ABSTRACT:
Today’s Digital era causes escalation of datasets. These datasets are termed as “Big Data” due to its massive amount of volume, variety and velocity and is stored in distributed file system architecture. Hadoop is framework that supports Hadoop Distributed File System (HDFS) for storing and MapReduce for processing of large data sets in a distributed computing environment. Task assignment is possible by schedulers. Schedulers guarantee the fair allocation of resources among users. When a job is submitted by user, it will be placed into a job queue. A job will be then divided into tasks and distributed among different nodes. Proper assignment of tasks will reduce job completion time. This can guarantee improved performance of the job. In this paper we study MapReduce model and evaluated task scheduling algorithms such as FIFO, Fair share, Capacity, Delay, IWRR and MTL of Hadoop platform.

Keywords: MapReduce, HDFS, Hadoop, Big Data, fairness, data locality

I. INTRODUCTION

Millions of users are using applications based on Internet services work with sheer volume of data has lead to parallel computing on clusters. Processing and storing giant amount of data in parallel manner become a challenge to computing globe. Hadoop[1] is a open source Java based framework which can run applications in the cluster that consist of reasonably priced hardware, for processing and storing large amount of data in distributed computing environment. Hadoop uses HDFS (Hadoop Distributed File System) for storing data and to process these data it uses MapReduce Programming model introduced by Google. Each MapReduce job is consisting of a number of map and reduce tasks. The MapReduce model for handling multiple jobs consists of a processor sharing queue for the Map Tasks and a multi-server queue for the Reduce Tasks [2]. The entire job is divided into self-governing tasks, and all the tasks require system slot to execute. MapReduce clusters have become most admired now-a-days because of their fault tolerance nature. One of the fascinating matter is their task scheduling. There are three important scheduling issues in MapReduce such as locality, synchronization and fairness. Section 2 contains brief review of some related research. Section 3 commence MapReduce programming model. Section 4 introduces some of the task scheduling algorithms of Hadoop and their comparative analysis. Section 5 wind up the article.

II. RELATED WORK

Face book has introduce a Fair share scheduling[3] algorithm which allocates minimum number of shared resources to each job whereas Yahoo has designed a Capacity Scheduling[4] algorithm which puts jobs in multiple queues and allocate certain system capacity for each queue. In [5] authors have proposed a waiting scheduling depending on length of waiting time. In [6] authors have examine weighted round robin scheduling algorithm and put forward weight update rule to reduce workload and to balance task allocation in improved weighted round robin algorithm along with experimental results. In [7] authors compared four different scheduling algorithms along with their advantages and disadvantages in tabular representation akin to Longest Approximate Time to End which takes into consideration node heterogeneity when deciding where to run speculative tasks and execute only tasks that will improve job response time. Self-Adaptive Map Reduce Scheduling algorithm which combines historical information recorded on each node and dynamically find the slow tasks. It saves time for execution and system resources, FIFO that works better when jobs are short, Enhanced Self-Adaptive Map Reduce Scheduling algorithm, which improves the performance in terms of estimating task execution time and launching backup tasks. In [8] authors describes the overview of Hadoop[1] Scheduling algorithms issues and problems and highlight implementation idea and corns-pros of algorithms. In [9] authors have proposed Multi-Threading Locality Scheduler which utilizes multi threaded architecture for scheduling. Two major factors, simulation time and energy consumption are used to test the evaluated algorithms such as FIFO, Delay, Matchmaking and MTL scheduling algorithms on virtualized infrastructure. MTL has advantage over other existing schedulers in solving data locality problem by parallel searching using multi threading techniques.
III. MapReduce Programming Model

MapReduce[10] is programming model used in Hadoop for processing vast amount of datasets in clusters. Map and Reduce are two significant functions of MapReduce. Users can develop their own customized map and reduce functions. Fig 1[11] gives an idea about the functionality of map and reduce function. Map function takes as an input a key/value pair and generates intermediate key/value pair, then intermediate list of key/value pair is sorted based on key value and it come to pass onto reduce function. Reduce function combine these values to produce possibly smaller set of values. Generally one or zero output value is generated per reduce function invocation. In sort, Map function split job into number of tasks and Reduce function assemble results of multiple tasks to produce final result.

![Fig 1. Functionality of Map and Reduce Function[11]](image)

Fig 2[12] gives information regarding different components in MapReduce Architecture. Name Nodes and Data Nodes are known as HDFS nodes where as Job Tracker and Task tracker are known as MapReduce nodes. Following table portray function done by each of the node.

<table>
<thead>
<tr>
<th>Sr. no</th>
<th>Node type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Name node</td>
<td>It supplies all the meta data and consists of the information about location/address of the data blocks.</td>
</tr>
<tr>
<td>2.</td>
<td>Data Node</td>
<td>It stock up blocks of HDFS.</td>
</tr>
<tr>
<td>4.</td>
<td>Task tracker Node</td>
<td>Carry out Map and Reduce Functions.</td>
</tr>
</tbody>
</table>

![Fig 2. Components of MapReduce[12]](image)

Every job comprises of many tasks. Job tracker in point of fact tracks what a task tracker is doing and then obtain the job done by a task tracker. A job tracker accepts applications from a client. It subsequently discuss with the name node and finds out location of the data on which processing wants to be done. Job tracker allocates the job to a task tracker which is in charge for execution of the tasks. Several task trackers can work in chorus thereby executing processes in parallel. When a work is fulfilled the job tracker updates its status. The client can inquire the job tracker regarding the position of the job.

A task tracker is a node in a cluster that accepts jobs from a job tracker. Each task tracker has a predetermined number of map slots and reduces slots in which it can run tasks. When a job tracker is provided with a task by the client, job tracker seems for task tracker. The job tracker opt that task tracker that resides in the same server as the data node. If not any of the slots of this task tracker is free next the job tracker looks for a server which is present on the same rack. Every task tracker
periodically updates the job tracker telling that it is active and in this process also makes the job tracker aware of number of empty slots.

**FIG 3. MASTER-SLAVE MAPREDUCE ARCHITECTURE[10]**

Fig 3[10] shows the Master-Slave MapReduce architecture, where job tracker node behaves reminiscent of a master and manages other task tracker nodes behave like slave nodes.

**IV. COMPARATIVE ANALYSIS OF EXISTING SCHEDULING ALGORITHM**

Mainly three scheduling issues be taken into consideration: fairness, locality and synchronization. Fairness finiteness has trade-offs between the locality and dependency between the map and reduce phases. Locality is defined as the distance between the input data node and task-assigned node. Synchronization be the process of transmitting the intermediate output of the map processes to the reduce processes as input is also considered as a factor which affects the performance [8]. Task Scheduling is an aspect that directly affecting the overall performance of Hadoop platform and utilization of system resources. There are various algorithms to resolve this issue with different techniques and approaches. A number of them obtain center of attention to improve data locality and some of them implements to provide Synchronization processing. Also, numerous of them have been designed to minimizing the total completion time [8]. In this section let discuss some of the scheduling algorithms.

**A. FIFO scheduling algorithm**

This is a default scheduler used by Hadoop which operates using a queue. In this approach job is first partitioned into individual tasks, and afterward loaded into the queue and assigned to free slots on Task Tracker (slave). Each job would exploit the complete cluster, as a result jobs be obliged to wait for their turn. The major disadvantage of this algorithm is that once the previous job With the scheduler, the major drawback is that only after finishing the previous job, subsequently jobs in the job queue will be assigned. The scheduler implementation is straightforward and proficient[13].

**B. Fair Share Scheduling Algorithm**

The Fair Scheduler was developed at Facebook to manage access to their Hadoop cluster [14]. The Fair Scheduler gives equivalent share of cluster capacity to each user. Users may assign jobs to pool, with each pool allocated a definite minimum number of Map and Reduce slots [10] [15]. If there are free slots in pools then they may be allocated to other pools, while excess capacity within a pool is shared among jobs. The Fair Scheduler is a preemptive that is to say if a pool has not received its fair share for a certain period of time, then the scheduler will destroy tasks in pools running over capacity in order to give the slots to the pool running underneath capacity. seeing that jobs have their tasks allocated to Task Tracker for computation, the scheduler track the shortfall between the amount of time actually used and the ideal fair allocation for that job. When slots become free, the next task from the job with the highest time shortfall is assigned to the free slot. Ultimately, this has the effect of make sure that jobs receive approximately equivalent amounts of resources. Shorter jobs are allocated sufficient resources to finish quickly. Simultaneously, longer jobs are assured to not be starved of resources.

**C. Capacity Scheduling Algorithm**

Capacity Scheduling algorithm was developed to manage fair distribution of resources among huge mass of users. The Capacity Scheduler allocates jobs based on the submitting user to queues with configurable numbers of Map and Reduce slots [16][19][20]. Queues which have jobs are given their configured capacity where as free capacity in a queue is shared between other queues. Scheduling is driven on a customized priority queue basis with specific user restrictions within a queue, with priorities adjusted based on the time a job was submitted, and the priority setting allocated to that user and category of job[19]. When a Task Tracker slot becomes free, the queue with the lowest load is selected, from which the oldest lasting job is nominated. A task is next scheduled from that job. In general, it has the cause of enforcing cluster capacity sharing among users, rather than between jobs.
D. Delay Scheduling
Authors of [17] and [18] have discussed delay scheduling algorithm. Fair scheduling algorithm was developed to allocate fair share of capacity to all the users. The scheduler launches a task from a job on a node. If task is run on the node that contain the data, it is most efficient, but when it is not possible, running on a node on the same rack is faster than running off-rack. Delay scheduling is used to improve data locality by asking jobs to wait for its turn for scheduling on a node with local data. When a node requests a task and if the head-of-line job cannot launch a local task then it is skipped and looked at next jobs. But if a job has been skipped for long enough then non-local tasks are allowed to launch to prevent starvation. In this algorithm although the first slot given to a job is not likely to have data for it, but tasks come to an end very quickly that some other slot contain data for it will free within a small amount of time.

E. Improved Weighted Round Robin Scheduling Algorithm
In [6] authors have proposed IWRW scheduling, based on the analysis of WRR algorithm. Under the unweighted conditions, tasks of each job are submitted to Task Tracker in turn. Under the weighted conditions, multiple tasks of the larger weight job will run in a round, and the job's weight will be changed along with the increase or decrease of jobs' number. At times, if the number of tasks of the smaller weight job becomes more, while the number of the larger Weight job is less, then the weight of the smaller weight job will be increased correspondingly, so the number of tasks which are assigned to Task Tracker will be relatively increased, and the weight of the larger weight job will be appropriately decreased, the number of tasks which are assigned to Task Tracker will be relatively decreased. However, the relationship between them remains the same in order to achieve load balance. This algorithm used weight update rules to reduce workload and to balance tasks’ allocation. The algorithm is easy to be implemented with low cost and suitable for the Hadoop platform that uses the only Job Tracker to schedule.

F. Multi-Threading Locality Scheduling
In [9] authors have designed MTL scheduler. MTL scheduler utilizes multi-threading architecture to perform all the scheduling processes where each thread is assigned to a predetermined block of jobs. The MTL starts by dividing the cluster into N blocks. Each block contains a number of commodity machines to process and store input data. Each block is scheduled by a special thread that schedules the jobs in the wait queue. Once a new job arrives to the cluster, the map-reduce scheduler contacts the Name node to determine the rack that includes the largest proportion of data locality tasks for this job. When any job is to be processed, the threads start searching in their blocks node for local map task, where each thread takes information regarding current task and starts searching in their blocks. Once a thread finds a local data for this task, it immediately notifies other threads to stop searching for this task and starts searching for next tasks. If all threads could not be able to find more local task, the threads starts in assigning just one non-local map task for each node in the block for this heartbeat. The MTL scheduler differs from the previous schedulers in dealing with synchronizations blocks scheduling, through using multi-threading scheduling. It uses multi-threading scheduling on a cluster of nodes. In fact, it has an advantage over other existing schedulers in solving data locality problem by parallel searching using multi-threading techniques.

<table>
<thead>
<tr>
<th>Approach/Parameter</th>
<th>FIFO Scheduling</th>
<th>Fair Share Scheduling</th>
<th>Capacity Scheduling</th>
<th>Delay Scheduling</th>
<th>IWRW Scheduling</th>
<th>MTL Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Non Preemptive</td>
<td>Preemptive</td>
<td>Preemptive when job fail</td>
<td>Preemptive</td>
<td>Preemptive</td>
<td>Non Preemptive</td>
</tr>
<tr>
<td>Implementation</td>
<td>Simple</td>
<td>Less Complex</td>
<td>Complex</td>
<td>Simple</td>
<td>Simple</td>
<td>Simple</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Response Time</td>
<td>Low for short jobs</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Performance</td>
<td>High for small cluster</td>
<td>High for both large and small cluster</td>
<td>High for large cluster</td>
<td>High for small cluster</td>
<td>High for both large and small cluster</td>
<td>High</td>
</tr>
<tr>
<td>Execution</td>
<td>Serially</td>
<td>Parallel</td>
<td>Parallel</td>
<td>Parallel</td>
<td>Parallel</td>
<td>Parallel</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No of Queues/Pools for user</td>
<td>Single queue</td>
<td>Multiple pool</td>
<td>Multiple queue</td>
<td>Single queue</td>
<td>Multiple sub queue</td>
<td>Single queue</td>
</tr>
<tr>
<td>Data Locality</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Fairness</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Type of job</td>
<td>Single batch job</td>
<td>Different types of job</td>
<td>Different types of job</td>
<td>Different types of job</td>
<td>Different types of job</td>
<td>Different types of job</td>
</tr>
</tbody>
</table>
The paper gives an overall idea about Hadoop MapReduce architecture and its different task scheduling algorithms. We analysed properties of various task schedulers based on working mode, response time, performance, data locality and fairness provision, execution style, resource utilization, load balancing. The choice of scheduling algorithm is entirely depend on user requirements, but in spite of everything we need widespread scheduling algorithm to get superior performance of Hadoop MapReduce Model.

References

[12] https://dipayan90.wordpress.com/2013/06/03/hadoop-architecture/