MapReduce Scheduler with Classification Algorithm to Enhance Hadoop Performance

Mohammad Ghoneem
Department of Information Technology
Pune University, Pune, India
Mhd89aiu@gmail.com

Prof. Lalit kulkarni
Department of Information technolgy
Pune university,Pune, India
lalit.kulkarni@mitcoe.edu.in

Abstract

MapReduce paradigm is considered one of the most important and powerful technique in Hadoop and cloud computing processing. MapReduce has a master slave architecture in which master node split each job into multiple small tasks known as map tasks [1]. After the successful execution of the map tasks at the slave nodes the results are transferred to another slave node which will execute the reduce task and return the final result to the master node [5].

Scheduling of tasks is very important in order to achieve high efficiency and improve the resources utilization in heterogeneous environment. The proposed work incorporates a design of a scheduler that will not overload any TaskTracker in the cluster, and prevent any re-launch of a task due to failure. The scheduler classifies the jobs into executable and non-executable according to each node capabilities.

Keywords: Hadoop, MapReduce, Linear Classifier, distributed processing, Heartbeat message, JobTracker, TaskTracker.

Introduction

In today's world with the evolution in soft technology people start depending on different application in many fields such as commercial and scientific to do their day to day work. These applications start seeking for more and more processing capabilities which can't be provided by a single machine. One of the potential solution to this problem was distributed and parallel processing which can be found as a MapReduce [1] function proposed by Google in 2004.

MapReduce can be found as an open source implementation in Hadoop. Hadoop MapReduce follows master slave architecture [4] in which a user submits a query to master node in the cluster. The master node then splits this query into sub queries known as map functions and assigns these map functions to the slave nodes which contain the block of data to be processed. Each slave node with map function process the data accordingly and return a result known as intermediate result [5]. Here the first phase of MapReduce is completed. The second phase start when master node get a notification from all slave nodes that map function is done by sending heart beat message [4]. Then the master node assigning a reduce function to n number of slave nodes. The slave node then start merging the intermediate results and apply the reduce function to get the final result.

This model can be enhanced by improving the scheduling technique and assignment of the tasks in the cluster. This can be done by making the scheduler aware about the resources in the cluster and the job requirement to be successfully executed.

Hadoop Scheduling Technique

In Hadoop master slave architecture the master node must execute a program known as a job Tracker (JT) while the slave nodes must run a program known as task tracker (TT) [4]. These two programs provide the communication between master and slave nodes in the cluster [1]. A heartbeat message is used by the task tracker in the slave node to communicate with

the master node (JT). The heartbeat message might contain information about the slave node indicating the state of the task (map or reduce) running at that node [7].

The heartbeat message [4] is also used by slave nodes to send to send a request to master node to get a task (map or reduce). Then the job tracker at the master node replies by assigning a task to that task tracker at the salve node. When job tracker receive a request from the slave nodes it try to schedule the map task according to their vicinity of input split [8], but in case of reduce task job tracker assign it to the slave nodes without taking any consideration to size and location of data blocks that need to be transferred in the network.

Different companies have introduced different schedulers for MapReduce according to their specifications. The default Hadoop scheduler follows FIFO rules in assigning tasks in the cluster. This scheduler provided by Apache Hadoop [6]. Facebook has developed fair scheduler which tries to share resources equally among the users. Another scheduler is capacity scheduler which introduced by yahoo [6]. The aim of this scheduler is to serve a large number of users.

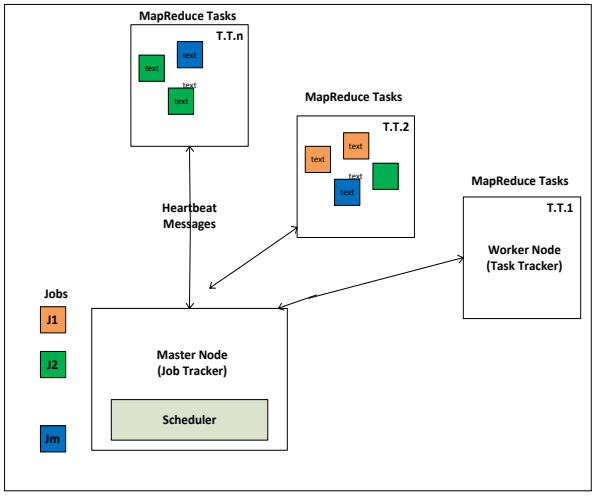

Figure 1: MapReduce Model

All of these schedulers perform well in case of homogenous cluster. But in case of heterogeneous environment these schedulers might degrade the performance for two reasons. The first reason is the variation in nodes capabilities such as processing power and RAM size. The second reason is the different requirement of different applications.

For these reasons the scheduler might overload the nodes in the cluster which lead to task failure and relaunch of the tasks. Also the scheduler will not fully utilize the resources in the cluster due to different applications requierments.

One of the emerging problems in big data is to make scheduler aware about node capabilities in order to fully utilize resources in the cluster. This can be done by providing the scheduler with a classifier. The input for the classifier will be the node capabilities and job requirements to be executed successfully. The output will be the job is executable or non-executable at a specific node.

Proposed Implementation

To implement the proposed module first we have to obtain the list of jobs submitted to the master node. The next step is to provide the classifier with tasks which are ready to be submitted to the slave nodes. There are three points to be covered in this implementation:

- Support Vector Machine classifier (SVM) which is used to classify tasks into executable and non-executable.
- How to maintain node properties and send them to master node so that it can be used by the classifier during the classification procedure [9].
- Get job requirement to be executed successful at the slave node [8].

We are using a SVM classifier which is a linear classification algorithm. This classifier is computationally efficient so it will not over load the master node during the execution. One more advantage is that the algorithm considered as online algorithm. This means that the classifier can receive the input at a real time which will improve the accuracy of the classification. Also the algorithm is suitable for problems with many irrelevant features.

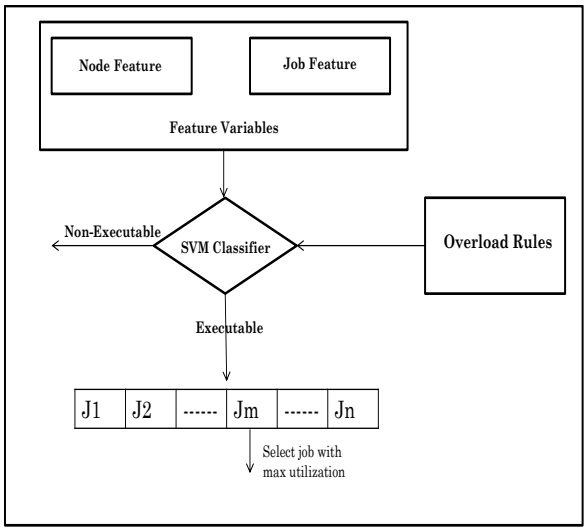

Figure 2: Scheduler Implementation

In order to integrate this algorithm into Hadoop fework we have to update the *assign task* method in the *org.Apache.hadoop.mapred.jobqueuetasks*

Scheduler class. The first input to the algorithm is the node properties such as processing power, RAM size, and storage. To get this information we have to use a distributed monitoring tool known as ganglia. This tool should be installed at each node in the cluster including the master node. Then we should configure the tool to get required information. After that each instance at each node will maintain the properties of the node and send this information to the master node as shown in figure 3. Then the master node will provide the classifier with this information as the first input.

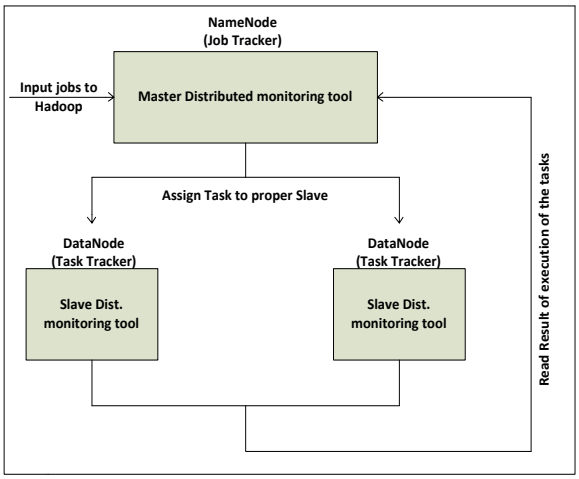

Figure 3: Distributed monitoring tool

The second input to the algorithm is job requirement as shown in figure 2. One of the important features of Hadoop jobs is that all of these jobs are repetitive in nature. Also the master node keeps a log of all previous execution of the jobs [8]. So we can maintain the job requirement form the logs at the master node. But in case if a job running for the first time then the job requirement can be chosen randomly. Here the first phase of the classification ends.

In the second phase after the classification is done and the tasks are assigned the execution of the tasks are monitored. After that the input to the classification algorithm is updated according to the execution state of previous running tasks.

Mathematical Model

Tasks of good jobs do not overload resources at the TaskTracker during their execution. Jobs labeled bad are not considered for task assignment. If the classifier labels all the jobs as bad, no task is assigned to the TaskTracker. If after classification, there are multiple jobs belonging to the good class, then the task of a job is chosen that maximizes the following quantity:

$$E.U.(J) = U(J)P(J = good|F1, F2, ., Fn) \quad (3)$$

where, E.U.(J) is the expected utility, and U(J) is the value of utility function associated with the MapReduce job J. J denotes a task of job J, and P(J = good | F1, F2, . . . , Fn) denotes the probability that the task J is good. The probability is conditional upon the feature variables F1, F2, . . . , Fn. Feature variables are described in more detail later in this section. The cluster is assumed to be dedicated for MapReduce processing, and that the JobTracker is aware and responsible for every task execution in the cluster. The proposed scheduling algorithm is local as it considers the state of only the concerned TaskTracker while making an assignment decision. The decision does not depend on state of resources of other TaskTrackers. The assignment decisions are tracked. Once a task is assigned, effect of the task is observed from information contained in subsequent heartbeat from the same TaskTracker. If based on this information, the TaskTracker is overloaded; it is concluded that last task assignment was incorrect.

The pattern classifier is then updated (trained) to avoid such assignments in the future. If however, the TaskTracker is not overloaded, then the task assignment decision is considered to be successful.

Cluster configuration

The cluster will consist of three nodes. One of these nodes is configured to be the master node. The other two nodes are configured to be slave nodes as shown in the table.

Master node	
Hard disk	100 GB
RAM	1 GB

Table 1: Master node configuration

Slave node1		Slave node 2	
Hard disk	75 GB	Hard disk	50 GB
RAM	1 GB	RAM	512 MB

Table 2: Slave nodes configuration

In our Hadoop cluster block size will be 64 MB as it is in default Hadoop. The number of replicas in the cluster will be two because we have only two slave nodes. The heartbeat message interval will be 5 seconds.

Hadoop Configuration	
Replication	2
HDFS block size	64 MB
Heartbeat interval	5 seconds

Table 3: Hadoop Configuration

Analytical results

In Hadoop cluster the number of failure tasks depends on types of applications being served and some parameters in default Hadoop configuration. One of the parameters is *mapreduce.job.maxtaskfailure.per.tracker*. This value limits the number of the re-execution of task at node. Another value is *mapreduce..max. map.failure.percent*. This value limits the number of map tasks failure in the cluster.

Initially, our SVM classifier weights are set to zero. These parameters will be changed and updated during the learning phase of the classifier. In the first iteration of the learning phase all tasks will be considered as executable tasks [7]. Let the memory usage be the over load rules for our classifier. If the utilization of the memory is greater than 70% percent, then the node will be considered overloaded. After the first execution of the tasks the weights should be updated according to the overload rules.

After the learning phase, the weights will not be updated until one of the tasks overload the node which is previously labeled as executable.

tasks	CPU usage	RAM usage
Task1	40%	267 MB
Task2	60 %	430 MB
Task3	30%	555 MB
Task4	90%	670 MB
Task5	55%	122 MB
Task6	10%	30 MB

Table 4: shows the resources consumption by different jobs.

During the execution the scheduler should be provided by the node capabilities and tasks requirements to be executed successfully. At the first iteration all the tasks will be labeled as executable. When the learning phase ends the weights of tasks are updated accordingly.

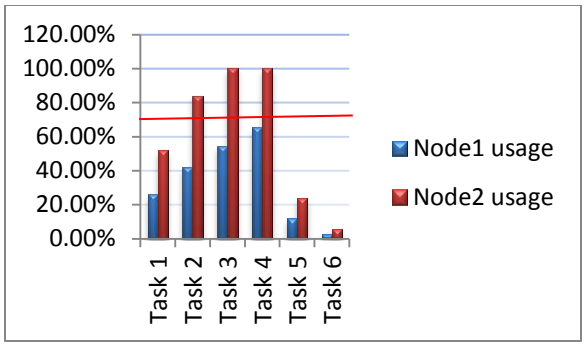

Figure 4: Memory usage of Slave Nodes

While the framework is in running the slave node1 and the slave node2 will send a request to master node to get some tasks to be serviced. When the master node receives the requests it provides the classifier with the required information about the nodes and tasks and wait for the result. If we consider a 70% usage of the RAM as the overload rules for the slave nodes, the output at the classifier will be as follows:

At the first request for the slave node1 with 1GB RAM all the tasks will be classified as executable, because none of the tasks will consume more than 70% of RAM as shown in figure 4. At the slave node 2 with 512MB RAM the tasks number 1, 5, and 6 will be considered as executable, since ream consumption is less than 70 %. The tasks number 2,3, and 4 will be classified as none-executable because it consumes more than 70% of RAM.

Node 1		Node 2	
Exe	Non-Exe	Exe	Non-Exe
Task 1		Task 1	
Task 2			Task 2
Task 3			Task 3
Task 4			Task 4
Task 5		Task 5	
Task 6		Task 6	

Table 5: Tasks Classification

Conclusion and Future Directions

Hadoop provides for the needs of a wide variety of possible users but does not provide a means to optimize the scheduler for individual users. The proposed scheduler for MapReduce will not overload any TaskTracker at point of time. Thus the burden of re-launching the tasks at different TaskTrackers is not necessary. This scheduler classifies the job into IO bound and CPU bound jobs. So, there will be a balance between the number of IO bound tasks and the number of CPU bound tasks running at every TaskTracker. This increases the hardware resource utilization.

In future, we plan to test the system on a much larger scale in order to observe any changes and optimize the code accordingly. In addition, there is a possibility of improving the accuracy of the classification further by taking into account the health of a node. The health of a node is based on the efficiency at which the node is running as well as the probability of the node failing in the near future.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating Systems, Design & Implementation, OSDI*, 2004.

[2] Marwa Elteir, Heshan Lin and Wu-chun Feng, 'Enhancing MapReduce via Asynchronous Data Processing,' *IEEE* International Conference on Parallel and Distributed Systems.

[3] Mohammad Hammoud and Majd F. Sakr, 'Locality-Aware Reduce Task Scheduling for MapReduce,' 2011 Third IEEE International Conference on Cloud Computing Technology and Science.

[4] Shamil Humbetov, 'Data-Intensive Computing with Map-Reduce and Hadoop,' 2011 Third IEEE

International Conference on Cloud Computing Technology and Science.

[5] Kyong-Ha Lee , Yoon-Joon Lee , Hyunsik Choi , Yon Dohn Chung , Bongki Moon, "Parallel data processing with MapReduce: a survey", ACM SIGMOD Record, v.40 n.4,pp:11-20 December 2011.

[6] Thirumala Rao and Dr. L.S.S. Reddy,"Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments".

[7] Dhok J, Varma V (2010), Using pattern classification for task assignment in MapReduce.

[8] Dr.J.Aghav and Shyam Deshmukh (2013), Job Classifivation for MapReduce Scheduler in Heterogeneous Environment.

[9] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In Proc. Of USENIX OSDI, 2008.

[10] K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines", in Proc. CloudCom, 2010, pp.388-392.

[11]Apache Hadoop. http://hadoop.apache.org .

[12] Ganglia. http://www.ryangreenhall.com (20/9/2014) monitoringhadoopclusters-using-ganglia.