Job Scheduling Algorithms in Big Data

Kunal Dhande
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60619, US
kdhande@hawk.iit.edu

Mohit Kulpe
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60619, US
mkulpe@hawk.iit.edu

***Abstract*— Hadoop framework has been widely used to process large-scale datasets on computing clusters. In Hadoop, MapReduce framework is a programming model which processes terabytes of data in very less time. This framework uses a task scheduling algorithms to schedule task efficiently. Scheduling of jobs in parallel across nodes is a major concern in distributed file system. A large number of policies, which can determine best structures of task scheduling algorithms, have been explored so far. These policies have significant value for optimizing system efficiency. The objective of all these approaches are maximizing system throughput with assigning a task to a suitable processor, maximizing resource utilization, and minimizing execution time. The objective of the research is to study MapReduce and analyze different scheduling algorithms that can be used to achieve better performance in scheduling.**

Keywords— Big data, Hadoop, MapReduce, Job Scheduling, Classification, Distributed data processing, Algorithms.

I. Introduction

Due to the development of new technologies the amount of data produced is growing rapidly every year. Big data is a collection of large datasets that cannot be processed using traditional computing techniques. Processing large-scale datasets has become an increasingly important and challenging problem as the amount of data created by healthcare industry, scientific research, social networking websites etc., explodes. This has certain challenges, it is very difficult to transfer such a huge data to the computing node and very high speed networks will be needed. Also, the cost required to transmission of data is also high. The era of big data requires new ways to store, manage, access and process the colossal amount of available data. Since parallel processing is scalable and can gain performance improvement by several orders of magnitude, it is generally accepted as a must for data-intensive applications in the big data era.

For processing and analysis of datasets many tools are available and the most popular and widely used is Apache Hadoop. Hadoop can handle all types of data such as structured, unstructured, pictures, videos etc. Hadoop supports redundancy, scalability, parallel processing, and distributed architecture. Due to such simplicity of the programming model and the run-time tolerance for node failures, As a result Hadoop clusters are getting popular, and therefore there is need for more efficient job scheduling algorithms to process data more quickly.

In general, distributed computing is a field of computer science that involves multiple computers, located remotely from each other. Each computer has a common shared role in a computation problem and coordinates their actions by message passing. Scheduling problem is also faced in other computing systems. The work in addresses scheduling in general-purpose distributed computing environment. The work in presents a survey of hadoop scheduling in multiprocessor distributed systems.

II. Background : Hadoop

Hadoop is an Apache open source framework that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. The core of Hadoop consists of a storage part, Hadoop Distributed File System (HDFS) and a processing part called MapReduce.

A. Hadoop Architecture

HDFS has a master-slave architecture. The master process, also called as NameNode, manages the global namespace and controls the operations on files. Each slave process. also called as DataNode, stores the files in form of data blocks and performs operations as instructed by the NameNode.

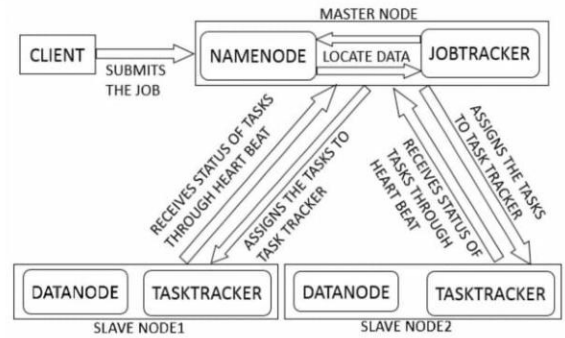

Fig 1: Hadoop Architecture

The NameNode manages the data replication and placement for fault-tolerance, performance and reliability. The NameNode splits and stores files in 64MB or 128MB data blocks across the DataNodes. Usually by default 3 replicas of each data block are stored in the HDFS. Failure detection mechanism is implemented in form of regular heartbeats from DataNodes to NameNode. If there is no heartbeat from a DataNode for long time it is marked as failed and can not be used for further operations.

B. *Working of Hadoop*

In MapReduce, a job consists of an input data, a MapReduce, and configuration properties. A task is an independent sub component of a job, performing either Map or Reduce processing. The JobTracker is the one which receives the job from the user and divides the job into Map tasks and Reduce tasks in the context of Hadoop-MapReduce the job refers to the top level unit of work for a MapReduce system. These tasks are then assigned to the TaskTracker. The JobTracker keeps track of all the tasks by communicating with TaskTrackers and monitoring them. The JobTracker keeps an eye on each TaskTracker by the Heartbeat messages. Heartbeat messages tell the JobTracker that a TaskTracker is whether alive or not. Fig 1 explains the working of Hadoop.

After completion of the entire job, JobTracker informs the status of job to the user. The numbers of MapReduce slots are predefined on each TaskTracker. These numbers determine how many Map and Reduce tasks a TaskTracker can run at a time. Hadoop jobs, submitted by different users, share the cluster resources. Scheduling is done for sharing of resources and to determine when a job can execute its tasks on the nodes.

III. *Issues in Hadoop/MapReduce Scheduling*

For managing multiple Map and Reduce tasks on multiple nodes of Hadoop shared cluster, an efficient scheduling algorithm is required. Scheduling algorithms are designed for achieving proper utilization of resources. The performance of scheduling policies may get affected by the issues such as locality, synchronization and fairness. These issues are discussed in this subsection.

1) *Locality*

Locality is the distance between input data node and the task node. If the input data is nearer to the computation node, then data transfer time is less. Many a times, it is not possible to achieve node locality; in such cases closest node is selected. Most of the scheduling policies try to assign tasks which are nearer to the input data node to lower down the communication cost and to save the network cost.

2) *Synchronization*

Synchronization is the process of transferring intermediate outputs of the Map processes as the input of the Reduce processes. The Reduce phase can be started only after completion of Map phase. As Reduce tasks are dependent on Map tasks, if one of nodes slows down, it will affect the overall performance of the entire process.

This synchronization problem arises in heterogeneous environment, because each node in the cluster has different computation capability and network bandwidth. This causes degradation in the cluster performance.

3) *Fairness*

Fairness refers to how fair a scheduling algorithm is for dividing the resources among users. A MapReduce job with heavy workload may use/utilize the entire shared cluster. As a result, the short computation jobs may not have desired response time. Therefore, workload should be fairly distributed among the jobs, sharing the cluster. Fairness should deal with locality and dependency between the Map and Reduce phases.

If each MapReduce job has roughly an equal share of the nodes and the input files are distributed, then few Map tasks need to load data from network. This results in degradation of throughput and response time. Sometimes synchronization overhead leads to the fairness issues i.e. when Reduce tasks must wait for the completion of Map tasks, starvation of other jobs may occur.

IV. Job Schedulers in Hadoop

This section provides classification of Hadoop schedulers and discusses various Hadoop schedulers.

A. Classification of Hadoop Scheduler

Classification of schedulers can be done based on following parameters: time, priority, strategy, environment, and resource awareness.

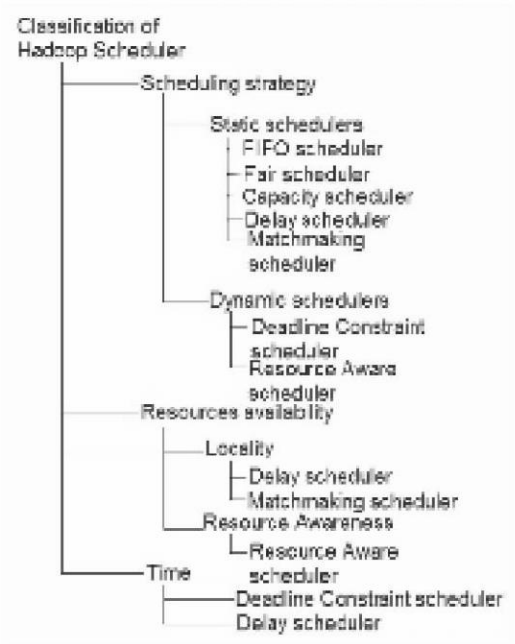

Fig 2. Classification of Hadoop Scheduler

In this section, Classification of Hadoop schedulers is done based on scheduling strategy, resource availability, and time.

1) Based on scheduling strategy (Static/Dynamic)

In Static scheduling, the allocation of jobs to processors is done before program execution begins. Information regarding job execution time and processing resources is known at compile time. The goal of static scheduling is to minimize the overall execution time of current programs. FIFO scheduler, Fair scheduler, Capacity scheduler, Delay scheduler and Map Matching scheduler are examples of static schedulers

In Dynamic scheduling, allocation of jobs to the processors is done during execution time. A little prior knowledge is known about the resource needs of a job. It is also unknown in what environment the job will execute during its lifetime. Decision is made when a job begins its execution in the dynamic environment of the system. Deadline constraint scheduler and Resource aware scheduler comes under dynamic scheduling algorithms.

2) Based on Resources availability

The scheduling is done based on resource requirements of a job. This scheduling is for improving the resource utilization and job performance. The resources can be CPU time, disk storage, memory, etc.

3) Based on Time

In time constrained scheduling, the scheduling of jobs is done based on a deadline, i.e. whether the job can complete its execution within the specified time or not. Deadline and Delay schedulers comes under time constrained scheduling.

B. Description of Scheduler

As Hadoop jobs should share the cluster resources a scheduling policy is used to decide when and where a job is to be executed. The objectives of scheduling are to minimize the completion time, maximize throughput, minimize overhead, and balance available resources of a parallel application by properly allocating the jobs to the processors.

Schedulers in Hadoop are:

1) FIFO Scheduler

The default scheduling policy of Hadoop is First in first out (FIFO). In this scheduler, the jobs that are submitted earlier gets preference over jobs submitted later. Whenever the job comes, the JobTracker pulls oldest job first from the Job queue. It does not consider priority or size of the Job. The drawback of this policy is that the strict FIFO job order reduces data locality, and only after completion of the previous Job, next jobs in the job queue will be assigned to nodes. This scheduler is mostly used when the execution order of job is not important.

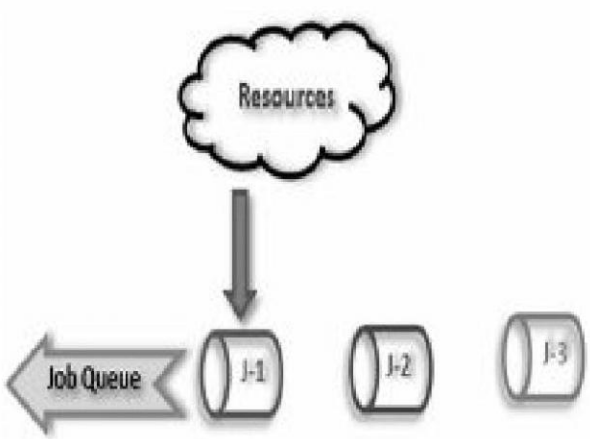

Fig 3: FIFO Scheduler

Working of FIFO Scheduler:

Consider above FIFO Scheduler, In above figure there is resource and Job queue with three jobs.

i. Here job1 (J-1) is at first position in job queue so initially resources will get allocated to job 1 and job1 will be executed first.
ii. After successful execution of job1 resources will be deallocated from job1.
iii. Now these unallocated resources will be assigned to next job i.e. job2 (J-2) and then job2 will be executed.
iv. Finally after execution of job2 resources will be allocated to job3 (J-3) and job3 execution will start.

So this is the process how jobs get executed in FIFO Scheduler. It is sequential execution process no matter which job has what priority. No previous job will be executed before execution of first job.

2) FAIR schedule

This mechanism was developed at Facebook to manage access the Hadoop cluster. The objective of this scheduler is to assign each user a fair share of the cluster capacity over a time. Users may assign jobs to pools, with each pool allocated a guaranteed minimum number of Map and Reduce slots. Free slots in ineffective pools may be allocated to new pools . Its preemptive technique that means the scheduler will kill tasks in pools running over capacity in order to give the slots to the pool running under capacity. Priority criteria are also assigned to various pools. Here tasks are scheduled in interleaved manner based on priority.

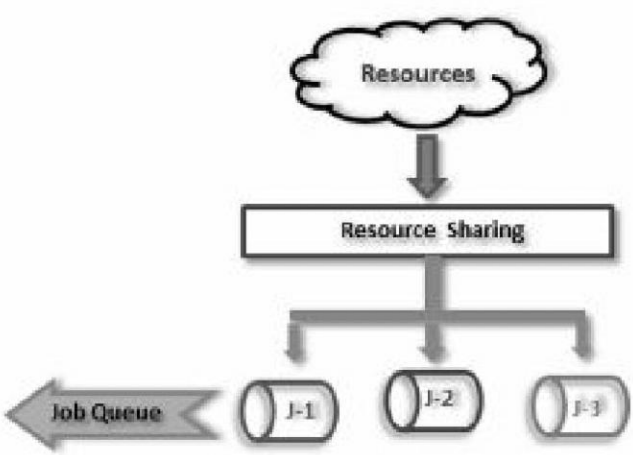

Fig 4: Fair Scheduler

Working of Fair Scheduler:

Consider above Fair Scheduler, In above figure there is resource Job queue with three jobs and Resource sharing block.

i. Here job 1 (J-1) is at first position in job queue so initially resources will get allocated to job 1 like FIFO Scheduler but allocation is only for particular predefined time slice and this allocation and deallocation of resources based on time slots is carried out by resource sharing block.
ii. After resource allocation job 1 execution will start and after completion of time frame resources will be deallocated from job 1 eventually job 1 execution will stop.
iii. Now in second time slice resources will assigned to next job i.e. job 2 (J-2) and job2 execution will start till second time frame finishes.
iv. In third time frame resources will allocated to job3 (J-3) and job 3 execution will start.
v. Here job 3 is last job so after completion of third time frame again resources will be allocated to job 1 and job1 execution will be resumed and after completion of fourth time frame job 2 and then job 3 until each job in job queue successfully executes

3) Matchmaking Scheduler

Matchmaking scheduling technique enhances the data locality of Map tasks. The core idea behind this technique is to give every slave node a fair chance to grab local tasks before any non-local tasks are assigned to them. Local task can be defined as the task which can be executed on the node where its data is present. Scheduler tries to find the match i.e. a slave node that contains the input data, for every unassigned Map tasks. A locality marker is used to mark nodes and to ensure that each node get fair chance to grab local tasks. Experimental results of show that this technique often leads to the highest data locality rate and the lowest response time for map tasks. Matchmaking scheduler also relaxes the strict job order for task assignment unlike present in FIFO scheduler.

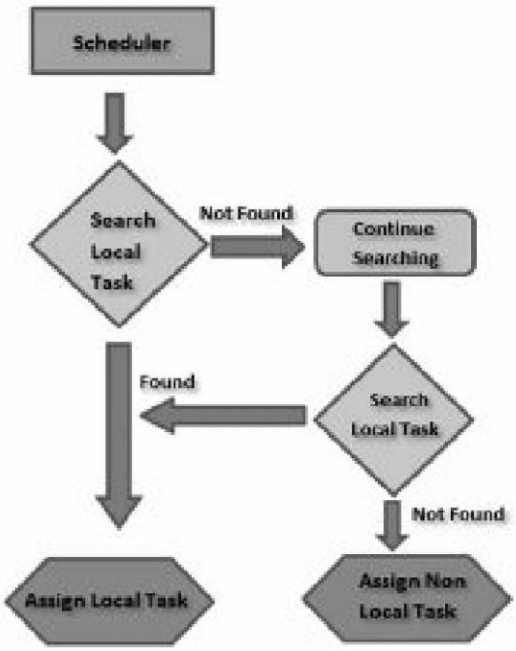

Fig 5: Matchmaking Scheduler Flowchart

Working of Matchmaking Scheduler:

Consider above Matchmaking Scheduler Flowchart

i. After getting input node/task scheduler checks whether there is any free local task which can be assigned to that input task.
ii. If scheduler finds local task which is free then scheduler assigns that local task to input task.
iii. But if scheduler doesn't find any free local task it waits for certain time and again continues search for local task.
iv. Now in second attempt if fortunately scheduler finds local task then it assigns local task to input node.
v. But if unfortunately again scheduler fails to find local task for second time, then the scheduler assigns non local task to input node.

4) Longest Approximate Time To End (LATE)

Speculative tasks are those tasks that progress very slowly. This may happen due to load on the CPU, slow background process, contention for resources etc. LATE scheduler tries to detect a slow running task to launch another equivalent task as a backup which is termed as speculative execution of task. If the backup copy completes faster, than the overall Job performance is improved. The goal of Speculative execution is to minimize a job's response time. Response time is important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring, and business intelligence. The method used for estimating time left for executing a task is Progress rate. Default implementation of speculative execution works well on homogeneous clusters but not on the heterogeneous cluster. The scheduler is highly robust to heterogeneity. This scheduler is mainly used for optimizing jobs performance. It does not ensure reliability.

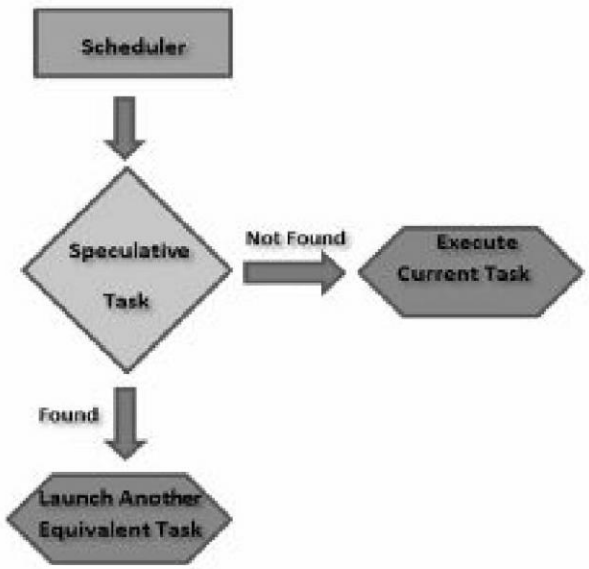

Fig 6: LATE Scheduler Flowchart

Working of LATE Scheduler:

Consider above LATE Scheduler Flowchart

i. After getting input node/task scheduler checks whether there is any Speculative task.
ii. If scheduler finds speculative task then it launch another equivalent task along with the current task.
iii. But if scheduler fails to find speculative task then it continues with normal execution of current task.

5) Resource Aware Scheduler

In Hadoop, scheduling based on resource utilization and resource contention on machines to be minimized is an active area of research. In earlier schedulers such as FIFO, Capacity scheduler, and Fair scheduler, admin can assign jobs to queues which manually guarantee their specific resource share. Resource A ware scheduler focuses on resource utilization such as CPU utilization, I/O utilization, Disk utilization, and Network Utilization, when different kinds of workload run on the cluster. Resource-aware JobTracker scheduling mechanisms are proposed, which make use of the resource

metrics:

- Dynamic Free Slot Advertisement
- Free Slot Priorities/Filtering.

In Dynamic Free Slot Advertisement, instead of having a fixed number of available computation slots configured on each TaskTracker node, the mechanism computes this number dynamically using the resource metrics obtained from each node. In Free Slot Priorities/Filtering, fixed maximum numbers of compute slots per node are retained.

Ordering of free TaskTracker slots is done for advertising, according to the resource availability.

Working of Resource Aware Scheduler:

i. Initially Resource-aware JobTracker scheduling mechanism is proposed.
ii. Resource-aware Job Tracker scheduling mechanisms make use of the resource metrics.
iii. There are two resource metrics:
   - Dynamic Free Slot Advertisement
   - Free slot priority/Filtering.
iv. In Dynamic Free Slot Advertisement, the mechanism computes number of computational slots dynamically using the resource metrics obtained from each node.
v. In Free Slot Priorities/Filtering, fixed maximum numbers of computational slots per node are assigned.

Scheduler	Technique	Environment: Homogeneous/ Heterogeneous	Job Allocation Policy	Advantages	Disadvantages
FIFO Scheduler	The job that submitted earlier get preference over job submitted later	Homogeneous	Static	Simple to implement and efficient	Reduces data locality and starvation
FAIR Scheduler	Assign recourses to each job such that an average over time each job gets equal share of available resources.	Homogeneous	Static	Fairness and dynamic resource reallocation. Can provide small response time for small jobs mixed with large jobs	Complicated configurations. It does not weight of each job, which leads to unbalanced performance in each node.
Map Matching Scheduler	Give every slave a Fair chance to grab local task before any non-local task are assigned to slave node.	Homogeneous	Static	Achieves not only high data locality but also high cluster utilization.	No particular
Longest Approximate Time To End (LATE)	the scheduler tries o detect a slow running task to launch another equivalent task as backup.	Both	Static	Scheduler is highly robust to heterogeneity.	It does not ensure reliability
Resource Allocation Scheduler	Dynamic free slot advertisement, Free slot priority/ filtering	Both	Dynamic	Helps in job performance management and resource utilization in cluster	Lack of support for preemption of reduced tasks and need additional monitoring and forecasting capabilities to manage network bottlenecks

Table 1: Comparison between job scheduling algorithm

V. Performance

To evaluate performance of above job scheduling algorithms, TeraSort benchmarking is used. The TeraSort benchmark is probably the most well-known Hadoop benchmark.

A full TeraSort benchmark run consists of the following three steps:

1. Generating the input data via TeraGen.
2. Running the actual TeraSort on the input data.
3. Validating the sorted output data via TeraValidate.

Nodes	Quantity	Specification
Master	1	2 single-core 2.2GHz Optron-64 CPUs, 8GB RAM, 1Gbps Ethernet
Slave	16	2 single-core 2.2GHz Optron-64 CPUs, 4GB RAM, 1 Gbps Ethernet, 1 rack, 2 map and 1 reduce slots per node

Table 2: System specification

Terasort benchmarking system configuration is explained in Table 2.

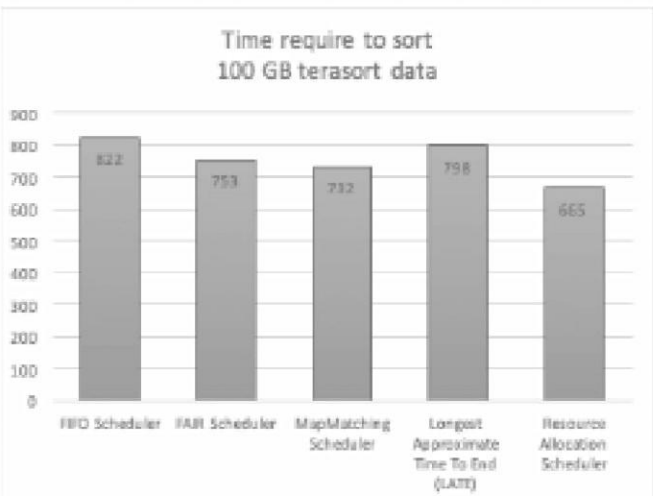

Fig 7: Performance evaluation - 100GB dataset

This terasort benchmarking is performed on the dataset of size 100GB. Fig, 7 shows the total time taken to sort 100GB data on 1 master- 16 slave hadoop system. From the figure, it can be concluded that Resource allocation scheduler has taken lesser time than all other algorithms, whereas FIFO schedule takes maximum time to sort data.

Conclusion

Local data processing takes lesser time as compared to moving the data across network. So to improve the performance of jobs, most of the algorithms work to improve the data locality.

Work load must be fairly distributed to achieve good performance.

To meet the user expectations, scheduling algorithms must use prediction methods based on the volume of data to be processed and underlying hardware. So as a future work we can consider developing the algorithms which can schedule the jobs efficiently on heterogeneous clusters.

Project Contribution

1. Literature survey - Kunal Dhande, Mohit Kulpe
2. Learning Hadoop architecture - Kunal Dhande
3. Research on Hadoop scheduler working - Mohit Kulpe
4. Research on issues related to scheduling - Kunal Dhande
5. Learning scheduling algorithms -
   a. FIFO, and Map Matching scheduler - Kunal Dhande,
   b. Fair, LATE, and Resource Allocation scheduler - Mohit Kulpe
6. Performance evaluation - Mohit Kulpe

References

[1] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "*Improving mapreduce performance in heterogeneous environments*," (2008) in 8th Usenix Symposium on Operating Systems Design and Implementation, (New York), pp. 29–42, ACM Press.

[2] Quan Chen; Daqiang Zhang; Minyi Guo; Qianni Deng; Song Guo; , "*SAMR: A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment,*"(2010) Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on , vol., no., pp.2736-2743.

[3] Houvik B Ardhan, Daniel A. Menasce. "*The Anatomy of*

MapReduce Jobs, Scheduling, and Performance Challenges", Proceedings of the 2013 Conference of the Computer Measurement Group, San Diego, CA, November 5-8, 2013.

[4] "*Apache Hadoop*", http://hadoop.apache.org.

[5] "Wikipedia", https://www.wikipedia.org/ .

[6] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, "*Job aware scheduling algorithm for mapreduce framework,*" (2011) in Proceedings of the 3rd International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, (Washington, DC, USA), pp. 724–729, IEEE Computer Society.

[7] Dongjin Yoo, Kwang Mong Sim, "*A comparative review of job scheduling for mapreduce*", Multi-Agent and Cloud Computing Systems Laboratory, Proceedings of IEEE CCIS 2011.

[8] C. H. Bennett and G. Brassard, "*Quantum cryptography: Public key distribution and coin tossing*" in Proc. IEEE Int. Conf. Compute., Syst. Signal, Bangalore, India, Dec. 1984, pp. 175–179.

[9] K. Kc and K. Anyanwu, "*Scheduling hadoop jobs to meet deadlines*" in 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 388 –392, 2010.

[10] X. Zhang, Z. Zhong, S. Feng, B. Tu, J. Fan, "*Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments*", in 9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 120-126, 2011.