Survey on Scheduling Algorithm in MapReduce Framework

Pravin P. Nimbalkar[1], Devendra P.Gadekar[2]
[1,2]Department of Computer Engineering,
JSPM's Imperial College of Engineering and Research,
Pune, India

***Abstract*— Use of internet produces large amount of data in units of hundreds of terabytes. Such large data called Bigdata can be structured, unstructured or semi structured. Traditional data management system is not convenient for storing and processing such large data. Hadoop is a most popular parallel and large data processing framework that is recently used. To achieve greater performance in processing large data, proper scheduling is required. The objective of this paper is to study MapReduce framework and various scheduling algorithms that can be used to gain better performance in scheduling.**

***Index Terms*— Big Data, Hadoop, MapReduce, Job scheduling.**

I. INTRODUCTION

Today large use of internet producing large amount of data. Various sources producing Big data are social networking sites, transactional data from enterprise applications/databases, sensors, mobile devices, machine generated data, huge number of high definition videos and many more. Now the question arises that how to process such large amount of data. Also this data generation cannot be stopped. The processing of Bigdata can be efficiently done using distributed computing and parallel processing. Hadoop is one of such distributed computing framework that incorporate the similar features as that of Google file system and MapReduce programming paradigm. In MapReduce each application is implemented as a sequence of map and reduce stages. These stages process the large number of independent data items. Map stage processes the input in form of key/value pair and produces the output as set of intermediate key/value pairs. Then the reduce stage merge all intermediate values associated with same intermediate key.

MapReduce automatically handles the node failure; hide the complexity of fault tolerance from programmer. MapReduce also allow performing map and reducing operation in parallel. MapReduce processes the large datasets with commodity computers. As MapReduce is becoming popular nowadays, One of the important factors that to be consider is their scheduling.

This paper gives the overview of various scheduling algorithms used in MapReduce. Also the description of Hadoop framework, Hadoop distributed file system and MapReduce framework is given here.

II. HADOOP

Hadoop is an open source, batch data processing, large scale, distributed computing framework for storing big data and to analyze this data. Hadoop provides scalability and also takes care of detecting the failures and handling them. By creating multiple copies of the data in different nodes, it ensures high availability of the data throughout the cluster.

Hadoop framework allows the developers to create program that process the data in parallel and focus on computation problems that occurs in it. All the details of parallel processing such as distribution of data to different nodes, restarting the failed tasks are hidden by Hadoop. Hadoop includes two things 1) Hadoop Distributed File System (HDFS) 2) Hadoop MapReduce

A. Hadoop Distributed File System (HDFS):

HDFS is a file system designed for storing very large files with streaming data access pattern, running clusters on commodity hardware. The size of files can be in terabytes. There are two measure components of HDFS, these is NameNode and DataNodes. NameNode is the master node on which daemon called jobtracker is runs. NameNode maintains and manages the blocks on DataNodes. It will be very expensive hardware and it will have double or triple redundancy devices. DataNode is the commodity hardware on which another daemon called Tasktracker is runs. Daemon is a process or service that runs in background. DataNode works as slave which are deployed on each machine and it actually stores the data. It is responsible for serving read and write requests for client.

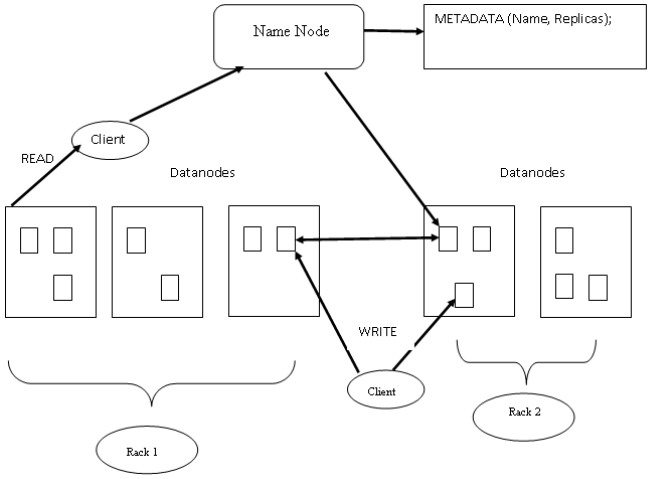

Figure 1. Architecture of HDFS

Fig 1 .Shows the architecture of HDFS. As shown in fig.1 rack is the storage area where multiple DataNode put together. To maintain the fault tolerance on Hadoop multiple copies of data are maintained on multiple DataNode. This is called data replication and each copy of data is called replica [1]. Minimum there should be three copies of replica. Basically all the operations are performed in terms of blocks. The default block size is 64 MB. The client interacts with NameNode and DataNode using SSH protocol. Client does read and write with DataNode and NameNode. NameNode stores the metadata on RAM. Metadata is not a actual data but it is the information about which are the NameNodes, where are the rack, on which rack which DataNode is there. There is something called secondary NameNode which periodically reads the data from RAM of NameNode and persist that data into hard disk. Bur the secondary NameNode is not a substitute for NameNode. If NameNode fails the system goes down.

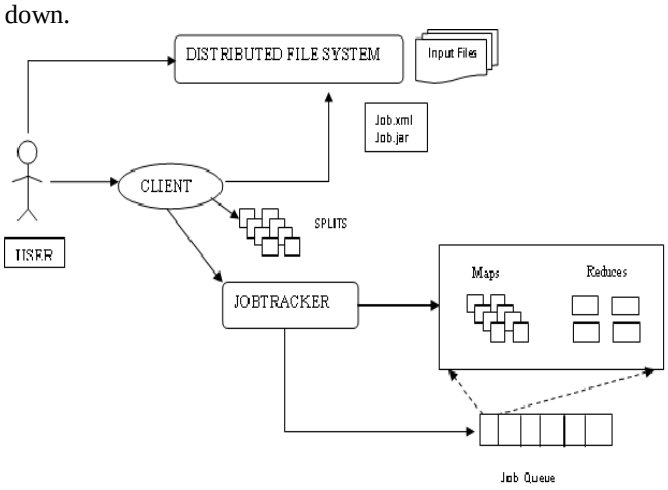

Figure 2. Job Tracker

The behavior of Hadoop is writing once, read many times. As shown in fig 2, if user has some query, he first copies the file like some kind of logs or information on distributed file system. Then user submits a job to client. When some kind of analysis is to be done and some kind of data is to be fetched out, client tries to get the information called metadata from DFS. When the job is submitted it is broken down into number of smaller jobs. The information about these jobs is uploaded on DFS. Then client submits the job to Jobtracker. Now the jobtracker try to find where are all the block located for the particular file which is to the be processed. It will initialize the job in job queue. Job queue has all the jobs that are created out of splits. The jobs in queue are served one by one. To process the jobs, first job from queue is picked up and jobtracker creates Maps and Reduces. Map and Reduce are the programs that will be send to DataNode for processing. There will be as many maps are created as the many input splits. Map processes the data first; it generates the kind of <key value> pair. On that particular <key, value> pair, the Reduces works to give final output. There map and reduces are written on the basis of business logic. The business logic is requirement based, what user want to processed.

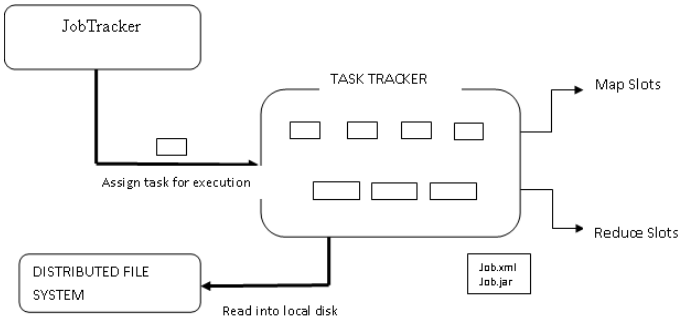

Figure 3. Task Tracker

As shown in fig 3. the Tasktracker sends heartbeats to jobtracker to inform the aliveness. The jobtracker picks the job for job queue and assign it to Tasktracker those are alive. Tasktracker has number of map and reduce slots that can be configured. Map processes the job and produces intermediate result in form of <key, value> pair. Reduce takes this <key, value> pair as input and produces the final output.

B. MapReduce:

Hadoop MapReduce is a software framework that allows to write a application which process large amount of data in parallel on a large cluster of commodity hardware[2]. Usually MapReduce job splits the input datasets into number of independent chunks. These chunks are processed by the map tasks in parallel. MapReduce sorts the output of map. This sorted output is then given as a input to reduce tasks. Typically both the input and output of jobs are stored in file system. All the functions like scheduling the tasks, monitoring their execution and re-executing the failed tasks are handling by the framework.

The MapReduce framework consists of one jobtracker and one Tasktracker per node in cluster. The Jobtracker works as a master and Tasktracker works as slave. The jobtracker takes care of scheduling the tasks, monitoring them and re-execution of failed task. As per the instructions of jobtracker, the Tasktracker executes the tasks.

The MapReduce framework views the input to job as a <key, value> pair and produces a intermediate set of <key, value> pairs as shown in fig 4. These pairs are then shuffled across different reduce tasks based on {key, value} pairs. Each Reduce task accepts only one key at a time and process data for the key and outputs the results as {key, value} pairs. The job submitted by user is then received by Jobtracker and breaks it into number of map and reduce tasks. It then assigns task to Tasktracker, monitors the execution of job and when

job is completed informs to the user. As in Hadoop all the jobs have to share commodity servers in cluster for processing the data, proper scheduling policy and algorithms are required.

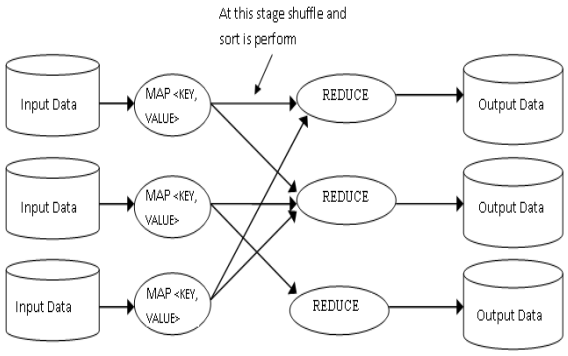

Figure.4: Hadoop MapReduce Architecture

III. SCHEDULING IN HADOOP

A. *Default FIFO Scheduler*

By default Hadoop has the first in first out scheduling for task assignment. When a job is submitted, it is divided into number here is facility of subtasks and these subtasks are queued into the job queue. Then the jobtracker assigns these tasks to available free slots of Tasktracker node. Also there is a facility for assigning the priorities to job, but it is not enabled by default. FIFO approach for scheduling is easy to implement. But there is a drawback that there is no fair sharing of resources.

B. *Fair scheduler*

Facebook developed the another scheduler for fair sharing of resources among jobs called fair scheduler [3].The fair scheduler assigns the resources to job in such a way that on average, equal share of available recourses will be given to each job. Therefore each job either is a job which require less time to execute or a job with more execution time. In fair scheduling set of pools are created where all jobs are placed. From this pool jobs are selected. Each pool is assigned a guaranteed minimum numbers of map and reduce slots. The excess capacity in pool is shared among the jobs by allocating free slots in idle pools to other pools running the jobs under capacity. If the pools do not get fair share then there is provision of preemption in fair scheduling. In such case scheduler can kill the tasks in pools that are lastly allocated. This will reduce the wasted computation. The preemption do not cause the preempted job to fail, only makes them longer to finish.

C. *Capacity Scheduling*

`Some of the principles of the fair scheduler are shared by capacity scheduler. Capacity scheduler [4] was invented by yahoo. In capacity scheduler pool is used instead of queues. Each queue is created with number of configurable map and reduce slots. Also queue is assigning a guaranteed capacity. Due to this capacity scheduler provides minimum capacity guarantee. If any queue is not using its full capacity then the excess capacity can be shared among other overloaded queue. Capacity scheduler also has the capacity of prioritizing jobs in queue. Generally higher priority tasks are executed before small priority task. In capacity scheduler there is strict access control on queues. These access controls are defined on per queue basis.

D. *Longest Approximation Time to End (LATE)*

Sometimes it is possible that some of the tasks are running slowly. It may be happed due to CPU load or slow background processes. The scheduler in this case uses the speculative execution. Speculative execution means scheduler finds out slow running task and starts another task as backup. Speculative execution only improves the performance but it does not ensured the reliability. If there are errors in task code due to which task is slow, this will also slow down the speculative execution. Also speculative execution assumes that the task progress is uniform on each node and also there is uniform environment. Due to these assumptions speculative executions do not work on heterogeneous environment. Zaharia et al [5] proposed an algorithm called Longest Approximate Time to End (LATE). Instead of progress made by the task, this algorithm uses remaining time to complete the execution of task. Using this approach significant improvement in the job response time over the default speculative execution can be obtained.

E. *Delay Scheduling:*

Fair scheduler allocates fair share of capacity to each node. But there are two locality problems that are observed with fair scheduler. These problems are head-of-line scheduling and sticky slots. First locality problem occurs with the small jobs. These jobs are having small input files and hence there are small numbers of input files to read. When job reaches for scheduling at the head of sorted list, one of its tasks is launched on the next slot that becomes free irrespective of which node this slot is on. When there is small job at head-of-line, it is unlikely to have data locally on the node that is given to it. Facebook observed this head-of-line scheduling problem in version of HFS without delay scheduling. The second locality problem of sticky slot is that, for any job there is common tendency that job should be assigned to same slot repeatedly. The problem occurs if strict queuing order is followed, then it not always possible to schedule the job with local data.

To overcome the problem of head-of-line, scheduler assigns task from job to the node with no local data. This will maintain fairness but violets the key objective of MapReduce that schedule the task near to their input data. Running a task on node that contains the input data is most efficient method. But if it is not possible, then running it on node with same rack is also faster than running on off-rack. Delay scheduling [6] improves the locality by making the jobs to wait until there is opportunity of scheduling on node with local data. When a node request for task and the head-of-job cannot launch a local task, then this job is skipped and subsequent jobs are looked up. However, if the job is skipped for long time, to avoid the starvation, non local tasks are allowed to launch. The key idea behind delay scheduling is that although the first slot we consider giving to a job is unlikely to have data for it,

tasks finish so quickly that some slot with data for it will free up in the next few seconds.

F. Dynamic Priority Scheduling:

Dynamic priority scheduling [7] supports dynamic distribution of capacity to the concurrent users. For distributing the capacity among the users, their priorities are considered. Automated capacity allocation and redistribution of capacity is supported in a regulated task slots market. In this approach uses are allowed to get Map and Reduce slots per time unit based on proportional share. These time slots are called as allocation intervals and they can be configured. Normally time slots are set in between 10 seconds to 1 minute.

Dynamic priority scheduling supports the preemption. If allocated time slots were not used for long time, they can be preempted and allocated to other users. Dynamic priority scheduling can be configured to behave like other schedulers. It can be reduced to FIFO scheduler if users or queues have no credit left. It behaves like fair scheduler when all queues are configured with same share and there is very large allocation interval.

G. Deadline Constraint scheduling:

To address the issue of deadline, this scheduler is developed. Along with deadline this scheduler gives more attention on increasing the system utilization. Handling the deadline requirement is carried out by job execution cost model and a constraint based Hadoop scheduler [15]. The job execution cost model considers various parameters such as runtime of Map and Reduce, Size of input data, distribution of data etc. Constraint based scheduler takes user deadline as a part of its input.

The jobs in deadline constraints scheduler [8] are scheduled only if it meets the specified deadline. Once the job is submitted, it is determined that whether the job can be completed within specified deadline or not. Irrespective of job running in system, available free slots are computed at given time and in the future. The job is said to be scheduled if and only if available slots are equal to or greater than the minimum no of tasks for both map and reduce. This scheduler shows that when there are different deadline for jobs, different number of tasks are assigned to Tasktracker and makes sure that specified deadline is met.

H. Locality aware Scheduling

Data Locality aware scheduling starts working when request from a requesting client is received [14]. When request is received, It schedules the task where input data is available on requesting node. If no such node found, it will select the task that is having input data nearest to requesting node. Then it declares whether to reserve the task on node that stores input data or schedule the task to the node that sent a request. Utilization of this method improves the data locality and also it involves runtime overhead that is the waiting time required to schedule the task to the node having input data. This waiting time can be longer than the second options runtime overhead. Hence the transmission time it takes to copy input data to requesting node. The jobtracker receives task from job queue and divides it into multiple tasks. Then data locality scheduling schedules tasks to Tasktracker nodes. This method would performs the functionality as follows: Accept request form requesting node, schedule task, reserve the task for node storing input data, schedule the task to node from which request is received by transferring input data to that node.

IV. COMPARISION

FIFO: This scheduler is easy to implement but drawback is when resources are used by large jobs, there will be starvation of small jobs.

Fair scheduler: Fair scheduler let the short jobs to finish in reasonable time while not starving the large jobs. Number of jobs that can run concurrently in job pool are limited. If too many jobs are submitted, then until the previous jobs are completed and frees the job slots, next job has to wait in scheduling queue.

Capacity scheduler; It has the ability to priorities the jobs. Thus higher priority jobs have access to resources sooner than low priority jobs. It has full access control on queue.

LATE: If some bugs slow down the jobs, speculative execution cannot be used as solution because the speculative jobs are also likely to be affected by some bugs.

Dynamic priority scheduler: Automatic allocation and distribution of capacity is supported. Also the preemption of job is allowed which leads to improve overall performance.

Deadline Constraint scheduler: Jobs are allowed to schedule only if they meets specified constraints.

V. CONCLUSION

For efficient storage and processing of Bigdata with best computational power with less time Hadoop framework is used. In this paper we studied various techniques that create efficient scheduler for MapReduce using which we can improve the speed of Hadoop system that in turns improves data retrieval and job processing.

REFERENCES

[1] http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[2] http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[3] MateiZaharia, "The Hadoop Fair Scheduler "http://developer.yahoo.net/blogs/hadoop/FairSharePres.ppt
[4] Hadoop's Capacity Scheduler, http://hadoop.apache.org/core/docs/current/capacity_scheduler.html
[5] MateiZaharia, DhrubaBorthakur, JoydeepSenSarma, KhaledElmeleely, Scott Shenker and Ion Stoica, "Delay Scheduling: A Simple Technique For Achieving Locality and Fairness in Cluster Scheduling"
[6] Thomas Sandholm and Kevin Lai. Dynamic proportional share scheduling in hadoop. In JSSPP '10: 15th Workshop on JobScheduling Strategies for Parallel Processing, April,2010
[7] K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines", in Proc. CloudCom, 2010, pp.388-392
[8] Mark Yong, NitinGaregrat, Shiwali Mohan: "Towards a Resource Aware Scheduler in Hadoop" " in Proc. ICWS, 2009, pp:102-109
[9] Dean, J. and Ghemawat, S., "MapReduce: a flexible data processing tool", ACM 2010.
[10] Chen He Ying Lu David Swanson, "Matchmaking: A New MapReduce Scheduling Technique", Department of Computer Science and Engineering, University of Nebraska-Lincoln Lincoln, U.S.
[11] Hadoop Distributed File System, http://hadoop.apache.org/hdfs

[12] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, Wei Ding, "Data Mining with Big Data", 2013.
[13] Yang XIA†, Lei WANG1, Qiang ZHAO1, Gongxuan ZHANG2, "Research on Job Scheduling Algorithm in Hadoop".
[14] PraveenkumarKondikoppa , Chui-Hui Chiu, Cheng Cui, Lin Xue and Seung-Jong Park, "Network-Aware Scheduling of MapReduce Framework onDistributed Clusters over High Speed Networks", Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit,September 21, 2012, San Jose, CA, USA
[15] Chien Hung Chen, "Deadline-Constrained MapReduce Scheduling Based on Graph Modelling ", IEEE 7th International Conference on Cloud Computing (CLOUD) 2014.
[16] Geetha J, N UdayBhaskar , P ChennaReddy,NehaSniha ,"Hadoop Scheduler with Deadline Constrints", International Journal on Cloud Computing: Services and Architecture (IJCCSA) ,Vol. 4, No. 5, October 2014 .

