An Impulsive Scheduler for Dynamic Heterogeneous Hadoop Environment

[¥] Shivam Tiwari, [§] Shailesh Samudrala, [¥] Anurag Sharma, [§] Yogesh Paul
[§] Department of Electrical and Computer Engineering, University of Florida
[¥] Department of Computer and Information Science and Engineering, University of Florida

***Abstract*- The MapReduce and Hadoop frameworks were developed to efficiently support large scale computations. There has been an increasing trend in deploying Hadoop clusters for variety of applications. Various Hadoop schedulers have been designed to execute user jobs on Hadoop cluster. The main idea behind these schedulers is to improve the mean execution time. The main drawback of the available schedulers is that they assume the Hadoop cluster to be homogeneous i.e., the nodes in the cluster have the same configuration. These scheduler also uses less system information for making scheduling decision In this paper we present an impulsive scheduler, which makes its scheduling decision based on the heterogeneity of job and resources taking into account the minimum share and the fairness among the user present in the system. The objective of our algorithm is to use the system information such as estimated job arrival rate and mean job execution time to improve the mean completion time of jobs submitted by the users.** It **classifies the jobs into classes using the system information, and then finds a matching of the job classes with the resources and sends the job on its calculated resources. The scheduler also considers the priority of the jobs, required minimum share for a job, and fair share of users to make a scheduling decision**.

I INTRODUCTION

With the advent of cloud computing technology it is now become possible to efficiently do large scale computations and data analysis. MapReduce is a programming model which was designed for improving the performance of large batch jobs on cloud computing. Hadoop is an open-source implementation of MapReduce. Hadoop framework allows distributed processing on large data sets across clusters of computers. Hadoop framework made it possible to do faster computation on large scale data by scaling up from a single server to thousands of machines.

New technology trend and new business model are also the main factors which make Cloud Computing a reality, with the advent of the trends and model services which are in reach of an individual, as an example the pay-as-you-go computing provided by Amazon Web Services allows user to choose their resources (VM cycle, software stack) without disrupting other customers. Applications like mobile interactive application which relies on large data set for their result, developing a decision support system, studying customer behavior, their buying habits. Batch processing systems that analyses terabytes of data are some of the new challenges that trigger the rise of Cloud Computing.

There is been a growing interest of deploying Hadoop cluster on various applications, the idea is to share a large single Hadoop cluster between multiple user. Each user submits jobs on the cluster; the jobs can range from long batch jobs like analyzing a large data-set, processing and performing computation on large scientific data to short jobs like queering on small data-sets. In Hadoop a single cluster is shared between multiple users. This sharing of cluster has several advantages such as increasing data locality (running the job where data is present), increasing utilization of resources and lowering the cost of execution of jobs in the cluster.

The Hadoop scheduler is the heart of Hadoop system. The task of the scheduler is to assign resources to jobs, maintain information about the progress of the jobs, and in case of a node failure send the task to other node for completion. An efficient scheduling algorithm should address issues like maximizing the cluster utilization, provide fairness among the users, quickly assign resources to jobs, and avoiding starvation of a task, handle node failure.

Initial Hadoop scheduling algorithms like the FIFO algorithm and the Fair-sharing algorithm uses small amount of system information to quickly multiplex the incoming jobs. The arrival of new type of job to a Hadoop cluster may degrade the performance of the system. Moreover the current scheduler of Hadoop neglects the heterogeneity of the system. Here heterogeneity can be in resources or in jobs inside the Hadoop cluster. A scheduling decision that ignores heterogeneity of the system and uses small amount of system information may lead to poor performance and can cause issues like less data locality, loss of fairness inside the Hadoop cluster. Therefore, applying Hadoop scheduling algorithms in heterogeneous environment may not be a good choice.

In a heterogeneous Hadoop environment it is useful to measure the possible performance gains by exploring the difference between various resources and jobs in the system. Gathering more information about the jobs such as their arrival rate, their mean execution time in the system can have significant impact on making better scheduling decision.

Research at UC-Berkeley [2] provides a means to gather Hadoop system information that can be used in making scheduling decision. It provide a means to estimate the mean job execution time of the in-coming jobs based on its structure and the number of map and reduce task for each job. We take advantage of the fact that in most of the Hadoop systems, multiple jobs are repeated in various patterns. As an example, Facebook runs the Spam detector applications on the Hadoop cluster every night. Therefore it is possible to get the arrival rates of various jobs in a Hadoop environment.

In this paper we present a Hadoop cluster scheduling algorithm which makes use of the system information to make its scheduling decision. It makes use of the fact that most of the jobs that run on Hadoop are repeated and follow certain trend, hence their arrival rate and mean execution time can be predetermined. The algorithm also takes into account the heterogeneity of both resources and jobs in assigning nodes/resources to jobs It classifies the jobs into classes using the system information, and then finds a matching of the job classes with the resources and sends the job on its calculated resources. The scheduler also considers the priority of the jobs, required minimum share for a job, and fair share of users to make a scheduling decision. Our algorithm is dynamic in the sense that it updates its scheduling decision whenever a change in system parameters occurs.

We evaluate the performance of our scheduler and the three current schedulers of Hadoop namely FIFO, Capacity and Fair Schedulers by using 4 different performance metrics: locality, fairness, satisfying minimum share of the users and mean completion time of jobs.

The results show that our algorithm simultaneously considers heterogeneity in available resources and jobs present in the cluster to make its scheduling decision. The 2 main advantages of our proposed algorithm are the resource utilization and reducing the mean completion time of jobs considering the heterogeneity inside the Hadoop cluster.

The remainder of this paper is organized as follows. In section 2 we provide a brief overview of Hadoop. Our Hadoop scheduling model and its subsystem layout is given in section 3. Then in section 4 we formally state the performance metrics that we used to evaluate the performance of our scheduler and the current scheduler of Hadoop (FIFO, Fair and Capacity). Section 5 gives the details of the environment on which we have measured the performance of our system and provide a comparative study between the available schedulers in Hadoop and that of our scheduler. Section 6 describes related work to our proposed system. Finally we conclude the paper and discuss possible future direction in section 7.

II HADOOP SYSTEM

Cloud computing provides big clusters that efficiently perform large scale data computation and analysis. To achieve this it is required to break down the application into smaller pieces and execute them in parallel on several machines at the same time.

MapReduce is a programming model that can efficiently parallelize the computation of large scale jobs. In MapReduce a user first specify a map function and a reduce function in order to process a large set of data. The programs which are written using this approach are automatically parallelized and executed over large clusters of machines. User doesn't have to bother about partitioning the input data, scheduling the program execution on a set of machines, machine failure etc. Hadoop is an open source implementation of MapReduce for scalable distributed computing. The Hadoop Distributed File System is a scalable and portable file system for the Hadoop framework which is designed to handle large files. The Hadoop scheduler breaks the user job into multiple map and reduce tasks, after that the Hadoop first runs the map tasks on a free resource and computes key/value pair from each part of the input. Then it generates intermediate Key/Value pair by grouping values by their key. Finally it run reduce task on the intermediate key/value pair to generate final result.

The scheduler is the main component of Hadoop system. The task of the scheduler is to assign resources to jobs, maintain information about the progress of the jobs, and in case of a node failure send the task to other node for completion. Scheduling in Hadoop is pool based i.e. when a resource is free it notifies the scheduler, upon receiving the notification from a free resource the scheduler searches through all the jobs that are waiting for a resource, applies some scheduling decision to select a job from the job queue and assigns that resource to one or more tasks of the selected job. The notification send by a free resource contains information such as number of current free slots. A Hadoop scheduler has to check certain parameters such as job priority, job arrival rate, number of free nodes in the cluster, data locality, minimum share of a user and nature of job (short or long) for scheduling jobs on the nodes. Various Hadoop Scheduling algorithms considers different parameters in making scheduling decision

III PROPOSED SCHEDULER MODEL

In this section we first present a high level overview

of our proposed algorithm and then discuss the subsystem design of the scheduler in more detail.

A. Architecture

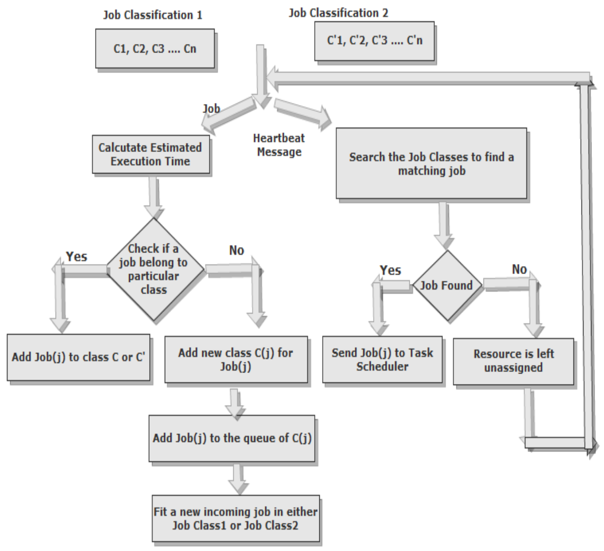
Job Classification 1
C1, C2, C3 .... Cn
Job Classification 2
C'1, C'2, C'3 .... C'n
Job
Heartbeat Message
Calcutate Estimated Execution Time
Search the Job Classes to find a matching job
Check if a job belong to particular class
Yes
No
Job Found
Yes
No
Add Job(j) to class C or C'
Add new class C(j) for Job(j)
Send Job(j) to Task Scheduler
Resource is left unassigned
Add Job(j) to the queue of C(j)
Fit a new incoming job in either Job Class1 or Job Class2

Figure 1: High level overview of our algorithm

Figure 1 describes the architecture of Impulsive scheduler. A Hadoop scheduler receives 2 messages: a message on the arrival of a new job in a cluster and a message from a resource whenever it becomes free. This message send by the resource is also known as a heartbeat message. Our scheduler performs two main operations, where each operation is triggered by the arrival of one of these messages. Upon the arrival of a new job in the Hadoop cluster the scheduler classifies the job and places it into the appropriate queue dedicated for that class. When a free resource sends a heartbeat message to a scheduler, the scheduler finds a matching job and assigns it to the free resource. By matching we mean to say that the scheduler find a job among a set of yet to be assigned jobs that is most appropriate for execution on a given free resource.

When a user submits a new job the scheduler performs its queuing algorithm to store the job in a desired queue. Our algorithm performs job classification to classify the jobs present in the cluster and route them to a corresponding queue. After that updated information of all the job classes is send to the scheduler which uses this information to choose a job for a free resource.

In the following sections we describe our algorithm in much more detail

B. *Use-Case Description*

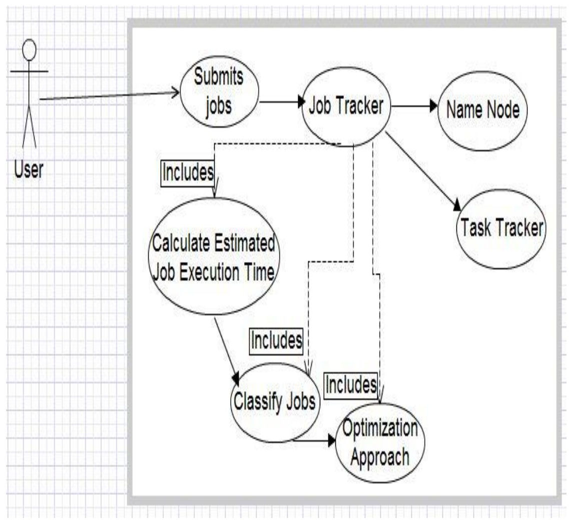

Figure 2: A Use-Case diagram depicting the working of our scheduler.

Figure 2 shows a use case illustration of what happens when a user submits a job in the Hadoop cluster. When a user submits job to the JobTracker, the JobTracker first calculate the estimated job execution time of the submitted job, after that it classifies the job in one of the 2 classifications: the minimum share classification and the fairness classification. One the job classification is done we perform job-resource matching to find appropriate resources for the submitted job.

The JobTracker then talks to the NameNode to determine the location of the data and submits the job to the chosen TaskTracker. The TaskTracker nodes are monitored and if they do not submit notification signal often enough, they are declared failed and the work is scheduled on a different resource/ nodes. The TaskTracker will notify the JobTracker when a task fails, the JobTracker then decides whether to resubmit the job elsewhere or mark the record as something to avoid.

C. *Job Execution Time Estimation*

It is required to estimate the job execution time on the resources on the arrival of a new job in the cluster. We take advantage of the fact that most of the jobs in Hadoop are repeated and once we know the time it take to run that job on a resource, we can use this information in the future to calculate its mean execution time in the cluster.

D. *Job Classification*

In a Hadoop system when a user request for resources to execute his job, it is required by the system to give the user his minimum share immediately. After satisfying the minimum share of all the users if the system still left with additional resources it divides those resources to all the users in a fair way.

Based on the above facts we classify the job in two main classifications: minimum share classification and fairness classification. When a user request comes in the cluster we first assign the minimum share to the user. If a user asks for more than his minimum share we assign the extra share fairly after we assign minimum share to all the users currently present in the cluster. As a result of this we first consider jobs in the minimum share classification first and after all the user receives their minimum share we consider jobs in fairness classification.

In the above mentioned classification we further divide jobs into sub-classes according to their arrival rate and their priorities. We define the minimum share classification as JobClass1 and fairness classification as JobClass2. A sub-class C_i has a specific arrival rate. Jobs in this sub-class C_i will have similar arrival rates and priorities. The minimum share classification of jobs is shown in Figure 3.

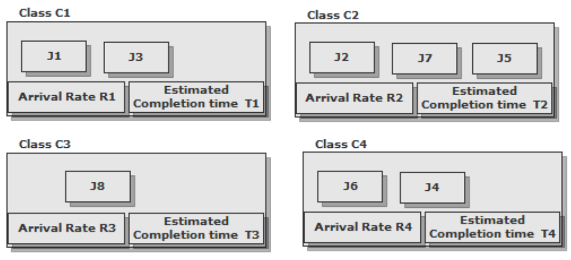
Class C1
J1
J3
Arrival Rate R1
Estimated Completion time T1
Class C2
J2
J7
J5
Arrival Rate R2
Estimated Completion time T2
Class C3
J8
Arrival Rate R3
Estimated Completion time T3
Class C4
J6
J4
Arrival Rate R4
Estimated Completion time T4

Figure 3: Minimum share classification of jobs

The fairness classification is also similar to the minimum share classification. We consider jobs in this classification only after all the users in the cluster have acquired their minimum share.

E. *Job Resource Matching*

Job resource matching is a well known problem that has been studied in the context of multiprocessors. It becomes all the more difficult when the offered resources in the environment are heterogeneous. In the past work has been done in direction of application placement on the resources. We adopt this work for MapReduce which is based on a Utility function.

The Job-Resource matching algorithm finds an optimal mix of jobs for a Task Tracker. To achieve this scheduler work on each worker node, tries different job combinations that will give the best performance executing a particular job on that node, and after certain number of iteration it finds a best matching of the number of jobs suitable for each worker node. This algorithm executes periodically or when a new job arrives to the system. After the Job-Resource matching is done it is now required to send a yet to be scheduled job to its appropriate resource. When a free resource sends a heartbeat signal to the scheduler, it selects a matching job for that resource and sends the selected job to each free slot in the resource.

IV PERFORMANCE MODEL

We will be using mean completion time for evaluating the performance of our scheduler and the current scheduler in Hadoop. Mean completion time is calculated by summing the time it need to complete all the jobs in the cluster divided by the number of jobs present in the cluster. A scheduler which takes less time to complete the jobs performs better.

$$MeanCompletionTime = \frac{\sum_{i=1}^{N} Completion\ Time(ti)}{N}$$

V RESULTS

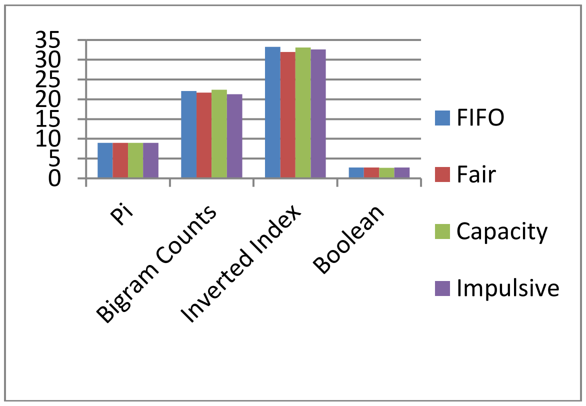

Figure 4: Graph for results of small jobs – run on 5 nodes

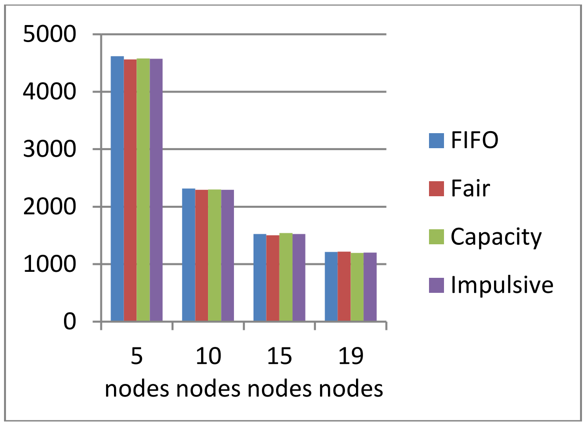

Figure 5: Word count execution time in seconds – input dataset is of size 12.6 GB

Job→ Scheduler	Pi-5	Boolean	Inverted	Bigram Count
FIFO	9.007	2.953	33.269	22.09
Fair	9.012	2.699	31.980	21.708
Capacity	8.98	2.717	33.119	22.381
Impulsive	8.982	2.729	32.631	21.312

Table 1: Results obtained by running jobs on 4 schedulers

.VI RELATED WORK

MapReduce was initially designed for small jobs with a simple scheduling algorithm like FIFO to achieve acceptable performance level. As there been a growing interest of deploying Hadoop cluster on various applications, the scheduling of jobs in a Hadoop cluster has become worth inspecting. Current scheduler of Hadoop includes FIFO, Fair and capacity. FIFO is the default scheduler of Hadoop, it operates using a FIFO queue i.e., and the jobs are processed in the order of their submission. Each job occupies the whole cluster, so jobs have to wait for their turn. In FIFO scheduler we can assign priorities to jobs, but priorities do not support preemption. The problem with FIFO scheduler is that in large system it can cause severe performance degradation.

Hadoop Fair scheduler was developed by Facebook. In fair scheduler jobs are grouped into Pools. Each pool is assigned a guaranteed minimum share and by default jobs that are uncategorized go into a default pool. Pools have to specify the minimum number of map slots, reduce slots, and a limit on the number of running jobs. It also implements a "shortest job first" policy which reduces response times for interactive jobs.

Another scheduler of Hadoop called Capacity scheduler was developed at Yahoo provides support for multiple queues. The queues are allocated a fraction of the capacity of the grid. The jobs submitted to a particular queue will have access to the resources allocated to that queue only. After allocating resources to the queue the free resources can be allocated to any queue beyond its capacity. The queues support job priorities but these are disabled by default. In order to maintain fairness among users, each queue enforces a limit on the percentage of resources allocated to a user at any given time.

VII CONCLUSION and FUTURE WORK

The current schedulers of Hadoop do not consider heterogeneity of Hadoop cluster resources and jobs in making scheduling decision. The algorithms are designed to keep the things simple by gathering less system information which in some case may lead to performance degradation. There has been a growing interest in applying MapReduce paradigm to a variety of applications giving rise to a need of a better scheduling algorithm that can make its scheduling decision based on the heterogeneity of the system. It is possible to estimate system parameters in a Hadoop system. Our proposed scheduler perform scheduling based on heterogeneity of jobs and resources taking into account the minimum share and fairness[5] among the user present in the system. It uses system information such as estimated job arrival rates and mean job execution times to make scheduling decisions.

Future work may include improving the data locality part on this scheduler so as to execute a job on nodes where its data is present. We have to work on the fairness part of our scheduler to improve the efficiency of our algorithm. Fairness measures how fair a scheduling algorithm is in dividing resources to the user in the system. A fair algorithm gives same share of resources to the user when their jobs have same priority, however when user priorities are not equal the scheduler must divide the resources in proportion to the user priorities.

More work has to be done in the direction of improving the **Minimum share dissatisfaction** of the scheduling algorithm. It measures how successful a scheduling algorithm is in satisfying the minimum share requirements of the user in the cluster. A scheduler which has less Minimum share dissatisfaction performs better. Better method can be used for jobs classification to further improve the performance of the scheduler.

REFERENCES

[1] Aysan Rasooli, Douglas G. Down. An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems. *CASCON 2011.*

[2] M. Zaharia et al. Improving MapReduce performance in heterogeneous environments. In OSDI, pages 29–42, 2008.

[3] M. Zaharia et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In EuroSys, 2010

[4] Aboulnaga, Z. Wang, and Z. Y. Zhang. Packing the most onto your cloud. In Proceedings of the first international workshop on Cloud data management, 2009.

[5] S. Agarwal and G. Ananthanarayanan. Think global, act local: analyzing the trade-of between queue delays and locality in mapreduce jobs. Technical report, EECS Department, University of California, Berkeley.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51:107{113, January 2008.}

[7] M. Zaharia, D. Borthakur, J. S. Sarma, K.Elmeleegy, S. Shenker, and I. Stoica. Jobscheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009- 55, EECS Department, University of California, Berkeley, April 2009.

[8] T. Sandholm and K. Lai. Dynamic proportional share scheduling in Hadoop. In
Proceedings of the 15th workshop on job scheduling strategies for parallel processing,
Pages 110 {131. Springer, Heidelberg, 2010.}

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In Proc. OSDI, pages 137–150, December 2004.

[10] D. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine grained synchronization. Journal of Parallel and Distributed Computing, 16(4):306–18, 1992

[11] V. K. Naik, S. K. Setia, and M. S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In Proceedings of Supercomputing, pages 824–833, November 1993

[12] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job Scheduling for Multi-User MapReduce Clusters. Technical Report UCB/EECS-2009-55, University of California at Berkeley, April 2009

[13] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

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

[15] HADOOP-3759: Provide ability to run memory intensive jobs without affecting other running tasks on the nodes. https://issues.apache.org/jira/browse/HAD-OOP -3759.

[16] Hadoop on Demand Documentation. http://hadoop.apache.org/core/docs/r0.17.2/hod.html

[17] A. C. Dusseau, R. H. Arpaci, and D. E. Culler. Effective distributed scheduling of parallel workloads. SIGMETRICS Performance Evaluation Review, 24(1):25–36, 1996

[18] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gang matching. In Proc. High Performance Distributed Computing, pages 80–89, 200

[19] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST'09), pages 85–98, February 2009.

[20] Huan Liu. Cloud MapReduce: a MapReduce implementation on top of a Cloud Operating System, CCGrid 2011

[21] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In SOSP 2009, 2009.

[22] A. Aboulnaga, Z. Wang, and Z. Y. Zhang. Packing the most onto your cloud. In Pro-ceedings of the first international workshop on Cloud data management, 2009.

[23] Run various jobs on Hadoop http://lintool.github.com/Cloud9/index.html