Simulation and Performance Evaluation of the Hadoop Capacity Scheduler

Jagmohan Chauhan, Dwight Makaroff and Winfried Grassmann

Department of Computer Science, University of Saskatchewan
Saskatoon, SK, CANADA, S7N 5C9
jac735@mail.usask.ca, {makaroff, grassman} @cs.usask.ca

Abstract

Hadoop task schedulers like Fair Share and Capacity have been specially designed to share hardware resources among multiple organizations. The Capacity Scheduler provides a complex set of parameters to give fine control over resource allocation of a shared MapReduce cluster. Administrators and users often run into performance problems because they do not understand the performance influence of the task scheduler parameters on MapReduce workloads. Interaction between parameter settings is particularly problematic.

In this paper, we implemented a Capacity Scheduler simulator component, integrated it into an existing simulator and then validated the simulator with small test cases, consisting of standard benchmark sort programs. We then studied the impact of Capacity Scheduler parameters on different MapReduce workload submission patterns with a more complex set of benchmark programs. Among other results, we found *maxCapacity* and *minUserLimitPCT* to be influential parameters suggested by previous work and that using separate queues for short and long jobs provides the best performance in terms of response ratio, execution time and makespan compared to submitting both types of jobs in the same queue.

1 Introduction

Research and government organizations as well as corporations produce huge amounts of data for varying internal and external purposes contributing to the world-wide data explosion often referred to as "Big Data". Applications that access and process Big Data in a parallel manner to produce meaningful information are called data-intensive computing applications. Analysis and processing of Big Data is crucial to these organizations and *data-intensive computing* addresses this need. MapReduce [6] was the most popular of the new processing frameworks that emerged. An open-source framework based on MapReduce called Hadoop[1] emerged in 2007 and is used by organizations like Yahoo! and Facebook, and many others. Organizations that do not have the need for a private cluster often utilize a shared cluster on demand [17].

The Task Scheduler is an important part of the MapReduce framework. Initially, MapReduce was designed to handle batch-oriented jobs and a FIFO task scheduler was suited to handle such jobs. The emergent needs to have better data locality, better response time for short jobs, cluster sharing between different organizations/users led to the development of various other schedulers, like FairShare,[2] Delay aware [22], Capacity,[3] Quincy [10], etc. Recently, YARN [17] was introduced to the Hadoop framework which fundamentally redesigns higher level shared cluster resource management.

Several Capacity Scheduler configuration parameters are provided that affect how the resources in the cluster are shared between organizations and

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

[2] http://hadoop.apache.org/common/docs/r0.20.2/fairscheduler.html

[3] http://hadoop.apache.org/common/docs/r0.20.2/capacityscheduler.html

users within an organization. Unfortunately, setting these parameters properly to provide the desired sharing behaviour is a challenging task. Our previous work measuring the influence of Capacity Scheduler parameter settings found that only some parameters have significant impact on the performance of jobs [4]. It is important to estimate the magnitude of these consequences before making any such changes in a real production system; a Capacity Scheduler simulator would save both time and cost. Simulation can characterize the behaviour of running jobs on a MapReduce cluster across different job submission patterns, job mixes, network topologies and cluster configurations.

In this paper, we modified an existing simulator named MRPerf [20] and then validated it against a real cluster setup using representative MapReduce applications. The validation experiments exposed the factors which limited prediction accuracy in the simulator. The factors identified can be used to design a better simulator.

We then conducted a simulation study to determine the degree to which the parameters identified as being influential in our previous experimental study [4] affect performance measures when the size and configuration of a cluster and its workload are varied to larger scales. In particular, we study three different job submission settings: (i) a sequence of long jobs followed by a sequence of short jobs, (ii) interleaving jobs, and (iii) separate queues for long jobs and short jobs. For input to the simulation, job types are characterized by a small number of parameters: the number of map/reduce tasks and the size of input and output data. The current design of MRPerf abstracts away other facets of job and task functionality.

Our simulation results confirm that *Capacity*, *maxCapacity*, and *minUserLimitPCT* influence the execution and waiting time, but do not affect the data locality of the tasks scheduled on the processing nodes. Finally, we observe that separating long jobs and short jobs into disjoint queues with proper parameter settings improves the overall execution time (makespan) of the short job queues. These experiments are a case study and not a comprehensive evaluation of the variety of MapReduce use cases.

The remainder of this paper is organized as follows. Section 2 introduces the Capacity Scheduler and its various configuration parameters. Related work is covered in Section 3. Section 4 describes the Capacity Scheduler simulator implementation. In Section 5, we provide the simulator validation results. In Section 6 we present and analyze the results of additional experiments on a simulated medium-sized cluster. Finally, Section 7 shows our conclusions and future work.

2 Hadoop Scheduling

A Hadoop cluster consists of 3 main control components: a *dedicated NameNode* server to provide file system support, a *secondary NameNode*, which generate snapshots of the NameNode's metadata and a *JobTracker* server to manage job scheduling decisions using a task scheduling algorithm. The task scheduler runs on the job tracker and decides where and when the tasks of a job will be executed.

There are at least three well-known Hadoop task schedulers:[4]

- FIFO: All the users submit jobs to a single queue, and jobs are executed sequentially.
- Fair Scheduler: Resources are evenly allocated to multiple jobs with capacity guarantees. Jobs are placed in pools; each pool has a guaranteed capacity. Excess capacity is allocated between jobs using fair sharing.
- Capacity Scheduler: A number of named queues are configured. Each queue has a guaranteed capacity (i.e. number of map and reduce slots). Any unused capacity is shared between the remaining queues. FIFO scheduling with priorities is used in each queue. Limits can be placed on the percent of running tasks per user, so that users share a cluster equally.

The Capacity Scheduler operates according to the following principles:

1. At cluster startup, a configuration file with all the task scheduler settings for the queue characteristics and other operations is used to initialize the system.
2. An initialization poller thread is started along with worker threads. The poller thread wakes up at specified intervals, distributes jobs to worker threads and then sleeps.
3. At job submission time, the scheduler checks job submission limits to determine if the job can be accepted for the queue and user.

[4] http://en.wikipedia.org/wiki/Apache_Hadoop

4. If the job can be accepted, the initialization poller checks initialization limits. If initialization is permitted, a worker thread is assigned to initialize the job.

5. Whenever the JobTracker gets a heartbeat signal from a TaskTracker indicating that a node is available, a queue is selected from all the queues with available jobs by sorting the queues by current settings and load. Queue- and user- specific limits are checked for capacity limits. The first permissible job is chosen from the chosen queue. Next, a task is picked from the job with preference given to tasks with data closer to the nodes. This procedure is repeated until all the jobs complete.

Note that the Capacity Scheduler dynamically assigns tasks to threads and nodes, respectively. In particular, tasks have the following relationships with data location: *node-local* tasks have the data on the local disk, *rack-local* tasks have data on a machine in the same rack and network data transfer is necessary to complete the task, and finally, *remote* tasks have data further away in the cluster.

Many configuration parameters are provided, allowing administrators to tune scheduling parameters for the jobs. Table 1 shows the configurable resource allocation parameters that we consider in our experiments. Our initial exploration of the effect of task scheduler parameters via experimentation on a small cluster [4] showed that these were the most influential in elapsed time performance. A value of -1 for a parameter means the parameter is not used. There are other parameters that are not shown which are categorized under memory management and job initialization.

Table 1: Capacity Scheduler Parameters

Parameter Name	**Brief Description/Use**	**Default Value**
queue.*qname*. Capacity	Percentage of cluster's slots available for jobs in queue.	1 queue @100%
queue.*qname*. maxCapacity	Maximum percentage of cluster capacity for a single queue.	-1
queue.*qname*. minUserLimit-PCT	A limit on the percentage of resources allocated to a user at any given time, if there is competition.	100
queue.*qname*. userLimitFactor	Capacity multiple for user to acquire more slots.	1

3 Related Work

Jiang, Ooi, Shi and Wu [12] studied the performance of MapReduce in a very detailed manner. They identified five design factors that affect Hadoop performance: 1) grouping schemes, 2) I/O modes, 3) data parsing, 4) indexing, and 5) block-level scheduling. By carefully tuning these factors, the overall performance improved by a factor of 2.5 to 3.5. Babu [2] measured a real system and found that the presence of too many job configuration parameters in Hadoop is cumbersome, highlighting the importance of an automated tool to optimize parameter values. Herodotou *et al.* [9] introduced Starfish, which profiles and optimizes MapReduce programs based on cost. Starfish aims to relieve users from having to fine tune job configuration parameters for different cluster settings and input data sets. All these profiling tools require a real system on which to test potential workloads and are tuned to the job scheduler and not the task scheduler.

Experiments with YARN utilized the Capacity Scheduler with particular parameter settings [17]. Production results at Yahoo! for a small cluster (10 machines) show the benefits of the YARN framework with a recently developed capability of YARN: work-preserving preemption. This is not modelled in current simulators, but shows the need for some method of evaluating scheduler performance at scale.

Several discrete event simulators for MapReduce application workloads have been developed. Each provides different levels of support for integrating new task schedulers and different details in the computation and communication models.

SimMR [18], MRSim [8] and SimMapReduce [16] are designed to evaluate different schedulers/provisioning strategies. SimMR focuses on JobTracker decisions and task/slot allocations among different jobs. MRSim measures scalability easily and captures the effects of different configurations of Hadoop setup on job completion times and hardware utilization. SimMapReduce allows researchers to evaluate different scheduling algorithms and resource allocation policies.

Hsim [13] simulates the dynamic behaviours of Hadoop environments and models a large number of Hadoop parameters such as node parameters (related to processors, memory, hard disk, network interface, map and reduce instances) and cluster parameters, (number of nodes, node configurations,

network routers, job queues and schedulers), and Hadoop system parameters. Mumak [14] can estimate performance of MapReduce applications under different schedulers. It takes a job trace derived from production workload and a cluster definition as input, and simulates the execution of the jobs using different schedulers. The detailed job execution trace (recorded in relation to virtual simulated time) is the output that can be analyzed to capture various traits of individual schedulers (turn around time, throughput, fairness, capacity guarantee, etc.).

MRPerf [20, 21] analyzes application performance on a given Hadoop setup, enabling the evaluation of design decisions for fine-tuning and creating Hadoop clusters. MRPerf was made open source to be used by the research community to enable exploration of design issues, validation of new algorithms and optimization in MapReduce.

The need for production level traces by some simulators makes them inappropriate for general research, since often the traces are proprietary information and not easily available to the academic community. Only recently have MapReduce simulators supported schedulers other than FIFO, Mumak being the first.

A new simulation environment, SLS[5] has been recently introduced by Yahoo! for the YARN framework includes support for many components within Hadoop. SLS provides detailed execution traces as well as resource usage metrics. It even provides analysis of low-level scheduler operations, measuring their overhead and assessing scalability. The environment is designed in a modular fashion to incorporate new scheduler development. There are many parameters in the simulator itself. The goals and approach of our work and SLS are similar, but SLS was not a mature tool at the beginning of our investigation.

Optimizing cluster utilization and reducing makespan has been studied using job scheduler adjustments. HFSP [15] uses a run-time estimate of job size and age to order jobs; pre-emption prioritizes shorter jobs without starvation. Verma *et al.* [19] use the FIFO scheduler and choose to order the jobs optimally to reduce makespan. While there has been a rich history of scheduling research in the area of cluster and grid computing, it is beyond the scope of this work to evaluate the design of such scheduling algorithms.

[5] http://hadoop.apache.org/docs/r2.3.0/hadoop-sls/SchedulerLoadSimulator.html

4 Simulator Implementation

The main code to implement the Capacity Scheduler simulator includes 1) queue setup, 2) initialization poller configuration, 3) job launching and initialization, 4) the main scheduler driver and 5) job removal. A new config file was created containing parameter settings and a conversion program written to support multiple users/new job types. The existing main logic in MRPerf interfaces with the scheduler code. The stages of MapReduce processing are coded as front-end TCL files. Interface code was required for the Capacity Scheduler code as was support for multiple heartbeats for reduce tasks. The Capacity Scheduler component required 500 lines of Python/TCL interface code and 2000 lines of C++ code for processing.

The MRPerf simulator lacked variability between simulation runs. A fixed set of jobs always produced the same results because the heartbeat arrival time at the JobTracker was at fixed times and in the same order as the DataNodes generated initial heartbeats. This is not realistic as there is always some randomness in heartbeat arrivals to the JobTracker due to operating system scheduling and network communication delays. A random seed was added to the time at which a node can generate its first heartbeat to create randomness in the system in two ways: a node can provide an initial heartbeat to the JobTracker at a random time, and no fixed order was placed on the nodes with respect to the initial heartbeat. The subsequent heartbeat from a node is generated a multiple of 300 ms later.

5 Simulator Validation

5.1 Setup

The experimental cluster used for validation experiments consisted of 5 nodes on local lab machines. One node was the JobTracker and NameNode while the other nodes served as TaskTracker and DataNodes. Each node had a 2.4 Ghz CPU, 4 GB of RAM and 250 GB hard disk, running Ubuntu 10.04 Linux, executing Hadoop 0.20.203. The data replication factor was 3. All nodes were on same rack and connected through a 1 Gbps switch. The simulator validation was conducted on a similarly-configured LINUX laptop.

In the validation experiments, *Sort* and *TeraSort* (called TSort hereafter) were used as the jobs as their output data is proportional to the input data

(a requirement for MRPerf); this kept the validation simple to implement and evaluate. Two queues were used, with 2 users per queue. Jobs in each queue were submitted with an inter-arrival time of 5 seconds, while jobs of the same type in different queues were submitted at the same time. *TeraGen* generated the input data for *TSort* (*RandomWriter* for *Sort*). *TSort* used 2 GB of input data, while *Sort* used 1 GB of input data, so as to have different input data sizes for different types of jobs, but to keep the dataset size intentionally small and limit the time required for validation. We generated jobs in the following order:

- *TSort* for user 1/user 3 in Queue 1/Queue 2.
- *TSort* for user 2/user 4 in Queue 1/Queue 2, respectively, after 5 seconds.
- *Sort* for user 1/user 3 in Queue 1/Queue 2, respectively, when their *TSort* job is finished. The same is done for user 2/user 4 in Queue 1/Queue 2 respectively.

Two different node configurations were used: one reduce slot per node and two reduce slots per node. We captured map, reduce, execution and waiting time. The graphs show the mean of 10 runs and the 95% confidence interval as error bars. "Job/Number" is used for identification purposes in both the graphs and the descriptions. Detailed comparison of phase execution times is necessary to identify sources of simulator inaccuracy. Chauhan [3] contains more experimental results.

5.2 One Reduce Slot per Node

Changing numQueues. In the first experiment, a single queue with 2 users and 4 jobs in total was used. In the second experiment, an identical queue was added (*Capacity* set at 50% for each queue, but not modifying *maxCapacity*). Figure 1 shows the results. With two queues, the map/reduce/total execution times increase for all jobs.

The confidence interval for the map execution time for *TSort/2* and *TSort/4* was large. The reason for this was *delayed map execution*, which occurs when a job is started before it should be, given the scheduling discipline in place. If job 1 is submitted before job 2, then job 2's map tasks do not execute until job 1's map tasks are finished or there are extra slots remaining idle. In some runs, job 2 gets one or two map slots quickly (even though job 1's map tasks were not finished). Later, when the JobTracker received new heartbeats from the TaskTracker, job 1 took preference as it was submitted earlier and until job 1 completed all its map tasks, there were no job 2 map tasks scheduled.

This occurred only once for *TSort/2* and *TSort/4* in the real system. These jobs almost always exhibited delayed map execution behaviour in the simulator. This leads to huge differences in execution and waiting time. The results match closely on the runs without delayed map execution. Delayed map execution differences are caused by heartbeat arrival times. The other major factor for *TSort*'s variation was straggler tasks in some real system runs when resources cannot be obtained in time to complete with other map tasks [1].

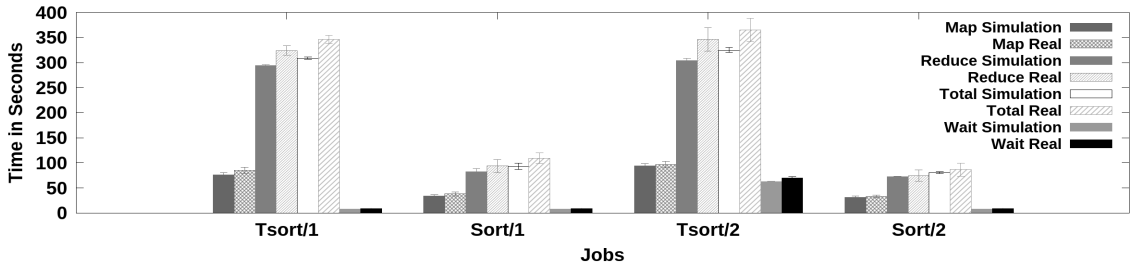
Delayed map execution also leads to increased reduce time in some executions. The reduce phase is started when a certain percentage of map tasks are completed. By default, it is 0.05% of the total map tasks. *Sort* with 1 GB of input data has 16 map tasks; one map task completion triggers the reduce phase. However, the reduce phase cannot proceed fully until the map phase is completed, which will be delayed until any preceding job completes all its map tasks. Straggler map tasks affect the reduce execution of all jobs.

Changing minUserLimitPCT. The *minUserLimitPCT* parameter was changed to 50%, allowing concurrent execution of map tasks for different jobs. All map, reduce and overall execution times increased. This trend was captured by the simulator. The high variation for reduce tasks in the real cluster shown in Figure 2 is due to stragglers.

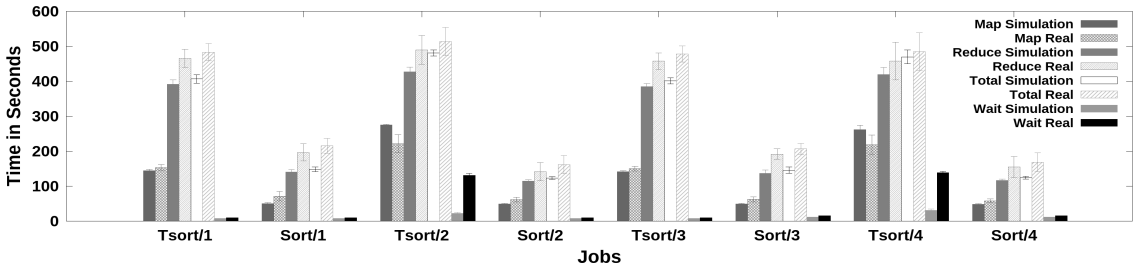
5.3 Two Reduce Slots per Node

In this configuration, some jobs' reduce tasks were executed in parallel on the same node (the *same-reduce-node effect*). Two different graphs are shown, one with reduce phases in isolation, and one when paired with another job on the same node, as the reduce time increases substantially in the paired configuration, skewing the average elapsed times. Table 2 shows the number of isolated-paired cases for the real cluster and simulation runs when the parameters were varied, respectively. Heartbeat arrival times affect the relative frequency of the same-reduce-node effect.

Changing numQueues. The same experimental parameters were used as with one reduce slot per



(a) 1 queue 100% capacity



(b) 2 queue 50% capacity each

Figure 1: *numQueues* (one Reduce Slot per Node)

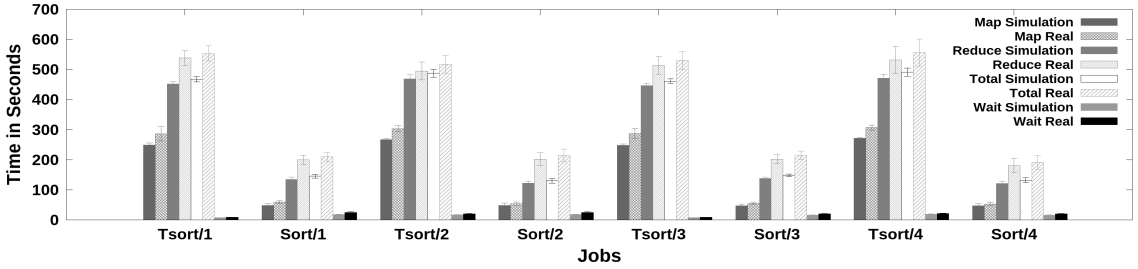
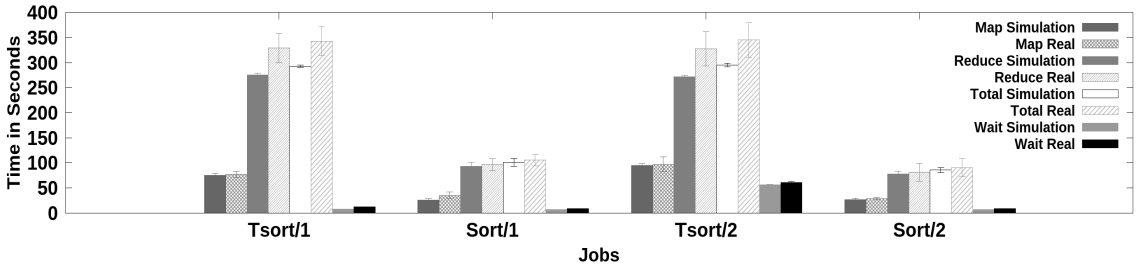


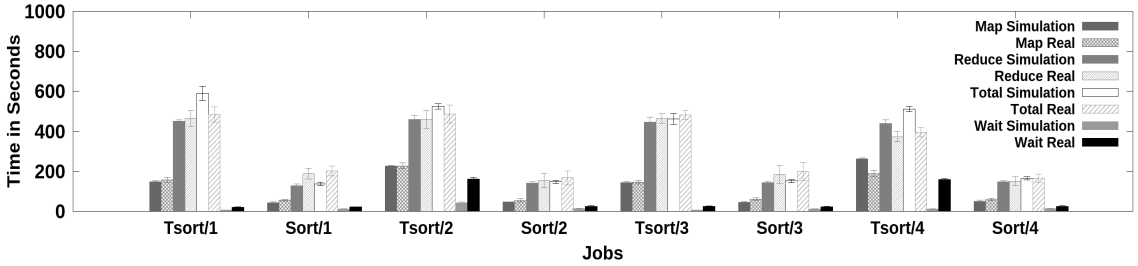
Figure 2: *minUserLimitPCT=50%, Capacity=50%* (One Reduce Slot per Node)

node. The results are shown in Figure 3. Single queue results matched well for *Sort*, but not for *TSort*. A small standard deviation exists because of straggler tasks in some real cluster runs. With the additional queue, map, reduce and total execution time increases for all jobs and differeces appear between the real system and the simulation in both applications. Delayed map execution happened once in the real cluster for *TSort/2* and *TSort/4*. Delayed map execution occurred in all simulation runs. This helps explain the variation for *TSort/2* and *TSort/4*. In the real cluster, *TSort/2* and *TSort/4* started when the map phase for *TSort/1* and *TSort/3* ended. In the simulation runs, they started map execution at submission time, leading to different waiting times.

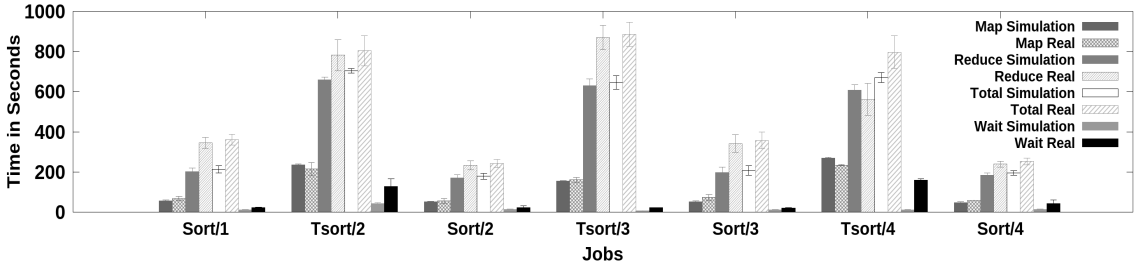
Differences in the reduce execution time in Figure 3(c) can be attributed to the same-reduce-node effect. The different scale on the graph shows that the paired reduce is much slower. Two independent factors cause the variation: 1) *which* job executes in parallel with which other job, and 2) *when* the jobs pair with each other. In the real cluster, *TSort/3* in particular was always paired with another *TSort* job and as *TSort* jobs execute longer, the reduce execution time for *TSort/3* is much higher than all other jobs. All other *TSort* jobs were paired with *TSort* as well as *Sort* jobs, reducing execution times less than *TSort/3*. If two jobs start their reduce phase on the same node simultaneously, they will be affected by competition for resources. If the reduce phase



(a) 1 queue 100% capacity Cluster



(b) 2 queue 50% capacity Cluster - Isolated reduce



(c) 2 queue 50% capacity Cluster - Paired reduce

Figure 3: *numQueues* (Two Reduce Slots per Node)

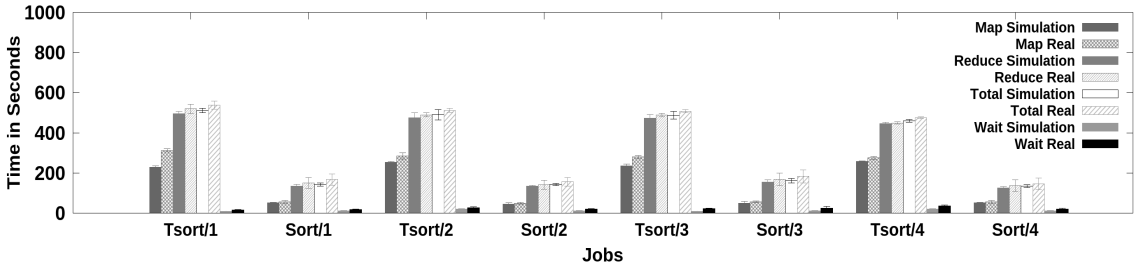
starts later for the second job, there will be less overlap. Discrepancies between simulation and the real system can be attributed to heartbeat arrivals.

Changing minUserLimitPCT. The identical experiment was conducted as with one reduce slot per node. The results in Figure 4 show increasing map, reduce and execution times captured by the simulator well only in the isolated reduce case. The difference in the means and large variation for reduce tasks in the paired-reduce phase is due to the same-reduce-node effect and delayed map execution.

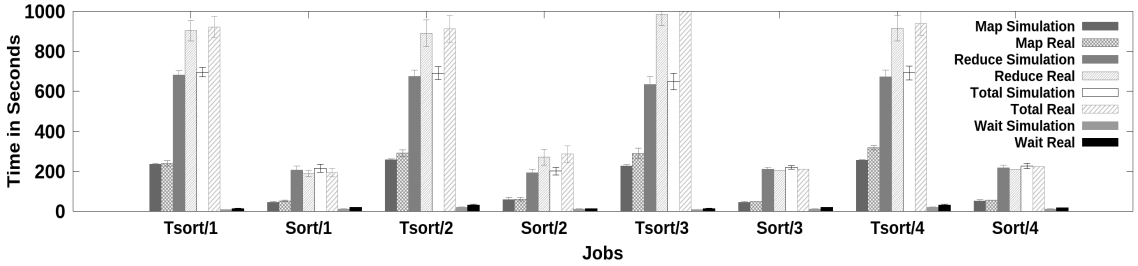
5.4 Analysis/Summary

The simulator depicts all behaviour observed on the real cluster for which there is a scheduler component modelled. The best results occur with *minUserLimitPCT* set to 50% and 2 reduce slots per node as all simulator averages were within the confidence interval of the real system. The worst results came with paired-reduce behaviour.

Several factors limited the simulator accuracy. First, straggler tasks (which are *not* modelled in the simulator), and the same-reduce-node effect occurring more frequently in the simulator increased variability. Second, the simulator does not model all the different sub-phases of the map/reduce phase. Factors like disk access time, CPU and network contention, prefetching etc. are not modelled in the simulator. Stragglers and same-reduce-node effect modelling is part of future work.



(a) Isolated reduce phase



(b) Paired reduce phase

Figure 4: *minUserLimitPercent* (Two Reduce Slots per Node)

Table 2: Isolated-Paired Reduce Runs

	Real Cluster							
Factor	TSort 1	Sort 1	TSort 2	Sort 2	TSort 3	Sort 3	TSort 4	Sort 4
Capacity	10/0	6/4	5/5	8/2	7/3	7/3	7/3	7/3
max Capacity	9/1	6/4	7/3	8/2	8/2	7/3	5/5	9/1
minUser Limit-PCT	6/4	8/2	7/3	7/3	6/4	9/1	7/3	9/1
	Simulation							
Factor	TSort 1	Sort 1	TSort 2	Sort 2	TSort 3	Sort 3	TSort 4	Sort 4
Capacity	2/8	4/6	3/7	7/3	3/7	5/5	5/5	4/6
max Capacity	2/8	3/7	7/3	4/6	3/7	6/4	4/6	4/6
minUser Limit-PCT	4/6	3/7	3/7	5/5	2/8	4/4	4/6	6/4

6 Simulation Experiments

A 31-node cluster was simulated under different job submission patterns. Each node was configured identically to the real cluster nodes. This is similar in size and configuration to Pastorelli *et al.* [15]. We limited ourselves to the parameters defined in Table 1 [4]. There were no discernible effects of *UserLimitFactor* in the experiments and so, there are no results presented based on this parameter.

The experiments were designed based on three questions: 1) What Capacity Scheduler settings can optimize the performance of short jobs when they arrive after a sequence of long jobs? In particular, are there settings that allow some short jobs to be executed while the majority of the cluster's resources are occupied by long jobs? 2) What are the effects of Capacity Scheduler settings where a long job and a series of short jobs arrive in an interleaved fashion? 3) Does providing different queues for short and long jobs improve system performance and the performance of short jobs in particular?

Two queues were used, each containing 50 jobs as described in Table 3, motivated by Chen *et al.* [5]. A 1 second inter-arrival time was used. The shuffle ratio is the amount of data generated by the map phase that needs to be sent to the reduce phase relative to the input data, while the output ratio is the size of the output file compared to the input file. Map compute and Reduce compute are multiplicative factors representing the processing time needed per task compared to the smallest job.

A full-factorial experiment with 10 replications was performed with settings as shown in Table 4.

Table 3: Job Types

Job Type	Description	% of job mix	Map Tasks	Reduce Tasks	Shuffle ratio	Output ratio	Map compute	Reduce compute
1	Data transformation	8	100	15	1	1	5	40
2	Aggregate and expand	6	200	1	0.025	3	40	5
3	Expand and aggregate	4	400	30	3	0.025	40	40
4	Data summary	2	800	8	0.075	0.0005	20	20
5	Small job	48	1	1	1	1	1	1
6	Small job	24	2	1	1	1	1	1
7	Small job	8	10	1	1	1	1	1

We defined a *short job* as one having 10 or fewer map tasks. We restricted *long job* sizes to those considered **large** (less than 50 GB), ignoring those which were denoted as **very large** or **huge** [5]. Including these job types would have totally dominated any scheduler settings effect, and are so rare (less than .002% in the Facebook trace) that the workload would need to be 3 orders of magnitude larger to consider their relative occurrence. We believe this does not affect our ability to evaluate scheduler performance on common workloads.

Table 4: Parameter Settings: Sequential Job Types

Exp No	Num Queues	Capacity	Max Capacity	User Limit Factor	Min User Limit PCT
1	2	50-50	-1	1	100
2	2	50-50	50-50	1	100
3	2	50-50	-1	2	100
4	2	50-50	-1	1	25
5	2	50-50	50-50	2	100
6	2	50-50	50-50	1	25
7	2	50-50	-1	2	25
8	2	50-50	50-50	2	25
9	2	70-30	-1	1	100
10	2	70-30	50-50	1	100
11	2	70-30	-1	2	100
12	2	70-30	-1	1	25
13	2	70-30	50-50	2	100
14	2	70-30	50-50	1	25
15	2	70-30	-1	2	25
16	2	70-30	50-50	2	25

To answer the first question, long jobs were submitted followed by short jobs. To answer the last two questions, two types of job submission patterns were submitted. In the *interleaved* case, 4 short jobs were submitted and then a long job until all 50 jobs were submitted with two equal capacity queues. Each user had a single job in the system. In the *separate queue* case, each queue was split in a 4:1 ratio, providing 4 queues and allocating more slots to the long job queues. Table 5 contains parameter settings for these experiments.

Table 5: Parameter Settings: Separate Job Queues

Exp No	Num Queues	Capacity	Max Capacity	User Limit Factor	Min User Limit PCT
17	2	50-50	-1	1	100
18	2	50-50	50-50	1	100
19	2	50-50	-1	2	100
20	2	50-50	-1	1	25
21	2	50-50	50-50	2	100
22	2	50-50	50-50	1	25
23	2	50-50	-1	2	25
24	2	50-50	50-50	2	25
25	4	10-10-40-40	-1	1	100
26	4	10-10-40-40	10-10-40-40	1	100
27	4	10-10-40-40	-1	2	100
28	4	10-10-40-40	-1	1	25
29	4	10-10-40-40	10-10-40-40	2	100
30	4	10-10-40-40	10-10-40-40	1	25
31	4	10-10-40-40	-1	2	25
32	4	10-10-40-40	10-10-40-40	2	25

Five important metrics are reported: *data locality*, *response ratio*, *elapsed time*, *coefficient of variation* in job execution time and *makespan*. Data locality is the percentage of node-local tasks. Response ratio is defined as the ratio of execution time to waiting time. Finally, makespan is the elapsed time until all jobs are completed.

We model the system from an initially empty setting. Comprehensive full-scale steady state sim-

ulations would require a much larger job pool and computation effort to evaluate. As well, this only addresses an interleaved scenario, in which jobs of any type can arrive. With temporally segregated job types, steady state measurements are not informative, and makespan is not relevant. Our results will be relevant in a system that experiences temporary load reductions where resources are not always fully subscribed and queues occasionally empty. We show how long the system will be utilized when given a specific set of tasks. For example, our input could be the daily set of jobs submitted by 2 organizations sharing the cluster submitted at a certain time of day. We are interested in confirming which parameter settings have an effect on these sample job submission patterns.

6.1 Equal Capacity Queues

Data Locality. The experiments in the simulation were done on a single rack. Data locality plays an important role in MapReduce environments. Reduced network activity for node-local tasks should reduce latency. For small jobs, the percentage of node-local map tasks scheduled is moderate to small (between 39% and 47% for Job Type 7, and between 6% and 12% for Job Type 5). Fewer node-local tasks means less data locality. For jobs with 1 map task, the distribution was bi-modal (either node-local or rack-local). Similarly, for jobs with 2 map tasks, the distribution was tri-modal (0, 50 and 100%). For large jobs, the percentage of node-local tasks is high (from 88% for Job Type 1 to 98% for Job Type 2). As the number of map tasks increases, the data is more widely distributed and nodes that have the data locally can be found.

Changing parameter settings does not significantly alter data locality in nearly all cases. The parameters provide stringent limits to ensure that a single job/user/queue cannot consume a disproportionate amount of resources. Data locality depends on which TaskTracker gets assigned a task and whether the data is local. The decision to give a task to a TaskTracker is determined by the JobTracker running the task scheduler, and is designed to be independent of Capacity Scheduler parameter settings. The absolute percentage of node-local tasks varies less than 2% for long jobs. For short jobs, the relative difference is at most 20% (from 39% to 47% for job type 7 between experiment 1 and experiment 2).

Response Ratio. In Figure 5, we present the execution and waiting times for selected job types. The smallest execution time is for Job Type 5, but in many of the configurations, an extreme amount of time is spent waiting. Table 6 shows the average response ratios for all the job types. The waiting times for all job types decrease when *minUserLimitPCT* is modified (Experiments 4, 6, 7 and 8). As well, the response ratio is improved, though less for the long jobs. Setting *minUserLimitPCT* allows more jobs to execute concurrently. The lower value

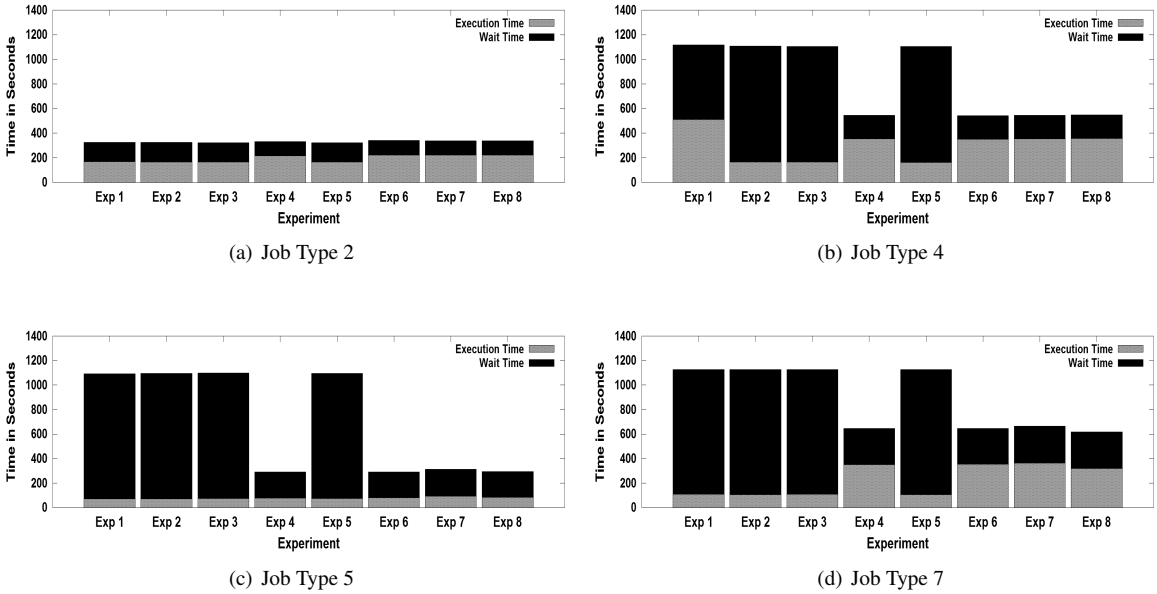
Table 6: Response Ratio: Equal Capacity

	Experiment							
Job Type	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1.2	1.2	1.2	1.0	1.2	1.0	1.0	1.0
2	1.9	2.0	2.0	1.5	2.0	1.5	1.5	1.5
3	1.2	1.5	1.5	1.1	1.5	1.2	1.1	1.1
4	2.1	6.7	6.7	1.5	6.8	1.5	1.5	1.5
5	15.0	15.0	14.0	3.0	14.0	3.6	3.2	3.5
6	12.8	12.9	12.8	3.0	12.7	3.0	2.7	3.0
7	10.3	10.4	10.3	1.8	10.5	1.8	1.8	1.9

for response ratio in experiment 1 for Job Type 4 is an artifact of delayed map execution. The benefit in elapsed time for most of the job types (in particular, Job Types 4 and 5) indicates these are preferred operating configurations.

Elapsed Time. Users are interested in elapsed time. Does a low response ratio lead to reduced elapsed time? Execution time increases for all job types when the response ratio decreases due to *minUserLimitPCT* changes, because of increased competition between the jobs. The improvement in response ratio does not change elapsed times for Job Type 2 (in fact, that increases slightly). For Job Type 4, elapsed time decreases greatly, because of the factor by which the response ratio improves. The response ratio changes by only 25% for Job Type 2. The response ratio for Job Type 4 improves by a factor of 4; thus, improving elapsed time. This improvement averages a reduction to 50% of the original elapsed time. For Job Type 5, elapsed time decreases by a factor of 4.

Execution Time Variation. The coefficient of variation in execution time for long jobs was less than 0.1. For short jobs, it lies in the range of 0.1-0.5 (Table 7), due to data locality. Node-local data usually provides very fast map times, but node-local tasks may take *longer* than rack-local tasks if there are co-located map/reduce tasks from the



(a) Job Type 2

(b) Job Type 4

(c) Job Type 5

(d) Job Type 7

Figure 5: Execution And Wait Times For Selected Job Types: Equal Capacity Queues

same or different jobs due to disk contention. Finally, remote disk contention may occur.

Table 7: Coefficient of Variation: Short Job Types

	Experiment							
Job Type	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
5	0.18	0.18	0.18	0.18	0.19	0.18	0.49	0.18
6	0.14	0.12	0.14	0.41	0.12	0.31	0.31	0.25
7	0.14	0.13	0.12	0.50	0.15	0.49	0.46	0.42

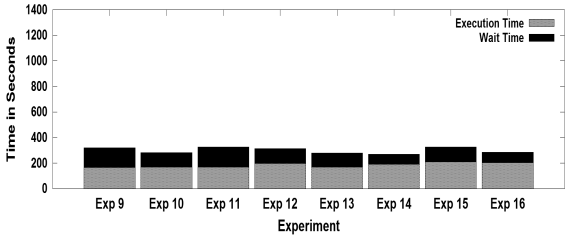
Makespan. Makespan is similar for both queues at 1230 seconds for nearly every experiment. The difference in makespan between the queues is less than 3 seconds across all experiments (within 0.2%, except experiment 2, which had a 1.5% smaller makespan at 1200 seconds). Makespan for the system is determined by the longest jobs.

6.2 Differential Capacity Queues

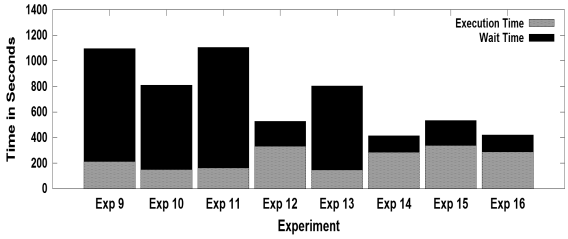
The next set of experiments had the relative queue capacity changed to 70-30%, so that the smaller queue is not squeezed for capacity but the larger queue still dominates resource allocation. The *minUserLimitPCT* setting reduces waiting time and improves response ratio as in the previous scenario for the high-capacity queue in most cases as shown in Figure 6. Approximate values of response ratios can be inferred from the diagram. Job Types 1, 2 and 3 did not show elapsed time reduction. Job Type 4 had an increased response ratio for experiment 9, but other experiments showed slight improvements. For the short jobs, there was significant improvement in response ratio for all experiments. Unfortunately, the improved response ratio in the high-capacity queue was occasionally accompanied by a significant increase in elapsed time for the shorter jobs, most notably experiment 15.

In the low-capacity queue, response ratios are largely unaffected for the long jobs, except for experiments 10 and 13, where they are over 10, as compared with around 6.7 for the equal capacity case. This is when *userLimitFactor* was set to 2. For the short job types, most response ratios increased, some substantially. In particular, Job type 5 had an increase in response ratio from 15 to 26 from experiment 2 to experiment 10. Figure 7 shows the execution/waiting times for selected Job Types for the low capacity queue.

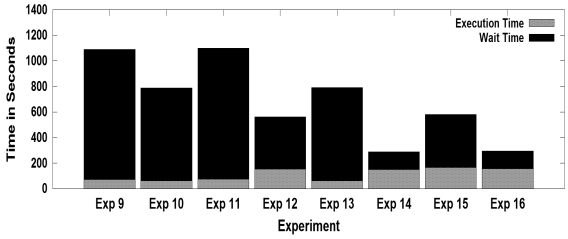
In experiments 10, 13, 14 and 16, where *maxCapacity* was set to *Capacity*, neither queue interfered with the other. Elapsed times were lower for the high-capacity queue than the low-capacity queue.



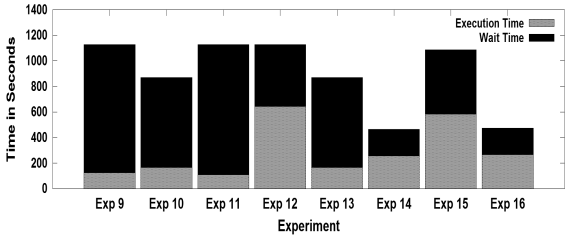
(a) Job Type 2



(b) Job Type 4

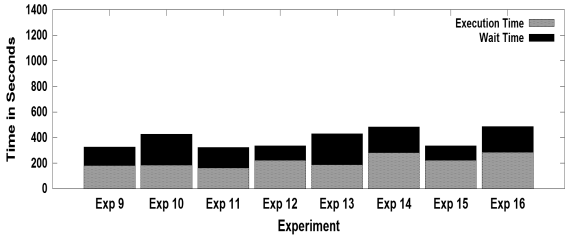


(c) Job Type 5

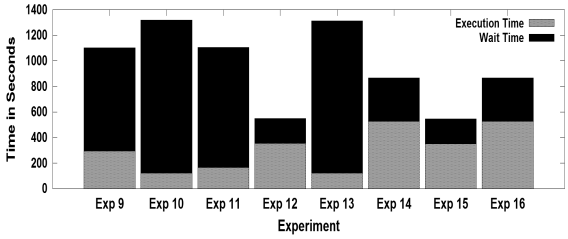


(d) Job Type 7

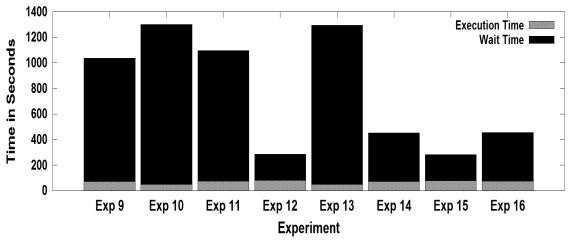
Figure 6: Execution And Wait Times For Selected Job Types: Differential Capacity (70% Queue)



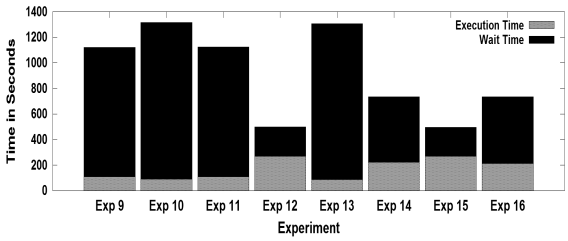
(a) Job Type 2



(b) Job Type 4



(c) Job Type 5



(d) Job Type 7

Figure 7: Execution And Wait Times For Selected Job Types: Differential Capacity (30% Queue)

The magnitude of the differences changed with the experiment and job type. In particular, experiments 14 and 16 had decreased execution times in both queues, but the high-capacity queue had decreased waiting times and the low-capacity queue had increased waiting times.

In all other experiments, there was queue interference. For Job Type 7, both execution and waiting time increase in experiments 12 and 15 in the high capacity queue, when compared with the corresponding experiments in Figure 5. In the low-capacity queue, execution time and wait time decrease. With *minUserLimitPCT* used, more users executed tasks in parallel, sometimes increasing elapsed times of the high-capacity queue to longer than the low-capacity queue.

The coefficient of variation of execution time for long jobs was less than 0.1. Data locality and disk contention (on either local or remote disk) again contribute to higher variation for short job types. Makespan is similar for both queues when *maxCapacity* is not set; the low-capacity queue can get additional capacity from the high-capacity queue. When the *maxCapacity* limit is used, the high-capacity queue makespan decreases by an average of 200 seconds. The low-capacity queue makespan increases by between 160 and 260 seconds.

6.3 Interleaved Jobs

This section examines performance in a controlled, but more realistic scenario in a Hadoop cluster: an interleaving job submission pattern, where 4 short jobs are followed by 1 long job and so on. Again, the most important parameter is *minUserLimitPCT*. It improved the response ratio for all job types. A job arriving earlier under default settings gets minimal improvement in response ratio as its waiting time does not change much. Table 8 shows the response ratios. For a job arriving later, the huge improvement in waiting time due to *minUserLimitPCT* improves the response ratio significantly (Experiments 20, 22, 23, and 24). Unfortunately, it does not affect makespan. In a system running over the longer-term, more shorter jobs could be executed and those arriving late would not suffer from long waiting times; thus, the appearance of a fair treatment of jobs as suggested by the analysis in the single queue case. The fact that the response ratio does not increase for other job types indicates that resources are being used more evenly.

Table 8: Response Ratio: Interleaved Scenario

	Experiment							
Job Type	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
1	1.18	1.18	1.18	1.02	1.18	1.02	1.01	1.02
2	1.90	1.94	1.93	1.46	1.91	1.46	1.47	1.47
3	1.35	1.51	1.51	1.15	1.50	1.15	1.15	1.15
4	8.46	8.70	8.91	1.79	8.48	1.79	1.77	1.80
5	2.03	2.02	2.03	1.41	2.02	1.43	1.43	1.42
6	3.19	4.16	4.24	2.47	4.10	2.53	2.53	2.48
7	4.20	6.91	6.85	1.90	6.70	1.89	1.90	1.90

6.4 Separate Job Queues

In Queues 1 and 2 are short job queues and receive Job Types 5, 6, and 7. Queues 3 and 4 are the long job queues and receive Job Types 1 through 4.

Setting *minUserLimitPCT* different than the default has improved the response ratio in all the conducted experiments so far. Separate queue submission reveals something different. Instead of *minUserLimitPCT*, *maxCapacity* improves the response ratio for short jobs. For long jobs, *minUserLimitPCT* improves the response ratio. Table 9 shows the response ratio for different Job Types.

Table 9: Response Ratio: Separate Queues

	Experiment							
Job Type	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
1	1.14	1.23	1.25	1.01	1.25	1.01	1.02	1.02
2	1.76	2.21	1.94	1.39	2.18	1.56	1.38	1.54
3	1.19	1.59	1.51	1.12	1.58	1.16	1.12	1.15
4	2.46	9.36	6.51	1.55	9.06	1.73	1.56	1.72
5	13.1	1.20	5.85	4.84	1.18	1.06	5.00	1.08
6	14.1	1.65	7.03	5.22	1.48	1.20	5.18	1.20
7	9.04	1.53	4.18	4.88	1.45	1.27	4.87	1.25

With separate queues, short jobs should finish faster with lower response ratios. In experiments 25, 27, 28 and 31, where *maxCapacity* limit is not imposed, however, the response ratio for short jobs (Job types 5, 6, and 7) is high due to the long job queues. These jobs take slots from the short job queues, leading to huge waiting times for short jobs. Imposition of limits reverses the trend. The *maxCapacity* limit puts an upper bound on the capacity a queue can use; lower waiting time and lower response ratios are the result.

Only jobs from a queue should determine the makespan, but the short job queue was affected by

the long job queue when *maxCapacity* was not set. Makespan was between 380 and 430 for Queues 1 and 2 with *maxCapacity* set, but within 5% of the long job queue in the other cases and even 3% longer in experiment 31. Limiting *maxCapacity* prevents long job queues from stealing slots from short job queues. It also decreases waiting times for short jobs and reduces the short queue makespan.

Figure 8(a) shows that the long job queue interferes with the short job queue, but in Figure 8(b), isolation is achieved along with good elapsed time for the shorter jobs. While Job Types 2, 3, and 4 experience longer wait times, only Job Type 4 has a drastic change in response ratio. This job type is relatively rare (2% of job mix), and therefore most jobs are positively affected by this configuration and the elapsed time of Job Type 4 is unaffected.

6.5 Analysis

We cannot make significant conclusions about the behaviour and timing of workload scenarios in general, due to the limited nature of the workloads and size of the simulated system. Variability in execution time caused by factors not modelled by the simulator prevent accurate predictions (e.g. same node effect), but our results are consistent within the restricted mode of operation.

With a long sequence of long jobs followed by small jobs, *minUserLimitPCT* plays the most important role in determining response ratio for specific job types and arrival times. Changing *minUserLimitPCT* also increases competition for resources and increased execution times for the jobs, with reduction in elapsed time for some job types.

For queues with unequal capacity, *maxCapacity* can be used for real performance gains. Otherwise, long jobs in the low-capacity queue interfere more often with high-capacity queue. For interleaved jobs, *minUserLimitPCT* improves the response ratio for all jobs, especially later arrivals.

For separate queues, two different behaviours were observed. With no *maxCapacity* limits imposed, long jobs affect the short job queue by stealing slots, increasing makespan and response ratio. With *maxCapacity* limits imposed, the short jobs are isolated from long jobs. Their response time is small and makespan is not affected by the long jobs. Setting *minUserLimitPCT* to guarantee sharing when there are more jobs than instantaneous cluster capacity improves the response ratio, but less than setting *maxCapacity*.

If queue capacity is to be divided among short and long jobs, long jobs must be given more capacity to obtain more slots over time. More accurate job size estimation [15] may have an additional increase in performance if jobs can be moved between queues. The examination of job size in terms of map and reduce tasks distinguishes our analysis from that of the general HPC community [11].

7 Conclusions/Future Work

There is limited knowledge about the effects of parameter settings on the performance of the Capacity Scheduler. In this paper, we integrated the Capacity Scheduler into the open-source Hadoop Simulator MRPerf and validated its performance against a real test cluster. This exposed some inaccuracies in the simulation model, namely stragglers and same-reduce-node effects, but was sufficiently accurate. Finally, a simulation study was performed to understand and quantify the impact of Capacity Scheduler parameter settings on a sample workload under different job submission patterns. From this limited set of experiments, the performance improvement obtained by treating long and short jobs differentially suggests new schedulers that could make use of previous work in cluster scheduling [7].

MRPerf can be extended to have multiple-disk support on a single node, memory configuration and speculative execution. Some sub phases are not currently modelled. Such sub-phase omission leads to prediction errors in the simulator as certain real system behaviours cannot be reproduced. As well, straggler support and stochastic heartbeat arrivals can be put into MRPerf.

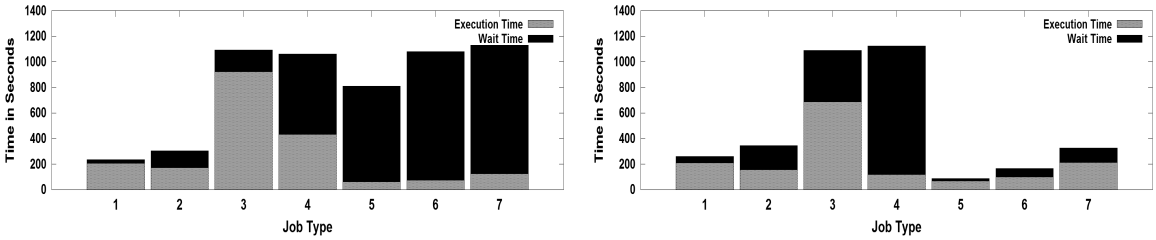
References

[1] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using Mantri. In *OSDI*, pages 1–16, Vancouver, Canada, Oct. 2010.

[2] S. Babu. Towards automatic optimization of MapReduce programs. In *IEEE SoCC*, pages 137–142, Indianapolis, IN, June 2010.

[3] J. Chauhan. Simulation and Performance Evaluation of Hadoop Capacity Scheduler. Masters thesis, University of Saskatchewan, 2013.

[4] J. Chauhan, D. Makaroff, and W. Grassmann. The Impact of Capacity Scheduler Configuration set-



(a) Experiment 25

(b) Experiment 26

Figure 8: Separate Queue Scenario wait/execution times

tings on MapReduce Jobs. In *IEEE CGC*, pages 667–674, Xiangtan, China, Nov. 2012.

[5] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads. *VLDB Endowment*, 5(12):1802–1813, Aug. 2012.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, Jan. 2008.

[7] N. Fallenbeck, H.-J. Picht, M. Smith, and B. Freisleben. Xen and the Art of Cluster Scheduling. In *VTDC*, pages 237–244, Tampa, FL, Nov. 2006.

[8] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu. MRSim: A discrete event based MapReduce simulator. In *Fuzzy Systems and Knowledge Discovery*, pages 2993–2997, Yantai, China, Aug. 2010.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In *CIDR*, pages 261–272, Asilomar, CA, Jan. 2011.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SOSP*, pages 261–276, Big Sky, MT, Oct. 2009.

[11] D. B. Jackson, Q. Snell, and M. J. Clement. Core Algorithms of the Maui Scheduler. In *Revised Papers from JSSPP*, pages 87–102, Cambridge, MA, June 2001.

[12] D. Jiang, B. C. Ooi, L. Shi, and S. Wu. The performance of MapReduce: an in-depth study. *VLDB Endowment*, 3(1-2):472–483, Sept. 2010.

[13] Y. Liu, M. Li, N. K. Alham, and S. Hammoud. HSim: A MapReduce Simulator in Enabling Cloud Computing. *Future Gener. Comput. Syst.*, 29(1):300–308, Jan. 2013.

[14] A. Murthy. Mumak: Map-Reduce Simulator. MAPREDUCE-728, 2009.

[15] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, and P. Michiardi. HFSP: Size-based scheduling for Hadoop. In *2013 IEEE Intl. Conf. on Big Data*, pages 51–59, Silicon Valley, CA, Oct. 2013.

[16] F. Teng, L. Yu, and F. Magoulaas. SimMapReduce: A simulator for modeling MapReduce framework. In *FTRA Mobile and Ubiquitous Engineering*, pages 277–282, Crete, Greece, June 2011.

[17] V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *IEEE SOCC*, pages 5:1–5:16, Santa Clara, CA, Oct. 2013.

[18] A. Verma, L. Cherkasova, and R. H. Campbell. Play it Again, SimMR! In *IEEE CLUSTER*, pages 253–261, Austin, TX, Sept. 2011.

[19] A. Verma, L. Cherkasova, and R. H. Campbell. Two Sides of a Coin: Optimizing the Schedule of MapReduce Jobs to Minimize Their Makespan and Improve Cluster Performance. In *MASCOTS*, pages 11–18, Washington, DC, Aug. 2012.

[20] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in MapReduce setups. In *MASCOTS*, pages 1 –11, London, UK, Sept. 2009.

[21] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. Using realistic simulation for performance analysis of MapReduce setups. In *LSAP Workshop*, pages 19–26, Garching, Germany, June 2009.

[22] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EUROSYS*, pages 265–278, Paris, France, Apr. 2010.