adjust the priority of incoming jobs in relation to variations of local contexts, the system improves the overall resource utilization delivering better performance and, consequently, better quality of service (QoS).

Kumar et al. [11] propose a context-aware scheduler (CASH); the proposed algorithm uses the existing heterogeneity of most clusters and the workload mix, proposing optimizations for jobs using the same dataset. This scheduler increases the performance in heterogeneous Hadoop clusters. Although still in a simulation stage, this approach seeks performance gains by using the best of each node on the cluster. The proposed algorithm is based on two key schemas. First, most 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; these schedulers classifies the 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. By implementing CASH, the performance of the heterogeneous cluster and the aggregate execution times of the jobs can be improved.

3.3 Distribution Aware Scheduling

In [12], the authors propose a scheduling method to improve the data locality of MapReduce. After receiving a request from a node, the method selects a task from the first level followed by the second and the third level of the node. Then, it checks whether the task is the only one on the first level of the node to issue a request. If so, the method skips the selected task, and selects another task for the node issuing a request. Otherwise, the method schedules the selected task to the node. In [13], the authors offered a data distribution aware task scheduling strategy for MapReduce system. Their strategy has two main phases: In the initialization phase, statistics number of copies of data processed by each map task. At the same time, statistics the number of localizable tasks for each worker; in the scheduling phase, according to the information above, calculating the scheduling priorities for each task and each works that requesting task, and scheduling the task to works based on this priority. This strategy regarding the distribution of data, schedules map tasks on the nodes that most likely contain relevant data, and reduces network overhead and improves the performance of system.

3.4 Energy Aware Scheduling

In [14], the authors propose a greedy algorithm, called Energy-aware MapReduce Scheduling Algorithm (EMRSA), this model is a framework for improving the energy efficiency of MapReduce applications, while satisfying the service level agreement (SLA). It can finds the assignments of map and reduce tasks to the Machine slots for minimizing the energy consumption when executing the application and that schedules the individual tasks of a MapReduce application for energy efficiency while meeting the application deadline. Tasks can be run in parallel, but no reduce task can be started until all map tasks for the application are completed. Important issue in MapReduce Scheduling Algorithms is that the user only specifies the deadline for the job without specifying a deadline for the map phase. However, since the reduce tasks are dependent on the map tasks, the data center has to determine a reasonable deadline for the map tasks with respect to the availability of the map slots in the data center in order to utilize its resources efficiently. The proposed algorithm finds the assignments of map tasks to the map slots satisfying the determined map deadline. Finally, EMRSA can finds the assignments of reduce tasks to the reduce slots satisfying the deadline, where all the reduce tasks start after the map deadline. Chen et al. [15] proposed a method for MapReduce jobs without relying on replication by divides the jobs into time sensitive and less time-sensitive jobs, where the former are assigned to a small pool of dedicated nodes, and the latter can run on the rest of the cluster. Also it is abled reduce the energy consumption. Land and Patel [16] proposed a method in MapReduce clusters for energy management by powering down all nodes in the cluster during a low Utilization period. EMRSA is able to find job schedules consuming 40% less energy on average than the schedules obtained by a common practice scheduler that minimizes the makespan.

3.5 Job Aware Scheduling

Nanduri et al. [17], propose a job-aware scheduling algorithm for reduce the jobs execution time in MapReduce. From the list of available pending tasks, the scheduler selects the one that is most compatible with the tasks already running on that node. In this model, This scheduler employs an event capturing mechanism on the TaskTrackers [18] which listens to events related to memory intensive, CPU intensive, disk intensive, and network intensive to monitor resource usage characteristics of that particular task. In [19], the authors propose a novel job aware scheduling algorithm that overcomes limitations such as limited utilization of computing resources, limited applicability towards heterogeneous cluster, random scheduling of non-local map tasks, and negligence of small jobs in scheduling. The proposed algorithm schedules jobs based on one of the following three criteria: job execution time, earliest deadline first, and workload of the job. Minimum execution time is selected. Hence, the average waiting time of jobs decreases considerably. The earliest deadline first criterion is appropriate when jobs have a strict deadline. The scheduling of non-local map tasks of jobs based on job execution time and earliest deadline first reduces the average waiting time. The scheduling of non-local map tasks of jobs considering workload of the job increases the resources utilization of the cluster. The proposed algorithm increases the resource utilization and reduces the average waiting time compared to existing Matchmaking scheduling algorithm [20].

3.6 Load Aware Scheduling

In [21], the authors offered load aware scheduler for MapReduce framework in heterogeneous cloud environments, and abbreviated it as LA scheduler. This scheduler improves the overall performance of Hadoop clusters and be able to reduce up to 20% in average response time by avoiding unnecessary speculative tasks.

3.7 Locality Aware Scheduling

In [22], the authors propose locality-aware scheduling algorithm (LaSA) to enhance data locality assignment in Hadoop scheduler and increases performance of data-intensive computing application in Hadoop MapReduce architecture. The aim of LaSA is to achieve locality-aware resource assignment in order to reduce the bottleneck of network transmission by following the weight of data interference. LaSA algorithm introduces a concept of weight of data interference in MapReduce framework and locality-aware scheduler in JobTracker. LaSA can calculate each node's weight of data interference and pick up a node with smallest weight and data locality to execute the task. LaSA include data nodes, the weight of data, the replica number of input data and the number of map slots on each node. The resource assignment depends on the weight of data interference on each node. LaSA algorithm focuses on the "rare" resource assignment. Rare resource is the data node with some data which is relative scarce. Because, the most part of data nodes which has requirement data is occupied by high priority task. If a node contains rare resource, the node's weight of data interference becomes large to keep the node from task assignment. LaSA algorithm is implemented in JobTracker. JobTracker before the task assignment, calculate the weight of data interference on each node with free slots. JobTracker picks up a node with the smallest weight of data interference and assigns the task to one's TaskTracker. LaSA calculate the weight of data interference to avoid rare resource to allocate in an easy way and introduce the concept of weight of data interference to enhance the data locality in MapReduce framework. LaSA, consider all factors that affect the data locality of a JobTracker. The job scheduler improves the data locality challenges in original MapReduce framework. The LaSA is using weight of data interference concept to arrange the resource assignment to avoid required data missing.

3.8 Network Aware Scheduling

In [23], the authors propose the method to make Hadoop scheduler aware of network topology is to extend the rack aware feature of the existing Hadoop scheduler to provide one more level of caching. An administrator controlled script will hold the information about which cluster the TaskTracker is associated with. When head of the queue task doesn't find a compute node with data then scheduling of the task is delayed for a specified duration of time. If any of the compute nodes become free with a data split corresponding the job being processed then scheduler assigns a map task to requesting TaskTracker. Duration for which a head of the queue map task is to be delayed is based on the average length of the map tasks for a job hence requires careful tuning. One JobTracker will manage the scheduling over all the clusters. TaskTrackers from different clusters request for map tasks as and when they get map slots freed. JobTracker uses the cluster awareness to schedule tasks on these TaskTrackers thereby improving the data locality. By increasing bandwidth between the clusters overall execution time decreases.

Network awareness is applied to nonlocal map tasks which require fetching data from some other data nodes. Network aware Hadoop minimizes the data movement from one cluster to another while executing map task by adding cluster level locality. Delay scheduling [8] can optimize the data locality and ensures that before the task is scheduled on a TaskTracker which does not have the data to process will be skipped for configured amount of time. If any of the TaskTracker becomes free in that duration which as the data to process then, task is scheduled on the second TaskTracker. For data intensive applications, data split movement takes more time than processing of the data split. Network awareness coupled with delay scheduling could be used to minimize the transfer of the data between the clouds and to improve Performance MapReduce.

3.9 Power Aware Scheduling

In [24], the authors proposed a power aware scheduling algorithm for MapReduce jobs in heterogeneous cloud resources in order to energy saving. This scheduler considers users' SLAs (Service Level Agreements). The proposed framework uses information about intermediate key distribution to select appropriate processors for map and reduce tasks. The slack times of map and reduce tasks are used in power reduction of CPUs. And, it can reduce power consumption of disk storage by decreasing disk access speed not to miss the required time. Since MapReduce framework is generally for data-intensive cloud computing, they considered energy saving both in processing elements and in disk storages.

3.10 Replica Aware Scheduling

In [25], the authors propose a method namely, Replica-aware Scheduling (Maestro) for map task scheduling mechanism to improve issue of huge amount of network traffic caused by map tasks execution on remote data in Hadoop. Furthermore, 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 [26]. 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. Maestro schedules the map tasks considering chunk locality and node availability. The scheduling of Maestro is in two waves: first wave scheduler and run time scheduler. The first wave scheduler is responsible for filling the empty slots of each data node based on the number of hosted map tasks and on the replication scheme for their input data. 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. These two waves lead to a higher locality in the execution of map tasks and to a more balanced intermediate data distribution for the shuffling phase. Maestro shows 95% improvement in speculative execution of data local map tasks and 34% improvement in execution time.

3.11 Resource Aware Scheduling

In [27], the authors offer a novel resource management and job scheduling method for MapReduce namely, Resource-aware Adaptive Scheduler (RAS). The aims of this method maximize the utilization of system resources and to meet the users' job completion time. RAS for achieving better utilization of resources and improves application performance, it extends task slot to job slot and leverages resource profiling information. RAS seeks to meet soft-deadlines via a utility-based approach and adapts to changes in resource demand by dynamically allocating resources to jobs. The scheduler tries to differentiate between map and reduce tasks when making resource-aware scheduling decisions. In [28], the authors proposed two resource-aware scheduling mechanisms to minimize competition on machines resources: Dynamic free slot advertisement mechanism and Free slot priorities/filtering mechanism. In Free slot priorities/filtering mechanism, cluster administrators retain the fixed maximum number of compute slots per node at configuration time. As TaskTracker slots become free, they are buffered for some small time period and advertised in a block. TaskTracker slots with higher resource availability are presented first for scheduling tasks on. Instead of scheduling a task onto the next available free slot, job response time would improve by scheduling it onto a resource-rich machine, even if such a node takes a longer time to become available.

3.12 TaskTracker Aware Scheduling

In [29], TaskTracker aware scheduling is propose for users to configure a maximum load per TaskTracker in the Job Configuration itself. The algorithm will not allow a task to run and fail if the load of the TaskTracker reaches its threshold for the job. Also this scheduler allows the users to select the TaskTracker's per Job in the Job configuration. The proposed scheduler schedules the jobs according to the current status of the TaskTrackers. So the scheduler is named accordingly. The proposed system divided into two components, the core scheduler module which will handle the actual scheduling part and a preprocessing module. When a Heartbeat is received from a TaskTracker, the TaskTracker information and List of scheduled Jobs should hand over to the preprocessor. The preprocessing module first compare the hostname of the TaskTracker against the list of TaskTrackers specified for the Job. If this check succeeds then it will compute the number of tasks currently running for the Job in the TaskTracker. If the number of currently running tasks is less than the number specified in the Job Configuration, then the Job object and TaskTracker information is hand over to the Core Scheduler module. The Core scheduler is a modified Fair Scheduler with a priority enhanced algorithm.

3.13 Usage Aware Scheduling

Traditional MapReduce schedulers usually didn't detect slow tasks. To solve this problem, inspiration from the ideas of both the Fair scheduler and LATE scheduler, in [30] the

authors presents a usage aware MapReduce scheduler to deal with the system heterogeneity by including task execution time in scheduling. The usage aware scheduler is able to reduce the overall completion time of MapReduce applications and being able to improve the overall performance of Hadoop clusters. The authors assume that the cluster consists of fast node, normal node and slow node. The proposed scheduler can reduce the overall execution time in heterogeneous environments and able each task's execution to determine the node capability. This algorithm uses a more accurate method to speculate straggler tasks and allocate the tasks to the node with better performance to reduce the overall system response time. A general principle is to let fast nodes accomplish more tasks. The execution of a reduce task can be divided into three phases as the follows.

- The copy phase, where the task fetches the outputs from map tasks.
- The sort phase, where outputs from map are sorted by key.
- The reduce phase, where a user-defined function is applied to the list derived from the sort phase.

Nodes communicate to synchronize with each other in copy phase and sort phase, while map tasks are executed independently. The proposed algorithm basically focuses on task assignment; it can be easily incorporated into the Fair scheduler as in the job selection phase. By using the usage-aware scheduler, 10% to 30% of execution time reduction can be expected in heterogeneous environment.

4. COMPARISON

Advantages and disadvantages of MapReduce scheduling methods are expressed in Tables 1. In a heterogeneous environment where each node has different computing power the heuristic method is not well suited. Sweet spot of a program is the spot at which early shuffle is triggered and provides the best performance for the program. But in COGRS, the sweet spot is determined statically, which is the disadvantage of these schedulers. In the disadvantage column, some of these algorithms have null value. Because, they can achieved to their proposed and due to result of many articles we believe they don't have any disadvantage that able to reduce their abilities and performances. All of these algorithms proposed to have some advantages and disadvantages.

Table 1. Comparison of different algorithms.

Algorithm	Advantages	Disadvantages
Center-of-Gravity Reduce	Decreased network traffic. Reduce job runtime.	Static sweet Spot determination.
Context Aware	Optimizations for jobs using the same dataset. Performance of the heterogeneous cluster and the aggregate execution times of the jobs can be improved.	-
Distribution Aware	Reduce network overhead. Improve system efficiency.	-
Energy Aware	Minimize the energy consumption. Minimizing the makespan.	-
Job Aware	Reduce runtime. Maximize the utilization of nodes resources. Ability to plug into FAIR and Capacity schedulers. Ability to implement in any distributed environment.	
Load Aware	Reduce response Time. Increase of cluster Utilization.	Ignore data locality for launching backup tasks.
Locality Aware	Reduce network traffic. Increase of performance. Avoid data missing.	-
Network Aware	Decrease execution time in FIFO and FAIR schedulers.	-
Power Aware	Save energy consumption. Consider Users' SLA.	-
Replica Aware	Reduce network traffic. Reduce runtime Provide a higher locality in the execution of map tasks.	-
Resource Aware	Reduce contention for CPU resources and I/O on the worker machines. Increase the performance of Cluster.	-
TaskTracker Aware	More control to the users for Job execution. Improve performance.	-
Usage Aware	Be able to reduce the overall completion time.	-

5. CONCLUSION

This paper attempted to comparison and analyzed thirteen different MapReduce Aware scheduling algorithms. Hadoop default scheduler takes care of only homogeneous clusters. Resource aware scheduling considers three resource capacities: CPU, memory and I/O. It can be extended easily to incorporate network infrastructure bandwidth and storage capacity of the TaskTrackers. Distribution aware scheduling can be used for Reduce network overhead and Improve system efficiency. For minimize the energy consumption can used Energy Aware scheduling. Scheduling algorithms above be able and try to increase and improve performance and utilization. Also For improvement data Locality and decreasing network traffic, Locality Aware scheduling is the good case.

6. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, "A View of Cloud Computing ", Comm. Of the ACM, Vol. 53, No. 4, April 2010, pp. 50-58.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters", In Proc. of 5th Symposium on Operating Systems Design and Implementation, 2008, pp. 137-150.

[3] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google File System", In ACM Symposium on Operating Systems Principles (SOSP), 2003.

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

[5] 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.

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

[7] 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.

[8] 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, pp. 265–278.

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

[10] M. D. Assuncao, M. A. S. Netto, F. Koch and S. Bianchi, "Context-aware Job Scheduling for Cloud Computing Environments", IEEE/ACM Fifth International Conference on Utility and Cloud Computing, 2012, pp. 255-262.

[11] 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, pp. 52–61.

[12] X. Zhang and Y. Ding, "A Distribution Aware Scheduling Method in MapReduce", IEEE Symposium on Electrical & Electronics Engineering (EEESYM), 2012, pp. 128-131.

[13] L. Guo, H. Sun et al., "A Data Distribution Aware Task Scheduling Strategy for MapReduce System", Cloud Computing, 2009, pp. 694-699.

[14] L. Mashayekhy, M. Movahed Nejad, D. Grosu, D. Lu and W. Shi, "Energy-aware Scheduling of MapReduce Jobs", IEEE International Congress on Big Data, 2014, pp. 32-39.

[15] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis", in Proc. of the 7th ACM European Conf. on Computer Systems, 2012, pp. 43–56.

[16] W. Lang and J. M. Patel, "Energy management for MapReduce Clusters", Proc. of the VLDB Endowment, vol. 3, no. 1-2, 2010, pp. 129–139.

[17] R. Nanduri, N. Maheshwari, R. Raja and V. Varma , "Job Aware Scheduling Algorithm for MapReduce Framework", 3rd IEEE International Conference on Cloud Computing Technology and Science, 2011, pp. 724-729.

[18] JobTracker Architecture, Available: http://hadoop.apache.org/common/docs/current/mapred_tutorial.html.

[19] S. Pati and M. A. Mehta, "Job Aware Scheduling in Hadoop for Heterogeneous Cluster", IEEE International Advance Computing Conference (IACC), 2015, pp. 778-783.

[20] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new MapReduce scheduling technique", IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), December 2011, pp. 40-47.

[21] H. H. You, C. C. Yang et al., "A load-aware scheduler for MapReduce framework in heterogeneous cloud environments", Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 127-132.

[22] T. Yi Chen, H. Wen Wei, M. Feng Wei, Y. Jie Chen, T. sheng Hsu and W. Kuan Shih, "LaSA: A Locality-aware Scheduling Algorithm for Hadoop-MapReduce Resource Assignment", International Conference on Collaboration Technologies and Systems (CTS), 2013, pp. 342-346.

[23] P. Kondikoppa , C. H. Chiu, C. Cui, L. Xue and S. J. Park, "Network-Aware Scheduling of MapReduce Framework on Distributed Clusters over High Speed Networks", Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit, San Jose, CA, USA, September 21, 2012.

[24] Y. Li, H. Zhang et al., "A Power-Aware Scheduling of MapReduce Applications in the Cloud", Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference, 2011, pp. 613-620.

[25] 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, pp. 435– 477.

[26] I. Polato, R. Ré, A. Goldman and F. Kon, "A comprehensive view of Hadoop research—A systematic literature review", Journal of Network and Computer Applications (2014), Volume 46, November 2014, pp. 1-25.

[27] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres and E. Ayguad, "Resource-aware adaptive scheduling for mapreduce clusters", Middleware 2011, 2011, pp. 187-207.

[28] M. Yong, N. Garegrat and S. Mohan: "Towards a Resource Aware Scheduler in Hadoop", in Proc. ICWS, 2009, pp. 102-109.

[29] J. S Manjaly and V. S Chooralil, "TaskTracker Aware Scheduling for Hadoop MapReduce", Third International Conference on Advances in Computing and Communications, 2013, pp. 278-281.

[30] J. H. Hsiao and S. J. Kao, "A Usage-Aware Scheduler for Improving MapReduce Performance in Heterogeneous Environments", International Conference on Information Science, Electronics and Electrical Engineering (ISEEE), Vol.3, 2014, pp. 1648- 1652.