A Hybrid Scheduling Algorithm for Data Intensive Workloads in a MapReduce Environment

Phuong Nguyen[1] Tyler Simon[1] Milton Halem[1] David Chapman[1] Quang Le[2]
[1]Dept. Computer Science and Electrical Engineering
University of Maryland Baltimore County
[2]General Dynamics Information Technology
Email {phuong3, tsimo1, halem, dchapm2}@umbc.edu and Quang.Le@gdit.com

Abstract— **The specific choice of workload task schedulers for Hadoop MapReduce applications can have a dramatic effect on job workload latency. The Hadoop Fair Scheduler (FairS) assigns resources to jobs such that all jobs get, on average, an equal share of resources over time. Thus, it addresses the problem with a FIFO scheduler when short jobs have to wait for long running jobs to complete. We show that even for the FairS, jobs are still forced to wait significantly when the MapReduce system assigns equal sharing of resources due to dependencies between Map, Shuffle, Sort, Reduce phases. We propose a Hybrid Scheduler (HybS) algorithm based on dynamic priority in order to reduce the latency for variable length concurrent jobs, while maintaining data locality. The dynamic priorities can accommodate multiple task lengths, job sizes, and job waiting times by applying a greedy fractional knapsack algorithm for job task processor assignment. The estimated runtime of Map and Reduce tasks are provided to the HybS dynamic priorities from the historical Hadoop log files. In addition to dynamic priority, we implement a reordering of task processor assignment to account for data availability to automatically maintain the benefits of data locality in this environment. We evaluate our approach by running concurrent workloads consisting of the Word-count and Terasort benchmarks, and a satellite scientific data processing workload and developing a simulator. Our evaluation shows the HybS system improves the average response time for the workloads approximately 2.1x faster over the Hadoop FairS with a standard deviation of 1.4x.**

Keywords— Hadoop, Scheduler, dynamic priority, scheduling, MapReduce, workflow

I. INTRODUCTION

The most popular Big Data tool for cloud computing offered today by most commercial providers such as Microsoft, Oracle, IBM, Amazon et al., is based on the Apache open source Hadoop Map Reduce environment [1, 2]. The original proposed MapReduce model is by Dean and Ghemawat [3]. One reason for its broad acceptance by many business and government organizations is that a rich set of Hadoop software supporting library systems have grown up providing a wide variety of complementary organizational requirements such as Hbase, Hive, Pig, Mahout and recently a workflow engine Oozie. A key benefit of the Hadoop core MapReduce system is that it automatically handles failures, hiding the complexity of fault-tolerance. Additionally, because data locality is critical [4, 5], the default Hadoop FIFO scheduler gives preferences to node local and rack local tasks to improve data locality.

Limitations of Hadoop FIFO occur when short jobs have to wait too long behind long running jobs, thus negatively affecting the job response time. The Hadoop FairS, developed by Zaharia et al. [4], was the first to addresses this limitation in depth by utilizing a fair share mechanism between multiple concurrent jobs. Over time, FairS assigns resources such that all jobs get, on average, an equal share of resources. This methodology significantly improved the average response time of Facebook queries [4]. Additionally FairS extends the data locality of FIFO by using a delayed execution mechanism [4].

Workload specific choice of MapReduce task schedulers affects the performance of MapReduce applications [6] because currently Hadoop FairS and FIFO schedulers depend on the frequencies of job submission patterns and the system workloads. For example, Y. Chen showed that if there is a long sequence of small jobs submitted after a few large jobs, then FIFO shows superior response time over FairS [6]. Furthermore, we show that while sharing resources in a MapReduce system, FairS response time is still longer than necessary due to dependencies between the Map/Shuffle/Sort/Reduce phases. Thus, we propose a new Hadoop Scheduler called HybS based on dynamic priority in order to reduce the delay for variable length concurrent jobs, and relax the order of jobs to maintain data locality. In addition, HybS provides a user-defined service level value for QoS. We show how dynamic priorities can accommodate multiple tasks' lengths, job sizes, and jobs' waiting time in this environment while reducing average response time even further beyond FairS.

The contributions of this work are as follows:

- A new MapReduce task scheduling algorithm, called HybS based on dynamic priority, is proposed to reduce workload response time of concurrent running MapReduce jobs.
- Hadoop Scheduling decisions using estimated Map running times and user-defined service level value
- Automatic relaxing of the order of job execution to preserve data locality.

The paper is organized as follows. Section I is the paper's introduction. The new task scheduling algorithm is presented in section II. Experimental analysis results of the algorithm are

978-0-7695-4862-3/12 $26.00 © 2012 IEEE
DOI 10.1109/UCC.2012.32

presented in section III. Related work is discussed in section IV. The final section V presents conclusions and future work.

II. SCHEDULING ALGORITHMS

A. The MapReduce Scheduling Model

The Hadoop scheduling model is a Master/Worker cluster structure, for which the master node (Job Tracker) is in charge of scheduling decisions, and the workers (node trackers) are responsible for executing tasks. Worker nodes will acknowledge the Job Tracker when they are available to run tasks and when the job is complete. They then pull tasks from the Job Tracker's running queue. The Job Tracker maintains a global single queue, and all jobs must submit to this queue which is ordered by the master policy as in Figure 1.

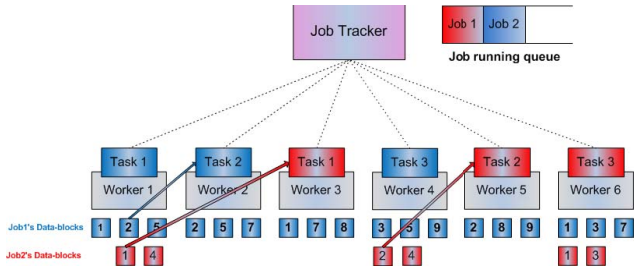

Figure 1 Worker nodes distribute the tasks within multiple jobs as resource nodes become available

A MapReduce application job consists of multiple parallel Map and Reduce tasks. Map tasks are independent from other Map tasks and do not communicate amongst each other during execution. Reduce tasks are also independent from other Reduce tasks. However, Reduce tasks depend on the output of Map tasks. In other words, Reduce tasks have to wait until all Map tasks of a given job are finished before they can be executed. Map tasks from different jobs can share the same input data files. The files are split into chunks of the same size and stored in a distributed file system and available before job submission. The scheduler assigns one Map task per chunk and gives preference to the node that already holds the respective data chunk (node-local tasks). If such local assignments cannot be made, then the task is assigned to a remote node where the respective chunk is moved. Since Hadoop can be configured to identify network topology, it can give secondary preference to rack-local tasks as opposed to non-local tasks run on a completely separate rack from their data chunks. A daemon process transfers output from all Map tasks (called intermediate files) to the local storage of nodes that run Reduce tasks. These Reducer nodes read these files from their local storage, apply the appropriate Reduce function and write the final output to files in the distributed file system.

We also assume in this scheduling model, that the MapReduce cluster shares execution between up to K jobs. In other words, the entire cluster is never dedicated to a single job. Sharing resources does not need to be equal however, as the response time of all jobs is improved by dynamically adjusting the job priority based on its runtime, waiting time and job size. The job's waiting time and the size are dynamic. As the job waits in the queue longer the waiting time is increased. Similarly, as the job's individual tasks are completed, the job's remaining size is decreased.

The objective of our scheduling algorithm is to improve the expected response time of any given job while still meeting the user specified service level agreements. Tasks are prevented from consuming all available resources and thus starving other tasks. HybS relaxes these policies to preserve data-locality of non-local data by assigning tasks from jobs out of order of their priority. This strategy is similar to delay technique employed by the FairS. However the HybS does not set a constant delay time *D*. Jobs with higher priority and jobs skipped by out of order assignments will be automatically pushed up in the priority queue for the next assignments.

B. Dynamic priority

HybS uses both dynamic priority and proportional share assignment to determine which tasks from which jobs should be assigned to the available resource node. Thus the priority of a job would affect the choice of the task assignment. The scheduling policies for the job priority can be assigned by an administrator by tuning the system parameters of the job's workload. One possible policy may be, as jobs wait longer their priorities are increased. Thus, the priority includes a factor such that the priority weight dependence on waiting time is

$$weight_{wait-time} = (\frac{t_d}{avgWaitTime})^{\alpha}$$

Where t_d is the time that a job waits in the queue, and *avgWaitTime* is the average job waiting time in the system, and α is the weighting factor. Job running time also can be a factor for a dynamic priority by favoring shorter running jobs over longer running jobs. With MapReduce jobs, we consider the Map task runtime as the job runtime. This factor will correspond to a second weight depending on job runtime as

$$weight_{run-time} = \left(\frac{r}{avgRunTime}\right)^{\beta}$$

Where *r* is the average runtime for the Map tasks of a MapReduce job, and *avgRunTime* is the average of the average runtime for the Map tasks for all jobs, and β is the weight factor. The average runtime for Map tasks of each job is getting from the historical Hadoop log files. We also use an online profiler for average of map running time if there is no information available in the job history logs. Further, we consider the number of remaining unscheduled tasks for a job as another weight as

$$weight_{Job-size} = (\frac{n}{avgNumTasks})^{\gamma}$$

Where n is the number of remaining tasks for the job and *avgNumTasks* is the average number of remaining tasks per job over the entire queue, and γ is the weighting factor. The combination of these weights becomes the final priority for the job in the system as

$$priority = (\frac{t_d}{avgWaitTime})^{\alpha} * \left(\frac{r}{avgRunTime}\right)^{\beta} * (\frac{n}{avgNumTasks})^{\gamma} \quad \text{(eq.1)}$$

The importance of each user in the queue may be accounted for as

$$weight_{user-queue} = u^{\delta}$$

where *u* is the user priority and δ is the weighting factor.

We set δ to 0 for the experiments in this paper, but this factor could be used to provide variable quality of service in a

multi-user or cloud environment by multiplying this factor to the other factors in equation eq.1.

The priority equation and weight factors are adapted from W. A. Ward at el. (2002), where the job priorities are used for scheduling parallel jobs in HPC systems using backfilling techniques [7]. The α, β, and γ weighting factors are called priority parameters. There are several possible policies for these priority parameters. If α=1 and β=γ=0 the scheduling policies become FCFS (First Come First Serve). Whereas α=0, β=-1 and γ=0 produces shortest job first. Furthermore, α=0, β=0 and γ=-1 yields smallest job size first while γ=1 produces biggest job size first. Determining the best non zero α, β, and γ sets for a given system and workload is an interesting problem because they give applications a choice of scheduling policies.

The dynamic scheduling algorithm considers the estimated runtime, job size, and waiting time of the job in the queue. The waiting time is increased as the job waits in the queue, whereas the remaining size of the job is reduced when its individual tasks are completed.

In systems with a high number of concurrent jobs, in order to reduce the overhead of searching through the queue for data locality, we can define a window *W* to limit the number of jobs considered for data locality. Window *W* is tunable parameter, and allows the balance of conflict tradeoff between scheduling efficiency and data locality. The complexity of the scheduling algorithm is the complexity of sorting the queue of n jobs and it is O(nlogn).

C. Scheduling Algorithm

Dynamic job priorities are used in HybS for reducing the latency of variable length concurrent jobs, while maintaining data locality. The dynamic priorities can accommodate multiple task lengths, job sizes, and job waiting times by applying a greedy fractional knapsack algorithm for job task processor assignment.

Fig. 2 presents the hybrid dynamic priority scheduling algorithm. If Hadoop has *C* available slots, then we would want to have *C* current tasks in execution at any particular time. At each heartbeat, each job job_i has F_i remaining tasks, priority of p_i (calculated from equation 1), and a user- defined service level value of s_i. The idea of service level value for a job is to provide variable quality of service as a function of system resources. One can set the same value for all of the jobs in his/her queue if preferred. The user-defined service level value must be between 0 and 1.

Of these F_i tasks, only the job's predicted number of tasks $A_i = F_i s_i$ would be eligible to be scheduled at any given time. We will schedule all node and rack local tasks from all jobs before we even consider scheduling a non-local task by the greedy fractional knapsack algorithm as in step 4 of Fig.2. All predicted tasks of job's highest priority that have data locality (node-local and rack-local) will be assigned first. If the job of highest priority does not have enough data locality, tasks of next highest priority job will be assigned. The relaxation of the highest priority job's assignment utilizes node-local data without waiting or delaying the execution of any task by a fixed amount of time.

For each heartbeat, when there are free tasks available on a node:

1. Update dynamic priority for all jobs based on *eq 1*
2. Calculate the predicted number of tasks A_i to be assigned for each job as $A_i = F_i s_i$ and sort the job queue with descending order of priorities
3. Assign node_local and rack_local tasks by a greedy fractional knapsack algorithm as

   loop_job: for *i* in *jobs* **do**
   for j in *Min(A_i,C)***assign node/rack_local task;**
   C = C-1;
   If (C=0) exit loop_job
4. If C>0, assign a non-local task to the first job in the queue.

Figure 2. HybS algorithm pseudo-code

III. EXPERIMENT RESULTS

A. Results of hybrid scheduler using Hadoop

The dynamic priority scheduling algorithm is implemented in Hadoop 1.0.1 and labeled as HybS in the Hadoop system for performance analysis. We ran Hadoop using the HybS, FairS, and FIFO schedulers for comparison.

We also performed the evaluations on two different configurations, a physical system and a virtual system.

Physical cluster: Hadoop 1.0.1 is installed on a cluster of 9 physical IBM blades, each with two Intel Nehalem 2.0 GHz quad core processors on the same rack. The total number of cores is 72. Each processor has 24 GB RAM and the local disk capacity of the Nehalem blades is 1TB each. Each blade is configured to run 4 concurrent map tasks, yielding a total map capacity of 36 slots.

Virtual cluster: Hadoop 1.0.1 is installed with the open source Eucalyptus cloud. A virtual cluster is configured with 16 virtual nodes, each node image is configured with 1 core and 64 GB of local disk and 4GB RAM. Total map capacity is configured for 32 slots (i.e. each virtual node runs 2 concurrent maps).

The internal physical network for both clusters above is a 1 Gigabit Ethernet connection. The Hadoop default configuration assumes each user is configured as a queue. Thus, the term user here has the same meaning as queue. We use the Hadoop examples, TeraSort and WordCount, for one class of benchmarking performance. We also use gridding of the NASA AIRS instrument data as a real-world benchmark of the HybS performance.

Hadoop's speculation for *Map* and *Reduce* tasks are turned off because we just want to compare schedules of MapReduce task executions by each scheduler.

TeraSort: Although called TeraSort, for our experiments the benchmark sorts data on the order of GBs per dataset, these workloads represent both I/O and CPU intensive jobs.

WordCount: A program that counts the number of times each word appears in a document. This application represents the regular distribution of task length and random distribution

of key value pairs. We run this program to count the appearances of words on Wikipedia dataset (66GB).

GridAIRS: A hyperspectral weather satellite re-projection problem represents both an I/O and CPU intensive workload. Each file is a 6 minute earth observing infrared measurement projected into a latitude/longitude grid utilizing a noise reduction algorithm [9]. One day of AIRS data is 14GB and consists of 240 granules or files. We configure each file as a HDFS block. This represents a scientific data intensive workload in which the distribution of key/value pairs has spatial data access patterns. This workload is both I/O and CPU intensive.

Table 1: Job input information for experiment 1 (runtime in seconds)

Job type	avg Map task runtime	avg Reduce task runtime	avg shuffle Task runtime	# of map in range	total input data in GB
Grid	113	49	8	60-240	<14
Terasort	7	21	21	90-900	<6
WordCount	400	24	62	09-32	<66

The experiments reported in this paper are run on the physical cluster unless we specifically indicate that experiments are run in the virtual mode. We run Hadoop only once on the cloud system because the cloud is a dedicated system, while the physical cluster is not. When the physical cluster is used, we report the average of 5 runs. We also report the overhead of running Hadoop in a cloud environment. The individual jobs are submitted to Hadoop by means of a MapReduce DAG workflow system [10].

Experiment 1. This Hadoop cluster has a capacity of 36 concurrent map slots run on the physical cluster. We will compare FIFO and the FairS against the HybS for two selected α, β, and γ parameters. There are 21 jobs run concurrently on the cluster in experiment 1. The jobs of various sizes and estimated runtimes are submitted in random order. In figure 3, there are some small jobs follow by some big jobs and then the rest are small jobs in submission order. The purpose of this workload is to see the reduction in waiting time by the scheduler for various job types. The workload can also represent the production workloads in HPC or in industry such as Facebook, and Yahoo where the job arrival time can be modeled as poison distribution. The number of required maps (from different input dataset size) is indicated in table 1.

In Fig 3. HybS **α=1 β=-1 γ=-1 (red)** has a lower or equal job response times to the FairS for almost all jobs except for the largest jobs (job 13). Since HybS **α=1 β=-1 γ= -1** use shortest job first and smallest job size first policies, both HybS and FairS give short jobs (job 4 to job 9) better chances to finish earlier while FIFO does not. There are similar patterns shown for jobs (18, 19, and 20). HybS **α=1 β=-1 γ=-1** has overall lower response times comparing HybS **α=0 β=-1 γ=-1** because it considers the job's waiting time factor. We choose these policies because short jobs should not have to wait for long running jobs to share the resources and also prevent jobs from starving where long running jobs wait too long for the resource allocation.

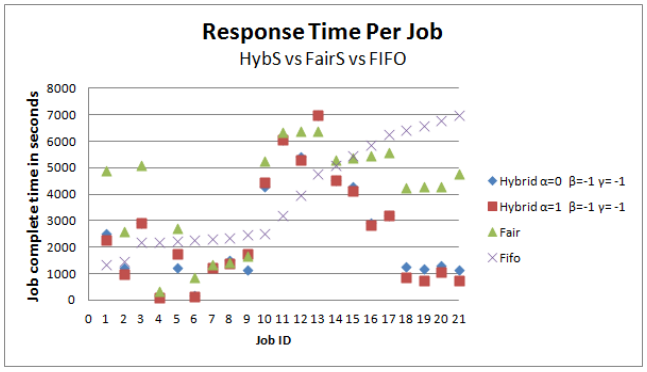

Figure 3 Total job's runtime HybS vs FairS vs FIFO scheduler. 21 jobs with different workloads described in table 5. HybS with α=0 β=-1 γ= -1 and α=1 β=-1 γ= -1(experiment 1).

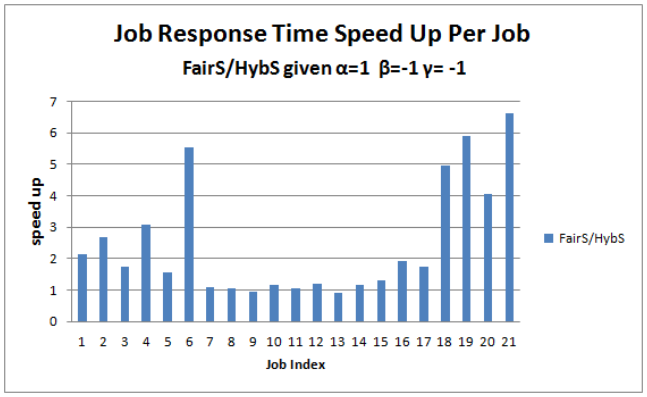

Figure 4 Detailed comparison of HybS α=1 β=-1 γ= -1 vs FairS for all jobs in experiment 1.

Fig. 4 plots the response time speed up of HybS over FairS for the parameters **α=1 β=-1 γ=-1**. HybS performed better for all jobs except job 9 and job 13. For small jobs (18, 19, 20, 21) HybS performs 4x to 6.7x faster, while FairS shows 1.04x to 1.09x faster for big jobs (9, 13). HybS provides 2.4x faster response time on average than FairS for the average workload of 21 jobs. For the job submission order and their workload in this experiment, the results show that the waiting time for small jobs is significantly reduced compared with FairS.

Experiment 2. A small workload of 95 small jobs running the Terasort benchmark with a variable number of Map tasks and input dataset sizes. The number of Map tasks ranges from 50 to 148 with different input dataset sizes on the order of (MB-GB). The average Map task runtime fluctuated between 4 to 6 seconds. This workload simulates the long sequence of small jobs in a snapshot of a day-long Facebook workload presented in fig. 11 (Y. Chen et al.) [6].

Figure 5 shows HybS is superior compared to FairS for the workload in experiment 2. HybS provides 2.1x faster response time for jobs on average over FairS for this workload.

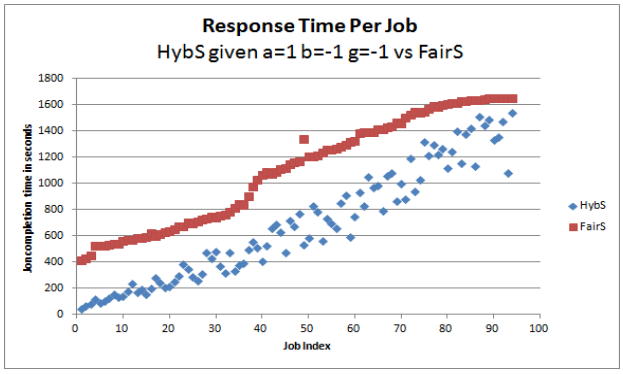

Figure 5 Total job's runtime HybS α=1 β=-1 γ= -1 vs FairS. The 95 Terasort jobs with different workloads are described in experiment 2.

HybS more quickly resolves the data dependence between the *Map* and *Reduce* stages. Because the final Reduce must wait for the output from all maps, the last Map task is the main latency bottleneck toward job completion. This improvement is likely due to the γ parameter of -1, which boosts the priority of jobs with fewer remaining tasks. In other words, HybS attempts to "finish off" nearly completed jobs rather than leave dangling a few *Map* tasks. This allows the *Reduce* to start more quickly with resolved data dependencies providing faster response time on average. Since the experiments are performed using Hadoop on the same rack and the number of tasks are higher than the cluster Map capacity, the rate of node local data is very high. We were not able to evaluate completely non-local tasks because we had one rack available. We plan to run experiment at large scale system with different racks to evaluate the data locality technique.

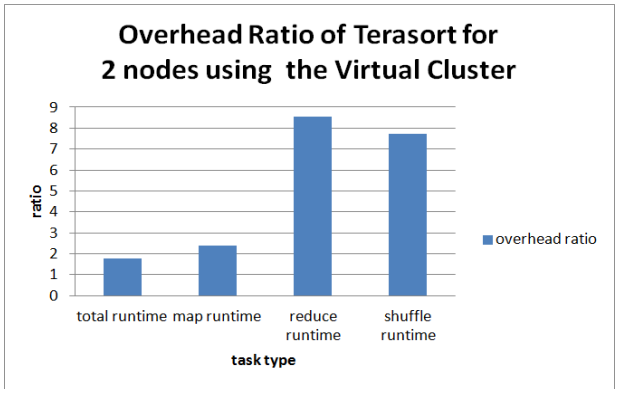

Figure 6 Average overhead ratio (Virtualized / Physical runtime) for two jobs of the Terasort benchmark using the Eucalyptus cloud (Virtual cluster)

Experiment 3 (on Virtual cluster). We ran the Terasort benchmark on 1.2GB of data using 2 jobs each with 60 maps using the Eucalyptus cloud. HybS also gives approximately 1.3x faster performance than FairS. Figure 6 shows that the overhead of these two jobs on the Virtual cluster as compared to same hardware configuration without virtualization. We used the same number of Map and Reduce slots in both configurations. We see the overhead is significant for the shuffle (7.5x slower) and reduce phases (8.2x slower). The reason is the rate of I/O and network I/O for virtual images is an order of magnitude slower than for of physical system. All virtual nodes share the same physical I/O and network bandwidth.

B. Simulation results

In addition to running our proposed scheduler on physical and virtual systems, we developed a simulator that allows us to investigate the optimal response time of a given workload for specific α, β, and γ values, as well as the effects of the service level value and system size for specific classes of jobs.

The simulator takes as input the fixed α, β, γ parameters, system size, and a workload description file formatted as follows:

<jobid> <number of tasks> <average time per task> <wait time> <service level value>. The simulator output consists of two files, a time based trace file for all of the states of the run and waiting queues and an Attribute Relationship File Format (ARFF) summary, which can be analyzed by the WEKA[11] Machine learning toolkit.

The simulator uses a greedy fractional knapsack algorithm that runs a subset of tasks for each job based on an ordered queue of prioritized jobs.

We make several assumptions in our simulations. All job wait times start at 1, as we assume the workload is a single group of jobs that all arrive to be scheduled at the same time and have no prior expectation of FIFO, as opposed to determining arrival rates with distributions for a given workflow.

There is no backfill and no preemption. Thus jobs will not be stopped if they haven't completed, and no jobs will be scheduled around a larger job if the large job has the highest priority. Because jobs are almost always split into independent tasks, even if the scheduler cannot satisfy a relatively high proportional service level value, say of 0.9, we guarantee that at least one task from the highest priority job will be scheduled, if there is room. Thus the large, high priority job, will not accrue wait-time, but will slowly eat up the available resources at the rate they become available.

In addition to these assumptions, system utilization is always at 100% because of our fractional knapsack approach.

Each Job within a workload can be classified into one of four groups based on their size, number of tasks, and duration time in seconds relative to the average for the workload. The classes are; small and short (SS), small and long (SL), large and short (LS) and large and long (LL). Figure 7 shows the classification for the first 20 jobs from the mixed workload outlined in experiment 1.

Response time is the value we would like to minimize for all jobs. Response time is a function of the time it takes to run the job including the time the job spent waiting, this can be formulated as an

$$expansion\ factor = \frac{(runTime + waitTime)}{runTime}$$

Ideally, jobs will have an expansion factor of 1.0, which means their response time is their runtime, with 0 wait time.

We use these terms interchangeably in our presentation of the simulation results.

Figure 9 shows simulation results for the experiment 1 data, run with **α=0 β=-1 γ=-1,** and **α=1 β=-1 γ=-1.**

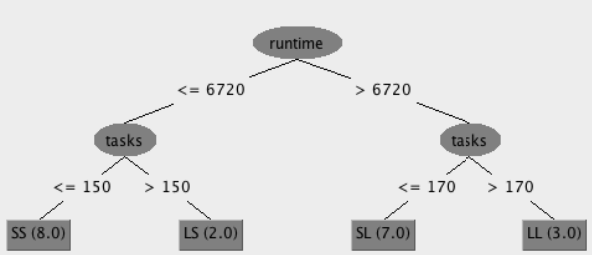

Figure 7: Summary of experiment 1 workload into job classes.

Figure 8. Expansion factor as a function of job size and α, β, γ values the second y axis is job size.

Figure 8 shows the relationship between the job size and the variation of α from 0 to -1. We see that when α=1 (the red line), the expansion factor is slightly lower. This is because the scheduler emphasizes wait time. The average response time is decreased by 7% when α=1 vs α=0. This is due to the large jobs, (jobs 4,13,14,15) getting a higher priority in the α=1 case. These results are consistent with our physical results in Hadoop for the experiment 1.

We also use simulation to identify the relationship between the service level value and response time. We would expect that a higher share value would get the job completed sooner if the emphasis of the α,β,γ values do not prohibit the job from having an increased priority. We studied the response time for job groups as a function of α,β and γ. We ran the simulator with a 20 job workload description file from experiment 1 and used values between 0 and 1 for each of the α,β,γ values, and ran the workload to completion, this resulted in 10^3 workload combinations. For each job we looked at the best and worst response times with the associated parameters. The α,β,γ values are different across job classes. For large long (LL) jobs the average minimum response was 1.0 with values in the range of α=(0.1-1.0), β =(0-0.6) and γ = (0.1–1.0). For the two large and short jobs (LS), the optimal response time of 1 was achieved with α=0.8, β =0.3 and γ = (0.1–1.0). For small and long (SL) jobs we observed an average response time of 1.7, with small and short jobs (SS) an average response time of 14.8 with α=(0.1-1.0), β = 0 and γ = 0. The high response times for SS and SL jobs are due to their short runtimes, the reasoning being that if the job only runs for a few seconds, then waiting for several times that amount, still isn't enough to move it to the top of the queue where larger jobs are waiting with higher priorities. Regarding workload simulation, future research will involve identifying multi-objective optimization algorithms for determining the best parameter combinations with the relative influence of the service level on job completion time within each class and overall workload completion time.

IV. RELATED WORK

Scheduling literature is a well-studied area of High Performance Computing [12]. Batch schedulers such as Torque are very popular in parallel and distributed environments, even though they do not support dynamic job allocations or resource changes over time. Instead, batch schedulers use priorities and backfill techniques to improve system utilization [13]. On Grid environments, Scheduling is also based on resource consumption and heterogeneous resource awareness. The task runtime at each node and the data transfer time are used for plan optimization. The data transfer time also critical in this model since data resides on geographically distributed sites [14, 15].

However scheduling in the MapReduce environment is a more recent development and is a subject of active research. Hadoop is by far the most dominant open source MapReduce system for large scale distributed computation and storage. Hadoop uses the simple FIFO algorithm by default. However, as the Hadoop community has matured, the Fair Scheduler and Capacity Scheduler provide share mechanisms for the Hadoop cluster among users, and improve latency in a multi-user environment. Fair Scheduler allocates equal resource shares to each of the users running the MapReduce jobs [4]. For a given user, there is no additional scheduling optimization beyond fairshare. In addition, the scheduling decision is not adapted dynamically for jobs as the jobs are in progress, since FairS focuses on the fairness and data locality [4].

FairS maintains data locality by delaying the Map task if there is no data currently available. This constant time delay *'D'* is configurable and needs to be adjusted for different workloads and resource types [4]. We use a similar technique for data locality. However, HybS automatically adjusts the tasks assignment by dynamic priority. If a job is skipped in task assignment, its wait-time is increased. Thus, its priority will be increased and likely be considered at the next heart-beat. As the tasks of a given job are completed, the number of remaining tasks for that job is reduced, which further affects the job's priority during runtime.

Zaharia proposed LATE (Longest Approximate Time to End) scheduler to robustly improve performance by reducing overhead of speculation execution tasks [16]. The technique works for MapReduce in heterogeneous environment. This technique is complementary to HybS for targeting a heterogeneous environment.

Sandholm and Lee (2010) evaluated ynamic Proportional (DP) share scheduling in Hadoop to allow user bids for map and reduce slots and controlling their spending of those slots over time [17]. Their DP system allows dynamic control over

the resource spending and allocation to provide quality of service. However, they were less concerned with the improvement of system utilization, application performance, or data locality.

Polo et al. (2010) provides an online profiler for MapReduce job completion time. They use this profiler to adjust resource allocations for different jobs [18]. However at the beginning of job execution, there is no information for online profiling. We also use an online profiler for average of map running time if there is no information available in the job history logs.

Regulated dynamic priority for MapReduce scheduling is a method using assigned priorities to offer different service levels for users over time [19]. Their dynamic priority aims to let users change their job priority overtime. The technique imposes regulation to prevent users from playing games with the priority. Our HybS uses dynamic priority from the system performance perspective and provides service level values for the users.

Kernel Canonical Correlation Analysis is used to predict the performance of MapReduce workloads [8]. This model is proposed for Hive queries and they suggest the use of this information for MapReduce schedulers. We currently use the average of map runtime in the job histories and in the future work, we plan to use this model to predict the MapReduce performance for HybS.

Phan et al. (2010) is focused on providing a scheduler for MapReduce jobs based on their deadlines [20]. This algorithm also falls into the dynamic scheduling category, and their approach is tied to the constraint satisfaction problem (CSP). However, they do not consider the dependencies between Map/Reduce phases that would make the CSP more complex.

V. CONCLUSIONS AND FUTURE WORK

A Hybrid Dynamic Priority scheduling algorithm, HybS, is presented for the Hadoop MapReduce environment. We show that HybS allows for a flexible policy-driven scheduling system that improves response time by an average of 2.2X with a standard deviation of 1.4 for two workload experiments using Hadoop's FairS on a 9 node cluster. HybS achieves this improved response time by means of relaxing the strict proportional fairness with a simple exponential policy model. We show how the default HybS parameters quickly resolve the data dependence between the Map and Reduce phases by improving the wait time of the last Map task on average. Additionally, we show simulation analysis of the optimal service level value and policies for overall performance as well as response time under a variety of conditions. In conclusion, HybS is a fast and flexible scheduler that improves response time for multi-user Hadoop environments. Future work will include evaluating the effects of the service level value on the total workload completion time. Additionally we will integrate the simulator into the HybS scheduler code in Hadoop to act as a workload profiler.

ACKNOWLEDGMENTS

This work is supported in part by Center for Hybrid Multicore Productivity Research, UMBC/CSEE, and an NSF CORBI grant between CHMPR/MC2 and CHREC/GWU. Thanks also to Navid Golpayegani for his work building our initial Eucalyptus cloud testbed on the CHMPR bluegrit cluster.

REFERENCES

[1] NIST big data workshop. http://www.nist.gov/itl/ssd/is/big-data.cfm June 2012.

[2] http://hadoop.apache.org/

[3] J. Dean and S. Ghemawat, “MapReduce Simplified data processing on large clusters,” In OSDI, pages 137-150, 2004.

[4] M. Zaharia et al. “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling,” In EuroSys 2010.

[5] Yunhong Gu and Robert Grossman, “Lessons Learned From a Year's Worth of Benchmarks of Large Data Clouds,” 2nd Workshop on Many-Task Computing on Grids and Supercomputers Portland, Oregon 11, 2009.

[6] Y. Chen et al. “The Case for Evaluating MapReduce Performance Using Workload Suites,” In MASCOTS 2011.

[7] W. Ward, Jr., Carrie L. Mahood, and John E. West, “Scheduling jobs on parallel systems using a relaxed backfill strategy,” Job Scheduling Strategies for Parallel Processing, pages 88-102. Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537

[8] Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, “Statistics-driven workload modeling for the cloud,” In Proc. Of 5th Intl. Workshop on Self Managing Database Systems, 2010.

[9] D. Chapman, M. Halem,P. Nguyen, J. Avery, "Noise reduction in gridded AIRS Brightness temprature grids using the MODIS Obscov algorithm," IEEE IGARSS, Munich Germany, Jul, 2012

[10] P. Nguyen, M. Halem, “A MapReduce Workflow System for Architecting Scientific Data Intensive Applications,” SECLOUD 2011, May 22, 2011, Waikiki, Honolulu, HI, USAM.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, “The WEKA Data Mining Software: An Update,” SIGKDD Explorations, Volume 11, Issue 1 2009.

[12] Joseph Y-T. Leung, “Handbook of scheduling algorithms, models and performance analysis,” Chapman & Hall/CRC ISBN L - 58488-397-9. 2004.

[13] http://www.adaptivecomputing.com/products/open-source/torque/

[14] Arun Ramakrishnan, Gurmeet Singh, Henan Zhao, Ewa Deelman, Rizos Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, “Scheduling Data -Intensive Workflows onto Storage-Constrained Distributed Resources,” in Seventh IEEE International Symposium on Cluster Computing and the Grid - CCGrid 2007.

[15] Jim Blythe, Sonal Jain, Ewa Deelman, Karan Vahi, Yolanda Gil, Anirban Mandal, Ken Kennedy, “Task Scheduling Strategies for Workflow-based Applications in Grids,” IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)

[16] Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I., “Improving MapReduce performance in heterogeneous environments,” In: OSDI 2008: 8th USENIX Symposium on Operating Systems Design and Implementation 2008.

[17] T. Sandholm and K. Lai, “Dynamic Proportional Share Scheduling in Hadoop,” LNCS: Proc. of the 15th Workshop on Job Scheduling Strategies for Parallel Processing, 2010.

[18] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguad´e, M. Steinder, and I. Whalley, “Performance-driven task coscheduling for MapReduce environments,” in 12th IEEE/IFIPNetwork Operations and Management Symposium, 2010

[19] T. Sandholm and K. Lai, "Map-Reduce optimization using regulated dynamic prioritization," presented at the eleventh international joint conference on Measurement and modeling of computer systems, Seattle, WA, USA, 2009.

[20] L. Phan, Z. Zhang, B. Loo, and I. Lee, “Real-time MapReduce Scheduling,” Tech. Report No. MS-CIS-10-32, UPenn, 2010.