International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 3, March - 2014

Learning Based Job Scheduling Algorithm Based On Map Reduce Framework

Rohit Attri
Dept of Computer Science and Engineering
SRM University Kattankulathur
Chennai, India

A. Selva Kumar
Dept of Computer Science and Engineering
SRM University, Kattankulathur
Chennai, India

***Abstract*— MapReduce has become a popular paradigm for large scale data processing in the cloud. The sheer scale of MapReduce deployments make job scheduling in MapReduce an interesting problem. The scale of MapReduce applications presents unique opportunity to use data driven algorithms in resource management. We present a learning based scheduler that tries to allocate a task on a node if the incoming task does not affect the tasks already running on that node.We propose an approach which tries to find a good mix of jobs for each worker node, and in turn decrease their runtime. In our model, the scheduler is made aware of different types of jobs running on the cluster. From the list of available pending tasks, our algorithm selects the one that is most compatible with the tasks already running on that node. We bring up machine learning based solution to our approach and try to maintain a resource balance on the cluster by not overloading any of the nodes, thereby reducing the overall runtime of the jobs.**

Keywords: Job Scheduling, Machine Learning, MapReduce

I. INTRODUCTION

Since its introduction, MapReduce [4] has become a standard programming model for large scale data analysis. It has seen a tremendous growth in recent years especially for text indexing, log processing, web crawling, data mining, machine learning etc [3]. MapReduce is mainly used for batch oriented jobs which tend to run for hours to days over a large dataset on limited resources of the cluster. This makes Job scheduling in MapReduce is an interesting problem, because efficient job scheduling can significantly lower runtime, or improve resource utilization. Both of the improvements result in reducing costs maintaining the cost savings that the providers expect.

Also Clouds are created to provide services to users; therefore providers have to compensate for sharing their resources and capabilities [1]. Because these computing resources are limited, efficient resource allocation algorithms for the cloud platforms are required. Efficient resource allocation would help in reducing the number of virtual machines used and in turn reduce the carbon footprint leading to a lot of energy saving [2]. Scheduling in MapReduce is analogous to this problem. The scheduling algorithms need to be designed in a more intelligent way to avoid overloading any node and utilize most of the resources on a particular node. Thus, the runtime of the jobs could be lowered to a greater extent leading to a lot of energy saving. This paper deals with scheduling of jobs on MapReduce cluster without degrading their runtime while still maintaining the cost savings expected by the users.

In this paper, we propose an approach that takes into account the compatibility of MapReduce tasks running on a node of the cluster. The algorithm ensures that the performance of an already running task will not be affected by a new task. For this, the scheduler should be aware of the resource usage information of each task running on the cluster. We present a machine learning based approach for job scheduling. The scheduling algorithm selects a task from the list of pending tasks that is most compatible with the tasks already running on the node.

II. KEY CONCEPTS IN HADOOP

To better understand our approach and the limitations of current Hadoop schedulers, we now explain the key concepts involved in Hadoop scheduling. Hadoop's MapReduce implementation borrows much of its architecture from the original MapReduce system at Google[4]. Figure 1 depicts the architecture of Hadoop's MapReduce implementation. Although the architecture is centralized, Hadoop is known to scale well for small (single node) to very large (up to 4000 nodes) installations [5].

Scheduling decisions are taken by a master node (Job-Tracker), whereas the worker nodes (TaskTrackers) are responsible for task execution. The JobTracker keeps track of the heartbeat messages received periodically from the TaskTrackers and uses the information contained in them while assigning tasks to the TaskTracker. If a heartbeat is not received from a TaskTracker for a specified time interval, the TaskTracker is assumed to be dead. In such a case, the JobTracker re-launches all the incomplete tasks previously assigned to the dead TaskTracker. Task assignments are sent to the TaskTracker as a response to the heartbeat message. The TaskTracker spawns each MapReduce task in a separate process, in order to isolate itself from faults due to user code in other tasks.

Fig. 1. shows the architecture of MapReduce in Hadoop. The scheduler runs at JobTracker (Master Node) which schedules the tasks of a particular job on various TaskTrackers (TT).

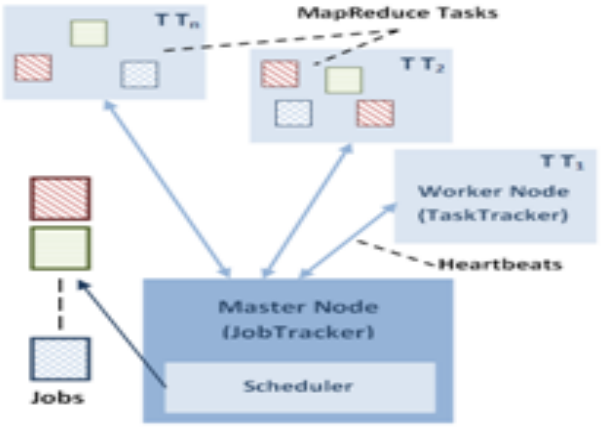

Fig. 1. Architecture of MapReduce in Hadoop

III. PROPOSED ALGORITHM

Our algorithm runs at the JobTracker. Whenever a heartbeat from a TaskTracker is received at the JobTracker, the scheduler chooses a task from the MapReduce job that is most compatible with the already running tasks. This algorithm aims to maintain stability at node and cluster level through clever scheduling of the tasks. The uniqueness of this scheduler is its ability to take into account the resource usage pattern of the job before its tasks are scheduled on the cluster.

A task can be broadly classified into cpu-intensive, memory-intensive, network-intensive and disk-intensive based on its resource usage pattern. A task should be categorized as a weighted-linear combination of parameters from each of these categories that will describe the true nature of the tasks. The complete nature of the task is represented through its TaskVector T_v as given below

$$\boldsymbol{T_v = T_{cpu} + T_{mem} + T_{disk} + T_{nw}} \qquad (1.)$$

Where T_x (x is cpu, mem, disk, nw) is a resource usage pattern for cpu, memory, disk and network of a particular job respectively.

Whenever the JobTracker (master node) receives an incoming job through the client, it queues the job into Pending Job Queue (J).The Task Selection algorithm takes the pending jobs from J and split it into map and reduce tasks.

For scheduling any task of a particular job, the algorithm calculates the TaskVector for the task. Each task has its own Map-TaskVector ($T_{k\text{-}map}$) and Reduce-TaskVector ($T_{k\text{-}reduce}$) that needs to be calculated. To calculate TaskVectors(Map-TaskVector ($T_{k\text{-}map}$) and Reduce-TaskVector ($T_{k\text{-}reduce}$)), the algorithm uses an event capturing mechanism on the TaskTrackers which listens to events related to memory, disk, network and cpu through 'atop' utility[6] to monitor resource usage patterns of that task and creates a TaskVector. The algorithm queues the map/reduce tasks into the TaskQueue. This queue is filled up with the tasks asynchronously as the new jobs arrive at the JobTracker. This task queue acts as the input to the algorithm. The algorithm picks up the task $task_k$ in FIFO manner. This $task_{k\ is}$ is submitted to the Task Assignment algorithm along with TT details to be approved for its scheduling.

The task assignment algorithm decides if a task can be assigned on a particular TaskTracker or not. The main aim of this algorithm is to avoid overloading any of the TaskTrackers by meticulous scheduling of only compatible tasks on a particular node. Compatible task means task that does not affect already running tasks on that node. We present a machine learning approach for compatibility checking.

A. Compatibility Checker using Machine Learning

A machine learning based approach of the algorithm is presented for checking the compatibility of incoming task on a particular TaskTracker. An automatically supervised *Incremental Naive-Baye's* classifier is used for this puspose. Whenever the TaskSelection algorithm checks for the compatibility of a task on a TaskTracker, the algorithm computes the compatibility through the outcome of the classifier. A Naive-Bayes classifier is a simple probabilistic classifier based on applying Baye's Theorem with strong (naive) independence assumptions. The classifier is trained on the basis of Hardware specifications of TaskTrackers (Φ), TaskVector of Incoming Task (T_v), and TaskVectors of tasks running on the TaskTracker ($T_{compound}(i)$).

$$\boldsymbol{T_{compound}(i) = T_1 + T_2 + \ldots + T_n} \qquad (2.)$$

where n is the number of tasks currently running on the TaskTracker.

1. TaskAssignment(task, TT)
2. Φ = getHardwareSpecifications(T)
3. Σ = getNetworkDistance($task_k$, TT)
4. T = getTaskVector($task_k$)
5. $T_{compound}$=getVectorsOfTasksOnTaskTracker(TT)
6. compatibility = classifier($task_k$, {Φ, Σ, T_v , $T_{compound}$ })
7. if compatibility ≥ C_{ml} then
8. return TRUE
9. else
10. return FALSE
11. end

Fig. 2. The Task Assignment Algorithm based on the Machine Learning Approach

$task_k$ = compatible denotes the event that the $task_k$ would be compatible with the other tasks running on *TT*. The probability P($task_k$ = compatible|F) is conditional on the feature set *F*. The classifier uses the prior knowledge accumulated to make decisions for the compatibility of a task. To achieve this, the posterior probability $P(task_k = compatible|F)$ is computed using Bayes theorem:

$$\boldsymbol{P\ (task_k = compatible\ |\ F\ ) = (P(F|task_k=compatible) \times P\ (task_k = compatible)) / P(F)} \qquad (3.)$$

The quantity P (F |$task_k$ = compatible) thus becomes:

$$\boldsymbol{P\ (F\ |task_k = compatible) = P\ (f_1\ , f_2\ , ..f_4\ |task_k = compatible)} \qquad (4.)$$

where f_1 , f_2 , ..f_4 are the features of the classifier ({Φ, Σ, T k , T compound (i)}). Assume that all the features are independent of each other (Naive-Bayes assumption). Thus,

$$P(F|task_k=compatible)= \prod_{j=1}^{4} P(f_j \mid task_k = compatible) \quad (5.)$$

The above equation is the foundation of learning in the classifier. The classifier uses results of the decisions made in the past to make the current decision. This is achieved by keeping track of past decisions and their outcomes in the form of posterior probabilities. If this posterior probability is greater than or equal to the administrator configured Minimum Acceptance Probability C_{ml} , then the $task_k$ is considered for scheduling on T T_i (algorithm). The overload rule can be configured based on the user requirements. For example, if the jobs are known to be memory intensive, then memory utilization can be used in deciding node overload.

The overload rules are used to supervise the classifier. But learning in our algorithm is automatically supervised as this process is completely automated. The overload rules are only required so that they can correctly identify given state of a node as being overloaded or underloaded.

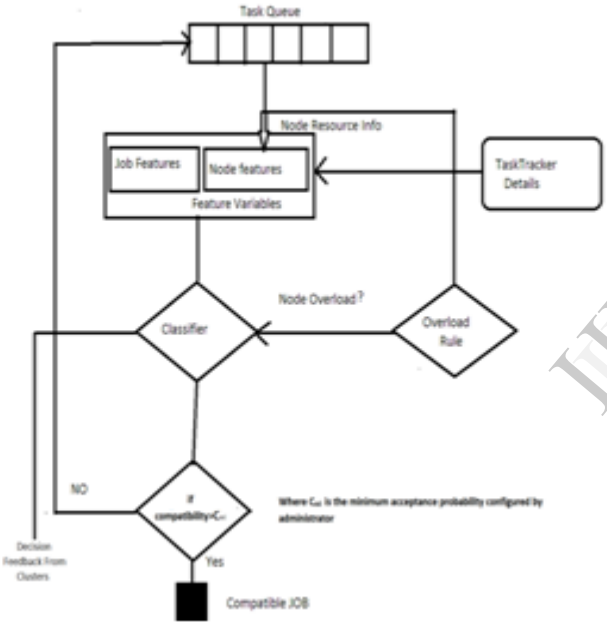

Fig. 3. Task Selection algorithm: The received task is tested for compatibility on a Incremental Naive-Baye's classifier

IV. EVALUATION AND RESULTS

We now discuss the implementation, discuss the results of the experiments conducted and compare the results with existing Hadoop scheduler. The scheduler used as a baseline for testing is Yahoo's Capacity scheduler[7] which has minimal resource awareness.

A. *Testing Environment*

The testing environment comprises of 16 nodes(1 master and 15 slaves). One of the node was designated as the master node which ran JobTracker and NameNode, whereas the remaining 15 nodes were worker nodes which ran TaskTrackers and DataNodes. The nodes were interconnected with a gigabit Ethernet switch. All of the nodes had Intel Quad Core, 2.4 GHzCPUs, with a hard disk capacity ranging from 160 GB to 1 TB, and 4 to 8 GB RAM. All of the nodes were installed with Ubuntu Linux 12.04v OS and JAVA 1.7.0_6.

B. *Experiments Description*

Table 1 provides the the Hadoop and algorithms description for testing our algorithm. Experiments were conducted on jobs like wordcount, Maximumtemperature, file conversions and terasort .

HDFS Block Size	64 MB
Heartbeat Interval	3 sec
Speculative execution	Enabled
Number of map slots per node	4
Number of reduce slots per node	2
Replication Factor	3
Number of queues in Capacity scheduler	3
Cluster resources allocated for each queue in Capacity scheduler	33.3%
Maximum Acceptance Probabilty	0.50

Table1. Hadoop And Algorithm Parameters

To compare our algorithm with the capacity scheduler, we have based our experiments on runtime of jobs and resource usage. Fig. 4 shows the comparison on the basis of runtime of the jobs between the capacity scheduler and machine learning algorithm. The overall runtime of the jobs is compared by varying the number of input jobs submitted to the scheduler. The experiments are expected to provide approximately 25% saving in overall runtime in machine learning approach when compared to Capacity scheduler. The amount of savings also depends on the number of jobs given to the cluster. As the number of jobs increase, the savings in the overall runtime also increase.

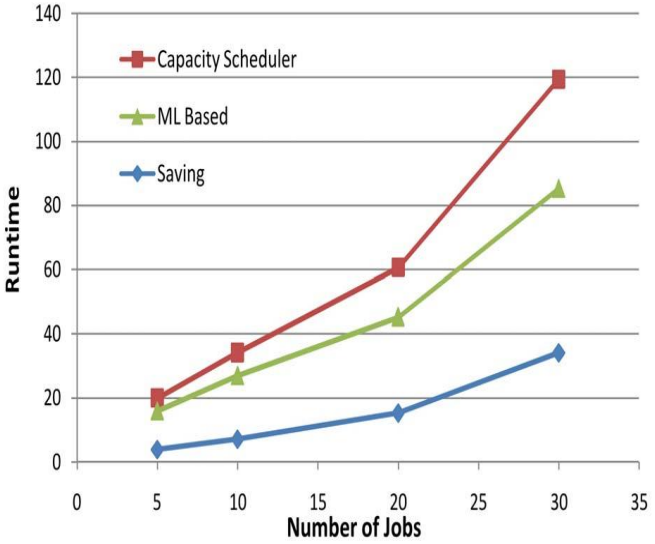

Fig. 4. Comparison of runtime(in hours) of the jobs between Capacity Scheduler and Machine learning based algorithm.

We also present the effect of the algorithm on the basis of cpu usage of the TaskTracker. A random TaskTracker was monitored for its resource requirements. Figure 4 shows comparison of cpu requirement on a TaskTracker between capacity scheduler and machine learning based scheduler.

From figure 5, we can deduce that the cpu requirements for the tasks running on the TaskTracker with capacity scheduler reaches upto 250% (resource requirement but not resource usage) whereas the cpu requirement remains below 100% in case of machine learning based algorithm except surges. This is because of the un-awareness of jobs in capacity scheduler.

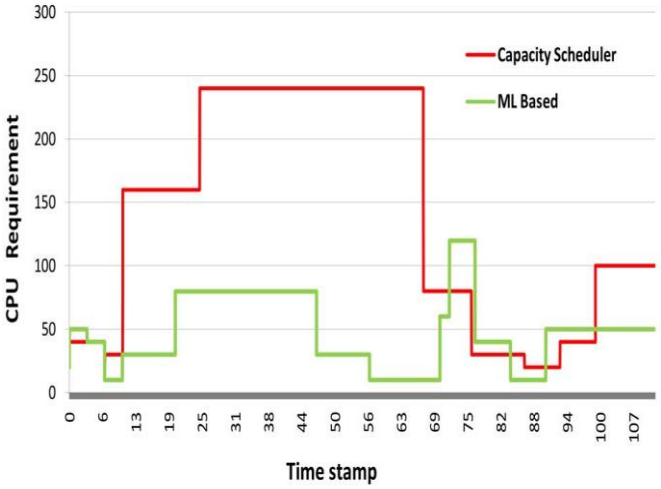

Figure 5. Comparison of cpu requirement on a TaskTracker between Capacity and Machine learning based algorithm.

V. RELATED WORK

In [4], the authors discuss the initial work representing MapReduce. Their work only discusses resource management. Hadoop's architecture [8] is inspired by their work. They also mention data execution and speculative execution i.e. re-execution of slow-tasks results in up to 40% improvement in response times. But they do not discuss ways to improve utilization on a cluster.

In [9], authors discuss a self-adaptive MapReduce scheduling algorithm which tries to classify the nodes to map-slow and reduce-slow nodes by using historical information. Whenever a task is found to be running slowly, the algorithm selects slow tasks and launches backup tasks accordingly on a faster node. In our approach, we try to understand the task compatibility on a node before a task is scheduled to avoid 'slow task' condition for that particular task on a node. And after a task is scheduled on a particular node, we try to monitor that node for overload condition and re-train the classifier accordingly to enhance decision making in future.

In [10], [11], the authors discussed about Bayesian Learning that has been used effectively in dealing with uncertainty in shared cluster environments .The authors have used a bayesian decision network (BDN) to handle the conditional dependence between different factors involved in a load balancing problems. Dynamic Bayesian Networks [12] have been used for load balancing as well. The similarity between their and our approach is the use of Bayesian inference. However whereas the authors in [11] have used a BDN, we use a Naive Bayes classifier, where all factors involved in making the decision are assumed to be conditionally independent of each other.

In [13], the authors have shown how MapReduce framework can be leveraged to run heterogeneous sets of workloads, including accelerated and non-accelerated applications, on top of heterogeneous clusters, composed of regular nodes and accelerator-enabled systems. The authors propose 'adaptive scheduler' which provides dynamic resource allocation across jobs, hardware affinity when possible, and would even be able to spread jobs' tasks across accelerated and non-accelerated nodes in order to meet performance goals in extreme conditions.

In [17], [18], [19], the authors discussed about Stochastic Learning Automata that have been used in load balancing. Learning automata learn through rewards and penalties which are awarded after successful and unsuccessful decisions respectively. However, whereas the authors have focused on conventional load balancing and process migration, we concentrate on task assignment for better job scheduling on the nodes of the clusters.

Existing Hadoop schedulers FAIR, Capacity, and Dynamic property [15, 7, and 16] offer very limited support for admission control. Existing schedulers have very little information regarding resource usage pattern of nodes. Thus, our work focuses on gathering much needed resource information on each node and thus uses that information in our learning so that tasks can be scheduled to the node on which it will be most compatible

VI. CONCLUSION

In this paper, we presented a scheduling algorithm that schedules tasks on a TaskTracker if it do not affect the normal execution of already running tasks on that TaskTracker. We discussed the benefits of having information about task and every node on the cluster. In this paper, we proposed machine learning based scheduling algorithm that selects the tasks that is best suited on a particular node. This algorithm aims to avoid overloading any node on the cluster and utilize maximum resources on a particular node thereby decreasing overall runtime of the jobs.

ACKNOWLEDGMENT

I would like to thank my guide A. Selva Kumar , Dept of Computer Science and Engineering for his able guidance, effort and encouragement toward the entire period when this work was in progress .

REFERENCES

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, JamesBroberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems,25(6):599 – 616, 2009.

[2] Nitesh Maheshwari, Radheshyam Nanduri, and Vasudeva Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. Future Generation Computer Systems, 28(1):119 – 127, 2012

[3] Jaideep Dhok, Nitesh Maheshwari, and Vasudeva Varma. Learning based opportunistic admission control algorithm for mapreduce as a service. In ISEC '10: Proceedings of the3rd India software engineering conference, pages 153–160. ACM, 2010.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.

[5] Scaling Hadoop to 4000 nodes at Yahoo!

http://developer.yahoo.net/blogs/hadoop/2008/09/scaling_hadoop_to_4000_nodes_a.html.

[6] 'Atop' utility. http://www.atoptool.nl/.

[7] Capacity Scheduler. http://.apache.org/common/docs/r0.20.2/capacity scheduler.html

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

[9] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer andInformation Technology (CIT), 2010 IEEE 10th InternationalConference on, 29 2010.

[10] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer andInformation Technology (CIT), 2010 IEEE 10th InternationalConference on, 29 2010.

[11] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer andInformation Technology (CIT), 2010 IEEE 10th InternationalConference on, 29 2010.

[12] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer andInformation Technology (CIT), 2010 IEEE 10th InternationalConference on, 29 2010.

[13] J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres, and E. Ayguade and. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In Parallel Processing (ICPP), 2010 39th International Conference on, pages 653 –662, 2010.

[14] Harry Zhang. The Optimality of Naive Bayes. In Valerie Barr and Zdravko Markov, editors, FLAIRS Conference. AAAI Press, 2004.

[15] Fair Scheduler. http://hadoop.apache.org/common/docs/r0.20.2/fair scheduler.html

[16] Dynamic Priority Scheduler for Hadoop. http://issues.apache .org/jira/browse/HADOOP-4768.

[17] AmyW. Apon, Thomas D.Wagner, and LawrenceW. Dowdy. A learning approach to processor allocation in parallel systems. In CIKM '99: Proceedings of the eighth international conference on Information and knowledge management, pages 531–537, New York, NY, USA, 1999. ACM.

[18] T. Kunz. The Learning Behaviour of a Scheduler using a Stochastic Learning Automation. Technical report, Citeseer.

[19] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. IEEE Transactions on Software Engineering, 17(7):725–730, 1991.