Size based Multithreaded Scheduler for Hadoop Framework

Poonam S. Patil
Department of Computer Engineering
Savitribai Phule Pune University
Pune, India

Rajesh. N. Phursule
Department of Computer Engineering
Savitribai Phule Pune University
Pune, India

ABSTRACT
The majority of large-scale data severe applications executed by data centers are based on MapReduce or its open-source implementation i.e. Hadoop. For processing huge sum of data in parallel Hadoop programming framework provides Distributed File System (HDFS)[2] and MapReduce Programming Model[3]. Job scheduling is an imperative process in Hadoop MapReduce. Hadoop comes with three types of schedulers namely FIFO, Fair and Capacity Scheduler. In some processing scenario these traditional scheduling algorithm of Hadoop cannot meet the performance requirements and fairness criteria of Big Data Processing. To address this issue new efficient scheduler is require who can identify the data size first and processed accordingly for performance improvement. This new MapReduce scheduling scheme Will improves MapReduce performance and erasure high speed data processing.

Proposed system will analyze the data size of individual DataNode and create threads based on threshold value decided by proposed scheduler. Processing of the threads is done parallel on individual DataNode by task tracker which will ultimately improve the data process performance. Because of that task Tracker will does the work in less time than the time required by the traditional Scheduler.

Keyword
MapReduce, Big Data, Scheduling, HDFS

1. INTRODUCTION

1.1 Big data
the term 'Big Data[5] Big data refer to massive data sets that have larger magnitude(Volume), more range, including structured, semi structured and unstructured data (variety), and arriving more rapidly (Velocity) than the traditional system, For handling this vast data industries require new technique, approach, tool's & architecture this will solve new problems very efficiently as well as old problems in the better way.

The main key enables for the growth of "Big data"[12] are:

- Increase storage capacity
- Increase processing power
- Availability of data

1.2 Apache Hadoop Framework
Hadoop[1] is a software echo system Hadoop is a software framework that provides a simple programming model to enable distributed processing of large data sets on clusters of computers. Hadoop applications are used to process big data since the traditional systems are incapable of processing such volume of data.

Hadoop software is a completely open-source framework for large data analysis. It includes two main components, i.e. HDFS for parallel Processing and Map Reduce for data analysis purpose.

1.3 Map Reduce
MapReduce[3] is a data intensive processing with support of cloud computing technology. It is a convenient programming interface for working in a cluster environment. MapReduce can be used for predictive analysis, e.g. it can be used to identify whether forecasting, earthquake prediction, etc. MapReduce is the programming backbone to get the distributed data processed. And the analysis of that data can be done by using the various front end analytical data tools like R tool, SAS.

The strengths of MapReduce are fault tolerance[3], an easy programming structure and high scalability. Map Reduce Programming framework can be applicable in various environments like web data processing, scientific analysis, high-performance computing, etc.

Hadoop MapReduce Framework is Master-slave Architecture in that job tracker works as Master node while Task Tracker works as Slave node.

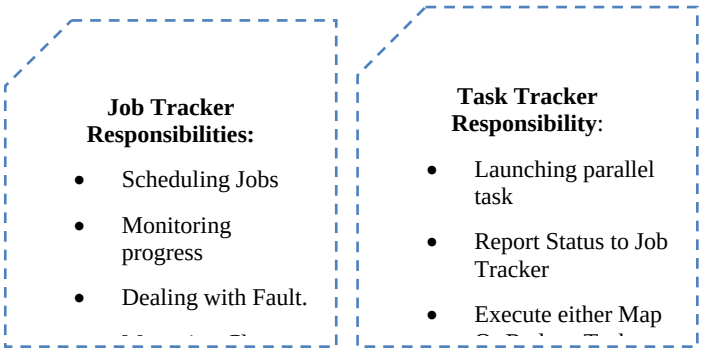

Figure 1: Hadoop Job and Task Tracker Responsibility

This programming model takes input as key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce.

Map, In Hadoop framework data is stored in HDFS the input data is taken from HDFS in the form of KEY VALUE PAIRS; Map function processes the input< KEY, VALUE > to form new< KEY, VALUE > this new value is called intermediate key I and this key are passes to Reduce function. **Iterator** function uses to pass the intermediate value to Reduce function. **Reduce,** this function accept intermediate key pass by Map function and respective set of values for the key.

1.4 HDFS

Hadoop distributed file system[2] is popular because of its scalability, reliability and capability of storing a very large file. HDFS is a Master-Slave architecture in that two-component present name node and data node. Name node works as Master Node accountable for Storing and managing of metadata. Name node is unique in the cluster, and it is a machine in a cluster with a high configuration while in other hand data node works as a slave node which is responsible for storing a block of data and serve that data on request over a network, these data blocks are replicated in the network. Slave nodes are commodity computers. There is one more component present in HDFS in secondary name node, which is responsible for the storing the backup for the name node's metadata. The important thing about the secondary name node is it will not work as a backup or standby node to a name node.

2. BACKGROUND

2.1 Issues in Scheduling

Locality: Locality[9] is defined as the distance between the input data node and task -assigned node. When the distance between data node and the computation node is less than the data transfer cost is less. Locality is a critical issue which can affect the performance in shared distributed environment; high locality improves the throughput of the tasks. If the locality requirement is not fulfilled, data transferring I/O costs can seriously affect the performance because of the shared bandwidth of network. Most methods of scheduling of map reduce jobs follow a policy of attempting to assign tasks to a place near the input data to save cost of network.

Synchronization: Synchronization[15] is the process of transferring the intermediate output of the map processes to the reduce processes as input is also consider as a factor which affects the performance. Mappers have to wait until all the map processes are finished to initiate sending intermediate output. Due to dependency between the map and reduce phases of processing, a single node can slow down the whole process, causing the other nodes to wait until it is finished. There are various factors which results in performance degradation in the synchronization step, few of them are as heterogeneity of the cluster, node failures miss-configuration, and serious overhead of the I/O cost.

Fairness: Various map-reduce jobs are performed in a shared data warehouses of enterprises like facebook, Amazon, Google and Yahoo. A map-reduce job with a heavy workload may dominate utilization of the shared clusters, so some short computation jobs may not have the desired response time. The demands of the workload can be elastic, so fair workload to each job sharing cluster should be considered. Fairness constraints have tradeoffs between the locality and dependency between the map and reduce phases. When each map-reduce job has roughly an equal share of the nodes and the input files are spread in distributed file system, some map processes have to load data from the networks. This causes a great degradation in throughput and response time. Synchronization overhead could affect the fairness. For example reduce processes have to wait for the completion of map processes which leads to idle nodes and starvation of other jobs. Due to this problem a poor utilization situation occurs.

2.2 Role of Scheduler in Hadoop

A scheduler plays a very significant role in the big data processing[5]. Hadoop implements the ability for pluggable schedulers that assign resources to jobs. However, as we know from traditional scheduling, not all scheduling algorithms are the same, and efficiency is workload and cluster dependent. Apache Hadoop framework should provide an efficient scheduling mechanism for enhanced utilization in a shared cluster environment. The problems of scheduling MapReduce jobs are mostly caused by locality and synchronization overhead. Also, there is a need to schedule multiple jobs in a shared cluster with fairness constraints. By default, Hadoop uses FIFO to schedule jobs. Alternate scheduler options: capacity and fair scheduler. Each scheduler having its own advantage and disadvantage according to volume, verity and velocity of data present in HDFS.

2.3 Scheduler in Hadoop

- **First-In First-Out Scheduling**

This scheduler is inspire by first come first serve basis it is an original and default scheduler, In FIFO scheduler each task are loaded into JOBQUEUE and executed one by one, this type of scheduler is easy to implement but its main disadvantage it's not consider fair sharing of resources.

- **Fair Scheduler:**

This scheduler is designed by Facebook. The aim of designing this scheduler is to obtain fair sharing of cluster resources. Along with fair sharing priority can be assigned to jobs for effective execution this type of scheduler is called priority fair scheduler.

- **Capacity scheduler:**

This scheduler is designed by Yahoo. This scheduler is designed in such a scenario where number of users are large and system needs to analyze the usage of clusters

- **Delay Scheduling:**

This scheduler is designed to improve the locality rate of the Hadoop cluster.

3 SIZE BASED MULTITHREADED SCHEDULER

This section presents the new technique for enhancing the scheduling mechanism of Hadoop. The main idea behind this system is to enhance the performance of data processing in case of big data with parallel processing using thread and synchronization mechanism. The scheduler which are provided by Hadoop by default cannot be applicable in some scenario For example, in FIFO scheduler[15], small jobs have a problem in waiting large job processing. FIFO also does not respect data locality for jobs that are needed in Map-Reduce scheduling framework. Fair share scheduler demands more time for job scheduling in context switch between jobs. Capacity scheduler does not respect data locality. And finally Dynamic priority scheduler interests to achieve special goals. So, this scheduler does not respect data locality too. Many other schedulers are also designed now a days for the specific purpose only so there scope is also limited to their application only. Most of these schedulers suffer various problems like data locality, synchronization, fairness etc.

The proposed system is based on the data present on the DataNode and processing of that data individual thread is created. So the multithreaded approach for processing the individual nodes data will definitely improves the system performance. If the processing data size is less than it can be done working in one thread and if the processing data size is more then it will create thread accordingly.

3.1 System Architecture

To evaluate data size based multi threaded scheduling algorithm its results are compared with the Hadoop default FIFO scheduler. proposed system is tested on private cluster of 1 master node and 4 slave nodes. It takes in to consideration of size of the data available in name node and according to that job tracker will create the treads. In proposed system input job is submitted to the Task Tracker. As each task tracker runs on Data Node, according to the data available in DataNode it divides that data in to blocks using threshold value.

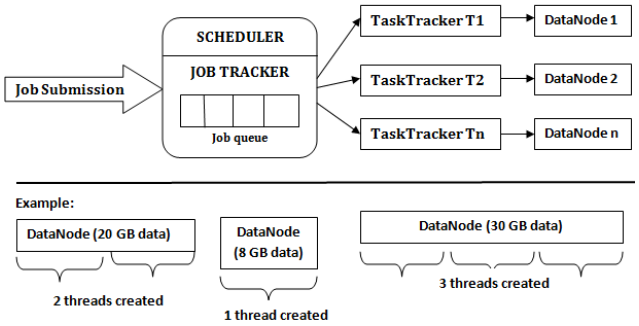

Figure 2: System Architecture

Example: Suppose any random DataNode in cluster having 20gb of data and threshold value is 3gb then the bocks splitting is using sizeOfData/thresholdValue, in this scenario 7 blocks will be created out of that 6 will be of size 3gb and one is of 2gb. For each block processing thread is created separately that independent thread will perform Map/Reduce operation on the data block.

3.2 Algorithm

Algorithm 1: Size based multi threaded algorithm

Thread creation:
Tnumbers= Data.size/maximumsizeofdata;
Thread[n] treads =Thread.create(Tnumbers);
for each thread of threads{
if size!=0{
size=size-maximumSizeOfData;
dataToProcess=Data.size;
process(dataToProcess);
}
}
process(dataToProcess){
// pass data to map reduce
// Process data
//Store data
synchronize(){//Store result}
}

3.3 Experimental Environment

To evaluate proposed scheduling performance private cluster of 1 name node and 4 data nodes is used. The respective hardware environment and configuration of these is shown in table 1.

Table1: Evaluation Environment

Nodes	Quantity	Hardware and Hadoop configuration
Master Node	1	2 single-core 2.2GHz Optron-64 CPUs, 8GB RAM, 1Gbps Ethernet
Slave Nodes	4	2 Single core 2.2Ghz Optron-64 CPUs, 4GB RAM, 2 map and 1 reduce slots per node.

3.4 Associated challenges

A scheduler plays a very important role in the big data processing, because of big volume of data fast processing of data becomes the big challenge. System may have to face Some other challenges like:

- Handling Data Synchronization in critical region e.g. Banks Sensitive Data or account balance
- Data Locality
- Handling data Dependency

4 SET THEORY

Input: Large data set.

Output: scheduling time require in the form of graphs.

Function: input, split data, scheduling, synchronization, data processing, result, graphs.

Hadoop function: map, reduce.

Constraints:

1. To observe performance of the proposed system it is required to use data in high volume.

2. Data set provided should be candidate for parallel processing i.e. data should not be transactional.

Assumptions & Dependencies: system will enhance the performance of data processing in case of big data with parallel processing using thread, synchronization mechanism.

5 RESULTS & DISCUSSION

Screen 1: RUN JOB USING FIFO SCHEDULER

The screen shows the result of job processing, the system output is shown on the localhost the output screen shows the user name, job name, job file which is stored on the data node, host name and address, and the important status i.e. in the form of SUCCESS and FAILURE status code which shows the job running status, job starting and finished time is also shown the output window showing the job finishing time if the job is running with traditional Hadoop FIFO scheduler which are compared with proposed systems scheduling time.

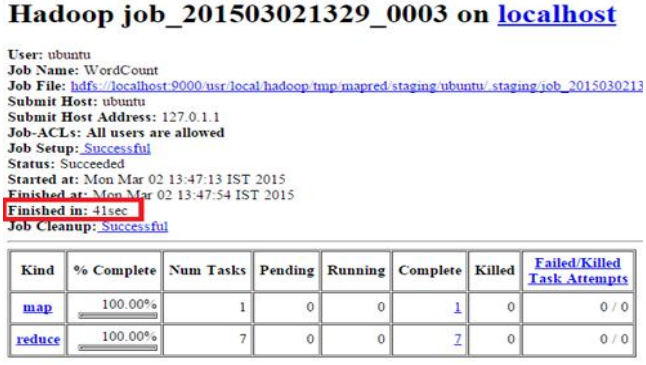

Figure 3: Run Job Using Proposed Scheduler

Screen 2: RUN JOB USING FIFO SCHEDULER

The output window showing the job finishing time which are compared with running the same job on traditional Hadoop FIFO scheduler shown in above figure.

Hadoop job_201503021329_0002 on localhost

User: ubuntu
Job Name: WordCount
Job File: hdfs://localhost:9000/usr/local/hadoop/tmp/mapred/staging/ubuntu/.staging/job_201503021
Submit Host: ubuntu
Submit Host Address: 127.0.1.1
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Mon Mar 02 13:33:11 IST 2015
Finished at: Mon Mar 02 13:34:18 IST 2015
Finished in: 1mins, 6sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	10	0	0	10	0	0 / 0

Figure 4: Run Job Using Fifo Scheduler

Proposed scheduler use concept of multithreading where file data is getting processed in parallel. Number of threads gets created based on the size of the data and processing logic runs in parallel. This enables maximum utilization of the resources and to achieve better result

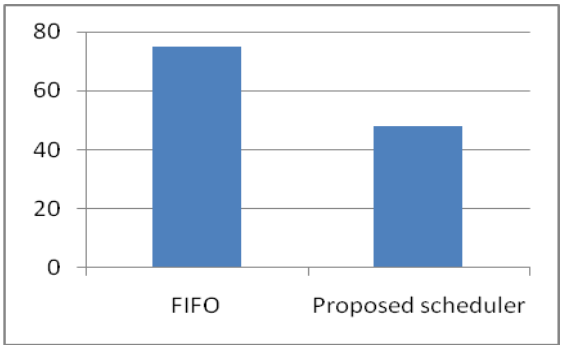

Figure 5: comparative analysis between traditional FIFO scheduler and Proposed Scheduler

6 CONCLUSION

Hadoop scheduler resource aware is one of the emerging research problems that grab the attention of most of the researchers as the current implementation is based on statically configured slots. There are various pros and cons of previous Scheduling policies which are developed by different communities. Each of the Scheduler considers the resources like CPU, Memory, Job deadlines and IO etc proposed system provides gives faire scheduling of task and give better and fast performance. In proposed scheduler the concept multithreading is used where file data is getting processed in parallel. Number of threads gets created based on the size of the data and processing logic runs in parallel. This enables maximum utilization of the resources and to achieve better result.

7 REFERENCES

[1] Apache Hadoop. Available at http://hadoop.apache.org

[2] ApacheHDFS.Available at http://hadoop.apache.org/hdfs

[3] ApacheMapReduceAvailableathttp://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[4] Apachefairescheduler.Availableathttp://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html

[5] ApacheCapacityscheduler.Availableathttp://hadoop.apache.org/docs/r1.2.1/capacity_schedulerhtmlJournal of Computational Information Systems 7: 16 (2011) 5769-5775 Available at http://www.Jofcis.com "Research on Job Scheduling Algorithm in Hadoop" by Yang XIA, Lei WANG

[6] A community white paper developed by leading researchers across the United States "*Challenges and Opportunities with Big Data*"

[7] Jeffrey Dean and Sanjay *Google, Inc." MapReduce: Simplified Data Processing on Large Clusters"*

[8] KyuseokShimSeoulNationalUniversityshim@ee.snu.ac.kr "*MapReduce Algorithms for Big Data Analysis"*

[9] Vasiliki Kalavri, Vladimir VlassovKTH The Royal Institute of Technology Stockholm, Sweden kalavri@kth.se "*MapReduce: Limitations, Optimizations and Open Issues*". TrustCom/ISPA/IUCC,Page1031-1038,IEEE,(2013)

[10] Yi Yao, Jianzhe Tai, Bo Sheng, Ningfang Mi, "LsPS: A Job Size-Based Scheduler for Efficient Task Assignments in Hadoop", In proceedings of the IEEE transaction, Copyright (c) 2014 IEEE

[11] Qutaibah Althebyan , Omar ALQudah, Yaser Jararweh Qussai Yaseen "Multi-Threading Based Map Reduce Tasks Scheduling", 2014 5th International Conference on Information and Communication Systems (ICICS)

[12] Jisha S Manjaly, Varghese S Chooralil Department "TaskTracker Aware Scheduling for Hadoop MapReduce" 2014 5th International Conference on Information and Communication Systems (ICICS)

[13] Runhui Li, Patrick P. C. Lee, Yuchong Hu "Degraded-First Scheduling for MapReduce in Erasure-Coded Storage Clusters" AoE/E-02/08 and ECS CUHK419212 from the University Grants Committee of Hong Kong, IEEE 2013

[14] Bin Ye, Xiaoshe Dong, Pengfei Zheng "A delay scheduling algorithm based on history time in heterogeneous environments" 2013 8th Annual ChinaGrid Conference

[15] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysisof traces from a production mapreduce cluster," in CCGRID'10,2010, pp. 94–103.