Job Classification for MapReduce Scheduler in Heterogeneous Environment

Shyam Deshmukh
Student, Department of Computer and Information Technology, College of Engineering (COEP), Pune,India
dshyam100@yahoo.com

Dr. J. V. Aghav
Professor, Department of Computer and Information Technology, College of Engineering (COEP), Pune,India
jva.comp@coep.ac.in

Rohan Chakravarthy
Student, University of Pune, Pune, India
rohan.chakravarthy@outlook.com

***Abstract*— Hadoop MapReduce is one of the most popular publicly available frameworks for big data processing in the cloud environment. 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. This is what we have attempted to do. The MapReduce scheduling system consists of a single Job Tracker per cluster and multiple Task Trackers. One of the primary responsibilities of the job tracker is to schedule all the user jobs which makes optimizing the scheduling by the Job Tracker an interesting problem. For an improved scheduling framework, we have implemented scheduling which also takes into account the actual resource requirements of the job, as opposed to relying completely on number of free Map and Reduce slots. In this paper, an attempt is made to create a scheduler that can learn and adapt itself to any possible application. The scheduler classifies the tasks to be assigned into two classes, schedulable and non-schedulable. This process weeds out jobs that are unlikely to run on a node using a process that is computationally cheap. The experimentation result detects the job which will overload a particular node and indicate the same to the scheduler. Thus, if it is highly unlikely that a job will successfully run on a particular node, that job will be classified as non-schedulable and not be considered for execution on that node. This will prevent jobs from executing partially, then failing and having to be rescheduled.**

Keywords*— *MapReduce, Resource Management, Scheduling, Cloud Environment.

I. INTRODUCTION

MapReduce [1], a simple programming paradigm became an obvious choice of many users as they can express relatively complex distributed programs easily[2]. The task assignment to compute-nodes is referred as a mapping. Effective mapping policies must account for various parameters[3], e.g. the set of available compute-nodes in the system, characteristics of these nodes etc. MapReduce uses a block-level runtime scheduling with a speculative execution. In homogeneous computing environment Hadoop [4] has shown a great success., sometimes resulting in even worse performance than with speculation disabled. A Map task is forked to process each data block. As soon as the node complete its task, gets new task to process more tasks. Slow running task on a straggler nodes are redundantly executed on other idle nodes [5], referred as speculative execution. Experimental observations by some research work [6] reveal that the homogeneity assumptions of MapReduce can cause wrong and often unnecessary speculative execution in heterogeneous environments.

Significant amount of research work carried out in developing efficient schedules. The default Hadoop scheduler uses FIFO as the basis for scheduling. The preemptive fair scheduler is developed by facebook which aims at providing users with a fair share of the cluster capacity or time. The capacity scheduler originally developed at yahoo was designed for large number of users. Many researchers worked on improvement in scheduling policies in Hadoop which has resulted in the development of Longest Approximate Time to End(LATE) scheduling, Delay scheduling, Dynamic priority scheduling and Deadline Constraint scheduling. Most of the implementation above are statically configured slots and does not consider resource availability. As the Job tracker simply treats each TaskTracker nodes as having a number of available task "slots" which may not utilizes its resources like CPU, Disk and Memory subsystem appropriately.

Ability to make hadoop scheduler resource aware is one of emerging research problem that grabs the attention of many researchers [7]. Most of the applications, in turn the hadoop jobs, are typically repetitive in nature. This provides us with an opportunity to use the historical performance details of the application from their previous runs and integrate these footprints into resource management algorithms. This information helps us training the JobTracker to schedule the jobs. Any task assignment decision can then be supervised by considering whether a particular job to allocate to a particular node or not.

II. PROPOSED IMPLEMENTATION

As jobs are submitted to the Hadoop cluster, they need to be made ready to run. In other words, they need to be placed into the RUNNING state so that they can be considered for scheduling on available nodes. This part of our system maintains a dynamic queue of jobs in the RUNNING state. These jobs are then submitted to the Perceptron. Our scheduler relies on information about node which is retrieved using the network monitoring tool Ganglia and job features [8]. If a

978-0-4799-2235-2/13 $31.00 © 2013 IEEE
DOI 10.1109/CUBE.2013.15

particular job has run before, its resource utilization is retrieved using the job history database. If it has not run before, a certain level of expected utilization is assumed and assigned to it.

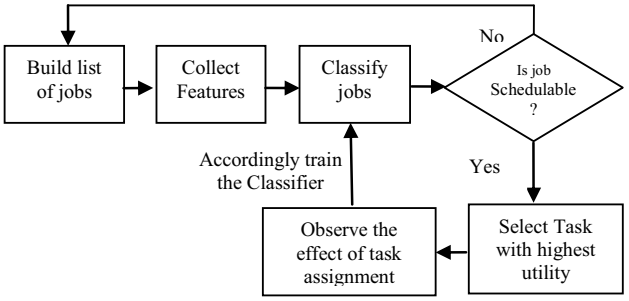

Fig. 1 Proposed Implementation Approach

Once we obtain the node features, we cycle through the job queue and submit the node features and the features of one job to the perceptron. If the perceptron classifies the job as Schedulable, the job is added to the list of Schedulable Jobs. Otherwise, the job is returned to the original job queue. This is repeated for all the jobs in the queue. After all the jobs have been classified, we come to the end of the first stage of classification.

After a job is scheduled on the node, the node is monitored to make sure it does not get overloaded during the execution of the job. If at any point during the execution of the job there is a sustained period of node resources being overloaded, it is understood that the classification of the job as Schedulable was incorrect. In this case, the Perceptron is updated to prevent such mislabeling in the future. The Fig.1 indicates the steps followed for implementation.

III. Implementation and Results

A. Implementation Details

Any linear classifier can be used to implement this algorithm for job classification. We have implemented this algorithm for Hadoop version 0.22.0. The scheduler customizes *assignTasks* method of the class org.apache.hadoop.mapred.JobQueueTaskScheduler. Only Perceptron classifier is used in the implementation. The Perceptron [9] we have implemented, is a single layer neural network algorithm for supervised classification of an input job into a binary output. Specifically, we take the features of a job into account, and classify it as a Schedulable or a Non-Schedulable job. This is explained in more details later. We have used the Perceptron as it computationally less expensive than other comparable algorithms. For example, the Winnow algorithm uses multiplicative updates versus the Perceptron's additive updates. The Perceptron Learning algorithm used accepts the inputs at real time as they are available, and so can be classified as an online algorithm. Node Features are obtained by polling the slave nodes after every heartbeat, through Ganglia. Ganglia [11] is an open source distributed monitoring system for very large clusters. The heartbeat interval is set to 5 seconds in the system. It is designed to impose very low resource overheads on each node in the cluster. Ganglia itself collects metrics, such as CPU and memory usage. By using GangliaContext, you can inject Hadoop metrics into Ganglia. The Ganglia Monitoring Daemon (*gmond*) needs to be installed on each slave Hadoop nodes. The *gmetad* service collects node metrics and exposes them over TCP. We used Ganglia instead of the inbuilt heartbeat function because the inbuilt function did not provide us with all the metrics we required. The reason for the selection of Ganglia as our software of choice was twofold. Most importantly, it had a very low overhead and so its effect of resource utilization metrics was negligible. Secondly, its output could be obtained via an XML stream, which made data extraction convenient.

Administrators can configure overload rules based on their requirements. For example, if a certain percentage of the cpu power needs to be set aside for emergency or high priority jobs, then CPU utilization or load average could be used in deciding node overload. Similarly, if the bandwidth usage should not cross a certain threshold to maintain costs, network usage can also be included in the overload rule. The overload rules supervise the classifier. But, as this process is automated, the learning in the algorithm is also automatically Supervised. The only requirement for an overload rule is that it can correctly identify given state of a node as being overloaded or underloaded based on the available metrics. The overload rule must remain the same during the execution of the system. This is required because the perceptron changes its weights if a node gets overloaded. Assume the rule is changed. The same state of the system which had previously overloaded it, now might not. Such discrepancies will cause massive accuracy issues and prevent the classifier from correctly learning.

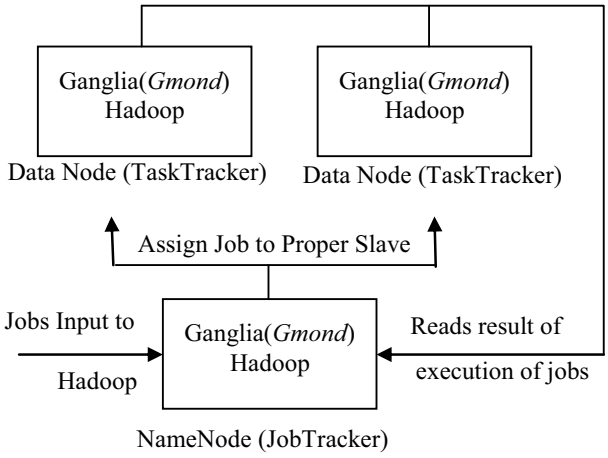

Fig. 2 Deployment Details

Fig.2 shows the node deployment along with the software running on the nodes. Every time the master receives a heartbeat from one of the slave nodes, all the jobs in the job queue are submitted to the perceptron in a FIFO fashion, until a job is classified as Schedulable. A Schedulable job is one which is not likely to overload the job, and a Non-Schedulable one is one which is expected to overload the node. As input,

the perceptron takes the current node features (according to the ganglia poll) and the job features based on job run history. We normalize all the job and node features on a scale of 1 to 10 based on a min-max function. If the same or a similar job has not been run before, then the job features are assumed to be equal to 5 and updated after it is actually run. If a job which is classified as Schedulable for a particular node proceeds to overload the node, then the weights of the perceptron are updated in order to reflect this incorrect classification.

B. Evaluation Methodology and Workload Description

To evaluate the implementation we used 3 VMs. . One of the VMs was designated as the master node which ran the NameNode and the JobTracker. The remaining 2 VMs were worker nodes. Both of the other 2 VMs had single CPU (Intel Quad Core, 2.4 GHz) and a single hard disk of capacity 250 GB. However, 1 of the VMs was configured with a RAM of 2GB and 1 with a lower RAM of 1GB for a reason that will be explained later. All of the nodes had Ubuntu Linux (11.04, Server edition) and SUN Java 1.6.0. We have used our modified version of Hadoop 0.22.0 for this evaluation. The values of important Hadoop parameters are described in TABLE I. All other parameters were set to their default values.

TABLE I. Hadoop settings used in evaluation

Hadoop Parameter	Value
Replication	2
HDFS Block Size	64 MB
Speculative Execution	Disabled
Heartbeat Interval	5 Seconds

A node was considered to be overloaded if the ratio crossed a user specified limit. For overloading jobs, we considered CPU utilization criteria. If CPU utilization by job crosses 80% of available CPU capacity then node is overloaded.

C. Results

In this experiment, we demonstrate the accuracy of the learning of the scheduler. We ran the WordCount MapReduce job provided in Hadoop. The job was run several times on randomly generated texts of size 284MB, 426 MB, 568 MB, 681 MB , 113 MB and 1 MB. The Aim of the scheduler is to prevent any of the nodes from getting overloaded. Initially certain jobs do overload some nodes during the 'learning phase' of the scheduler. In following input sample, we have provided six jobs with different sizes of word count job. We considered these jobs as different type on the basis of size parameter. As the size of the input increases both CPU and Network resource usage increases.

Initially, Perceptron weights are zero, these weights will be updated during the learning phase. In first step, all the inputs provided to the perceptron as input are considered as Schedulable jobs. The overload rule is dependent on CPU utilization. If the utilization of CPU by job is greater than 80% then the node will be overloaded and the weights updated. During the learning phase all the jobs will run and update the weights based on whether or not they overload the node. However, after the learning phase, the weights are updated only if the job overloads the node. Thus, after the learning phase, the first 3 jobs will execute on cluster without updating weights. The job 4 having maximum size and CPU utilization above 8 so it will overload that node from cluster, the perceptron immediately updates the weights of the classifier. Thus, first the job is not scheduled on the node with the lower configuration, but does get scheduled on the other node. This shows that for nodes in different states, the jobs are evaluated separately. However, the job 4 continues to overload the other node when it is scheduled on them. Thus, after a few more runs, it is observed that job 4 is never scheduled on any of the nodes.

TABLE II. Resource usage by Jobs provided in Learning phase

Jobs	CPU	Size
Job1	4	284 Mb
Job2	6	426 Mb
Job3	7	568 Mb

Jobs	CPU	Size
Job4	9	681 Mb
Job5	3	113 Mb
Job6	1	1 Mb

TABLE II shows the resource usage by the different jobs which is considered after running and monitoring them previously. After learning phase, the jobs with different sizes are provided to the perceptron. The jobs are classified into Schedulable and Non-Schedulable on the basis of all features of jobs. The Job features are obtained from previous run history, or with the use of user specified values. The perceptron will retrieve the Node features using Ganglia. The perceptron algorithm combines both features and uses a function which calculates the value of these features whether it will belong to Schedulable or Non-Schedulable class. The function returns a value of zero or one as a output. If the output of perceptron function is zero then the current job is Non-Schedulable for the given node, and the job will not be scheduled on that particular node. It is important to note that this classification is not binding. If the value is one then the perceptron will schedule that job on current node. A node at time t might already be running other tasks and hence would not be suitable for a CPU intensive job and so the job would be classified as Non-Schedulable for the node at time t. However, if at time t+1 the node is idle, then that same job will be classified as Schedulable for the node at time t+1.

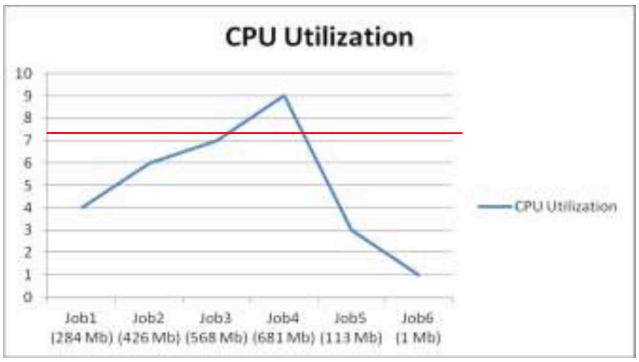

Fig. 3 CPU Utilization by the Jobs with variable sizes

The graph above shows the average CPU utilization by the jobs. The perceptron learns that job 4 consistently overloads the nodes and it will not schedule that type of jobs on the overloaded node. The red line in Fig. 3 above shows the limit set by us as the overload point in the system, i.e. 80% of CPU utilization.

IV. Conclusion and Future Work

We have implemented a system for Job Classification in a MapReduce environment using Perceptron. The key feature of the system is its ability to adapt to heterogeneous workloads. Since heterogeneous workloads are what are encountered in the real world, our solution has multiple practical uses. Previous work in this area [8] has shown that using the Naive Bayes Classifier results in a speedup of task assignment. Our system performs the job of weeding out jobs that are highly unlikely to run on a node without overloading it, so that precious scheduling time is not wasted on tasks that will overload the node, leading to a further speedup and keeping resource utilization within acceptable limits. Currently, we have tested this scheduler on a small cluster of 3 virtual machines with only 2 of them having a TaskTracker to execute the tasks. 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. By maintaining a log of the events and status of nodes leading up to their failure, a pattern can be established. This can be used to predict if a particular node is likely to fail, and avoid considering critical jobs to run on it before the 1st stage of the classifier itself.

References

[1] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters, ”In Communications of the ACM, Volume 51, Issue 1, pp. 107-113, 2008.J.Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon,1892, pp.68–73.

[2] M. Stonebraker et al. MapReduce and parallel DBMSs: friends or foes? Communications of the ACM, 53(1):64–71, 2010.

[3] V. Shestak, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, “Probabilistic resource allocation in heterogeneous distributed systems with random failures,” Journal of Parallel and Distributed Computing, Volume 72 Issue 10, October, 2012, Pages 1186-1194.

[4] Hadoop. http://hadoop.apache.org/.

[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] B.Thirumala Rao et. al. "Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing",Global Journal of Computer Science and Technology, Volume XI, Issue VIII, May 2011.

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

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

[9] Perceptron. http://en.wikipedia.org/wiki/Perceptron.

[10] Ganglia. http://www.ryangreenhall.com/2010/10/monitoring-hadoop-clusters-using-ganglia.

[11] http://slashdot.org/topic/bi/facebooks-corona-when-hadoop-mapreduce-wasnt-enough/.