Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Trace Driven Analytic Modeling for Evaluating Schedulers for Clusters of Computers

Shouvik Bardhan
sbardhan@masonlive.gmu.edu

Daniel A. Menascé
menasce@gmu.edu

Technical Report GMU-CS-TR-2014-02

Abstract

Large enterprises use clusters of computers with varying computing power to process workloads that are heterogeneous in terms of the type of jobs and the nature of their arrival processes. The scheduling of jobs from a workload has a significant impact on their execution times. This report presents a trace-driven analytic model (TDAM) method that can be used to assess the impact of different schedulers on job execution times. The TDAM approach uses an implementation of the scheduler to schedule jobs that are fed into analytic models of the computers in the cluster. These analytic models use closed queuing network methods to estimate congestion at the various nodes of the cluster. The report demonstrates the usefulness of the TDAM method by showing how four different types of schedulers affect the execution times of jobs derived from well-known benchmarks. The report also demonstrates how the method can be applied to heterogeneous computer clusters such as the ones used to run MapReduce jobs.

1 Introduction

Many enterprises today run applications on a cluster of heterogeneous machines. Apache Hadoop [24] and associated software stacks (e.g., HBASE, Accumulo, Flume and many others [28, 29, 30]) are examples of such software platforms and products. Large Internet software organizations like Amazon, Yahoo and Facebook run thousands of Hadoop jobs on a routine basis on clusters comprising of thousands of server nodes. These jobs have varied completion time requirements since some are ad hoc and quick query jobs, some are medium size data mining jobs, and some are very large (in terms of resource requirements and completion times) analytical processing jobs.

Hadoop developers have developed several different schedulers over the years to schedule MapReduce jobs on Hadoop platforms to suit their particular organizational needs. These schedulers need extensive testing and verification for correctness. Most job schedulers in the Hadoop ecosystem have complex XML configuration files to set up queues and quotas. But, more importantly, testing and validation is needed to check if the schedulers are appropriate for the intended workload mix.

The efficacy of a job scheduler can be assessed in many different ways: (1) *Experimentation:* Select a representative mix of real jobs, setup a real cluster, run the jobs using a given scheduler and measure the job's completion time. This method is very onerous mainly because obtaining a suitable free cluster for experimentation is often very difficult. (2) *Simulation:* Simulate a representative mix of real jobs running through a simulated scheduler and using simulated servers. This method is complex because not only the scheduler but the processing and I/O resources of the servers have to be simulated in software. (3) *Analytic modeling:* Develop analytic models of the scheduler, servers and their queues using the proper arrival distributions. This is not trivial because modeling the behavior of even moderately complex scheduling disciplines and their interaction with the server models for heterogeneous workloads may not be mathematically tractable.

This report presents a novel method to assess schedulers for server clusters. This method, called TDAM (Trace Driven Analytic Model), relies on the implementation of the scheduler under evaluation and on analytic closed queuing network (QN) models to assess resource contention at the servers. The implemented scheduler takes as input a synthetic trace of jobs of various types and schedules them to the "servers," which are modeled by the server QN models. By using an implementation of the scheduler on a simulated or synthetic job trace we avoid the complexity of modeling the behavior of scheduling policies. The analytic QN models capture the congestion of CPU and I/O resources at the various servers. These QN models are solved using the Epochs algorithm (see [14]), which uses Mean Value Analysis (MVA) [15, 17] for finite time intervals, called epochs, using an operational analysis formulation [5]. We applied the TDAM method to assess four cluster scheduling policies under different workload types.

The rest of this report is organized as follows. Section 2 discusses the need for assessing different job schedulers. Section 3 describes the building blocks of the TDAM method.

The following section presents and discusses the results of using TDAM on different types of workloads and scheduling policies. Section 5 discusses related work and lastly section 6 presents some concluding remarks and future work.

2 The Need for Robust Job Scheduler Assessment

Most known schedulers, including those created for scheduling Hadoop MapReduce jobs, do not provide maximum completion time guarantees. A big challenge in today's Hadoop and similar platform clusters is to manage the allocation of resources to different jobs so that a service level objective (SLO) can be advertised and met. This requires (1) a quick, efficient, and realistic way to assess the schedulers before they go into production; and (2) an efficient method to estimate resource congestion at the servers.

Our solution to this challenge is the TDAM method, which allows any type of job scheduler to be evaluated on any job trace. It is conceivable that even for a single enterprise, depending on the time of day and workload mix, one scheduling scheme outperforms another. Dynamically choosing schedulers based on the workload mix and overall resource utilization (which requires collecting and analyzing data from every node and other cluster characteristics like network bandwidth and topology) is the subject of self-configuring autonomic job scheduling [16]. We do not discuss autonomic aspects of dynamic scheduling in any detail here.

One of the primary goals of our research is to show how to apply the TDAM method to enterprises using the Hadoop platform to run MapReduce jobs. Originally, Hadoop was designed to run large batch jobs infrequently. The out-of-the-box FIFO scheduler packaged with Hadoop was sufficient for that purpose. There was really no need to worry about resource utilization to complete jobs in a timely manner. However, other schedulers were developed as the number of jobs increased many fold and organizations started to use Hadoop not only for batch analytics but also for small ad-hoc queries. The *Capacity Scheduler* [31] was developed for more efficient cluster sharing. The *Fair Scheduler* [32] was introduced to maintain fairness between different cluster users.

A few more schedulers have been developed since then [12]. These schedulers did not provide any method to assess their effects on a job trace, but more importantly there was no way to know how they would behave under moderate or heavy workloads without actually running real jobs. The next section describe the Job Scheduler Evaluator (JSE), which is based on the TDAM method and affords efficient assessment of job completion times under a variety of scheduling disciplines.

3 A Job Scheduler Evaluator Based on TDAM

Figure 1 shows a component-level diagram of the JSE. The workload produced by the *Workload Generator* is a stream of jobs with their types and the list of tasks that compose the jobs. For example, MapReduce jobs are composed of map tasks and reduce tasks. The workload trace is parsed by the *Job List Manager* and by the *Task List Manager* components of JSE.

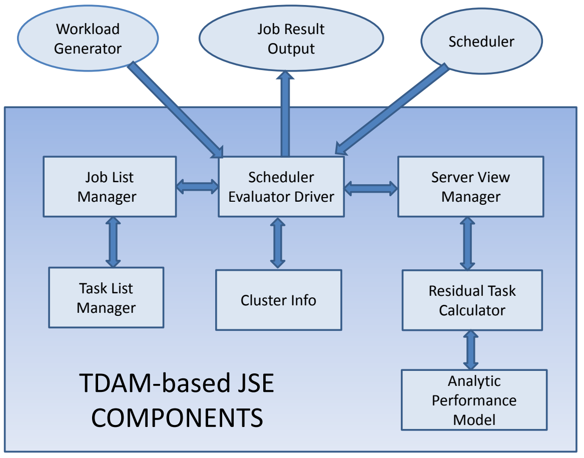


Figure 1: Components of the TDAM-based JSE

The *Scheduler Evaluator Driver* uses the Epochs algorithm [14] and the scheduling policy provided as input to generate a stream of tasks to individual server views managed by the *Server View Manager*. The Epochs algorithm estimates the execution time of jobs in a job/task stream. Time is divided into intervals of finite duration called *epochs*. The first epoch starts when the first job arrives and the end of an epoch is characterized by either the arrival of a new job or the completion of a job. The Epochs algorithm predicts the start time of the next epoch on any cluster node by being aware of the residual times of each task as tasks arrive and leave the nodes. The *Residual Task Calculator* component is at the heart of the JSE and computes the residual service time of each task in execution using the Epochs algorithm [14]. This algorithm relies on the operational counterpart for finite time intervals of the Mean Value Analysis (MVA) [17] equations for closed QN models when flow balance, one-step behavior, and homogeneous service times are met [4, 5]. The Epochs algorithm was validated experimentally using a micro-benchmark and real jobs from a Unix benchmark [14].

The *Analytic Performance Model* block implements the Approximate Mean Value Analysis (AMVA) algorithm [15]. In essence, the Epochs algorithm along with the AMVA calculation engine allows us to predict the completion time of individual tasks when a stream of jobs arrives to be executed on a cluster node.

The idea behind JSE involves scheduling a stream of jobs that need to be executed virtually (since these jobs are not really executing on servers) and thus have to be scheduled to run on different nodes of a cluster. As jobs arrive, the *Scheduler Evaluator Driver* uses the scheduling policy to decide which cluster node should receive a task. So, each server in the cluster sees a stream of arriving tasks (which is a sub-stream of the original arriving stream). We then apply the Epochs algorithm [14] to each individual server using its sub-stream.

The *Scheduler Evaluator Driver*, as implemented here, does not have a callback mechanism from the task being "executed" on the server into the scheduler driver when the task completes. This means that when a task finishes, it cannot let the scheduler know that it is done. This was taken care of in our implementation by having the scheduler be aware of the next epoch information for each server, i.e., the time in the future when a task will finish. Not having the servers have their own thread of control simplifies the driver code.

4 Using the JSE

We used the following four non-preemptive scheduling policies to illustrate the used of the JSE.

- Round Robin (RR): Chooses the servers in the cluster in a round robin fashion. This scheduling scheme is oblivious to the utilization of any server resource (either CPU or disk).
- Least Response Time (LRT): Selects the server on which the incoming job is predicted to have the least response time. Since the scheduler has the exact states of all the jobs running on all the servers, it can calculate the response time of the incoming job if it were added to any node in the cluster.
- Least Maximum Utilization First (LMUF): Selects the server with the minimum utilization for the resource with the highest utilization is the one that receives an incoming job. The utilization of the resources at each server is calculated as a snapshot at the time the new job arrives to be scheduled.
- Least Maximum Utilization First-Threshold (LMUF-T): Similar to LMUF except that a job is not sent to a server if the utilization for the resource with the highest utilization at that server exceeds a certain threshold. In that case, the job is queued at the scheduler. When a job completes at any machine, the scheduler attempts to send the queued job again to one of the servers. The goal of LMUF-T is to bound the contention at each node by having jobs wait at the scheduler. It may be more advantageous to wait a bit at the scheduler and then be assigned to a less loaded machine.

For the workload used to compare the schedulers, we developed a program that creates job streams by randomly selecting jobs from one of the three benchmarks: Bonnie++ [25], Nbench [27], and Dbench [26]. Inter-arrival times were assumed to be exponentially distributed, even though this assumption is not required by TDAM. Any arbitrary arrival process that satisfies the homogeneous arrival assumption can be used [4, 5]. The job stream files thus created, along with the scheduling scheme and the number of servers in the cluster are the main input parameters to the JSE.

We consider both single-task jobs and multi-task jobs. In a multi-task job, the various tasks of a job run on the same or different machines of a cluster and the job is deemed to have completed only when all its tasks have completed. MapReduce jobs are examples of multi-task jobs [24].

First, we consider the effect of different scheduling schemes on the makespan of various single-task jobs. Then, we consider the impact of the CPU utilization threshold in the LMUF-T scheduling policy. Lastly, we discuss the results of running the JSE with multi-task jobs on a heterogeneous cluster.

4.1 Effect of the Scheduling Policy on the Makespan

This section considers how the scheduling policy affects the makespan, i.e., the time needed to execute all jobs in a job stream. Table 1 shows the characteristics of the jobs used in the evaluation carried out in this section. For single-task jobs, we made very minor modifications to three benchmark programs (Bonnie++ [25], Nbench [27] and Dbench [26]) and measured their CPU and disk service demands (see Table 1). Changing the input parameters to these benchmark programs allowed us to obtain two sets of service demand values (e.g., for Bonnie++, the two sets of values for CPU and disk demands are [8.2 sec, 9.8sec] and [16.4 sec, 19.6 sec]). The job inter-arrival times are exponentially distributed with averages of 3 sec and 6 sec. Thus, as shown in Table 1, we obtained four different workloads by combining two service demand sets and two average inter-arrival time values.

For each workload in Table 1, we created 10 job streams by randomly selecting, with equal probability, the type of job at each arrival instant. Figures 2(a)-2(d) depict the makespan for all 10 streams for each of the four workloads and for each of the four scheduling disciplines. The utilization threshold used in LMUF-T is 70% for all the graphs. Note that the y-axis does not start at zero so that the differences between the schedulers become easier to visualize. However, as discussed

Workload	1	2	3	4
Job ↓	Inter-arrival time:6 sec		Inter-arrival time:3 sec	
Bonnie++	(8.2, 9.8)	(16.4, 19.6)	(8.2, 9.8)	(16.4, 19.6)
Nbench	(25, 0)	(50, 0)	(25, 0)	(50, 0)
Dbench	(5.5, 4.5)	(11, 9)	(5.5, 4.5)	(11, 9)

Table 1: (CPU, Disk) service demands (in sec) for benchmark jobs Bonnie++, Nbench, and Dbench, and two values of the workload intensity

below, statistical tests were used to analyze the data. The following conclusions can be drawn from these figures. First, LMFU-T provides the worst (i.e., the highest) makespan in all cases. This is a consequence of LMFU-T not sending jobs to a server when the utilization of its bottleneck device exceeds the threshold T. While LMFU-T is inferior than the other policies for a 70% threshold, there may be good reasons to use this policy: (1) the energy consumption of a server increases with its utilization, and (2) running servers at high utilizations may reduce their reliability and their lifetimes. Second, LRT and LMUF provide similar makespans at the 95% confidence level for most job streams and workloads. The reason is that there is a strong correlation between the response time at a server and the utilization of its bottleneck device. Finally, RR is worse than LRT and LMUF in many cases, wspecially for the ones with higher service demands and/or higher workload intensity values. This is an expected result because RR is oblivious to the workload or load on the server.

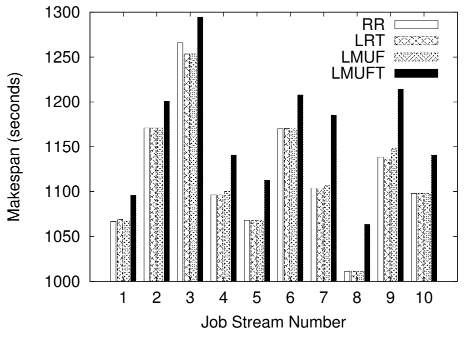
We performed one-factor ANOVA [13] at the 95% confidence level for the data shown in the graphs of Figs. 2(a)-2(d). LMFU-T was shown to be significantly different at the 95% confidence level then the three other scheduling disciplines. We applied the Tukey-Kramer [13] procedure to the three other scheduling disciplines and found significant differences among them in some of the tested workloads.

4.2 Effect of Scheduler Throttling Based on Device Utilization

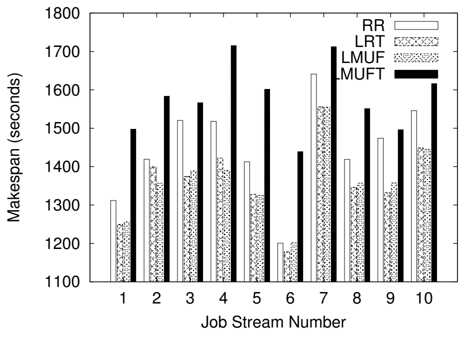
We now discuss the effect of holding schedulable tasks at the scheduler queue when a particular device utilization is higher than a specified threshold as done in LMFU-T. In that case, the total execution time of a job consists of two components: wait time at the scheduler and time spent at the server, which includes time spent using resources and time waiting to use resources at the server.

The graphs in Figs. 3(a)-3(f) show timing data (wait time at the scheduler, server time, and total time) for workloads 1 and 3. These workloads contain a mix of the three types of jobs in Table 1. However, each of the graphs only shows average values for one specific job within the multi-job workload. Figures 3(a)-3(c) show the total job execution time, server time, and wait time versus the CPU utilization threshold for workload 1. As the threshold approaches 100%, the wait time goes to zero as expected because no jobs will be queued at the scheduler. However, as can be seen in the figures, the server time increases due to increased congestion at the servers. For each type of program (meaning different service demands), the range of threshold values that provides the best total execution time is not the same. For Bonnie++, this range is [0.6, 0.9]; for Nbench it is [0, 0.4] and for Dbench the lowest execution time is reached for a CPU utilization threshold equal to 100%.

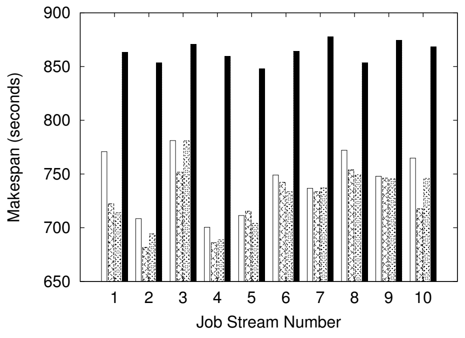
Figures 3(d)-3(f) show the total execution time, wait time, and server time, for the same jobs as in Figs. 3(a)-3(c) (i.e., same service demands) but with an average arrival rate twice as big. This corresponds to workload 3 in Table 1. In the case



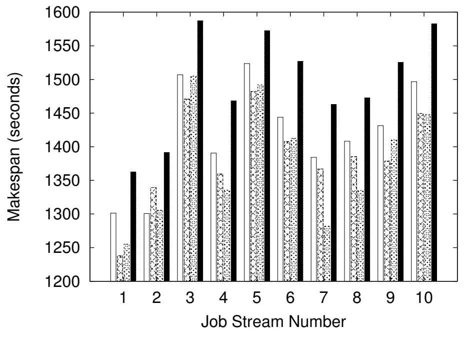
(a) Makespan vs. job stream number for workload 1



(b) Makespan vs. job stream number for workload 2



(c) Makespan vs. job stream number for workload 3

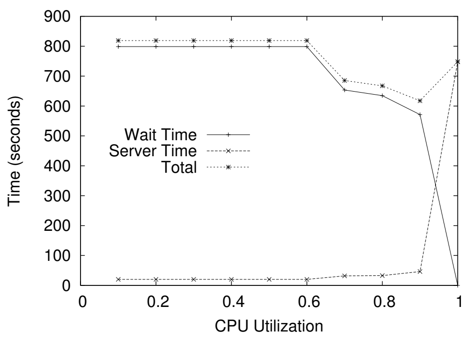


(d) Makespan vs. job stream number for workload 4

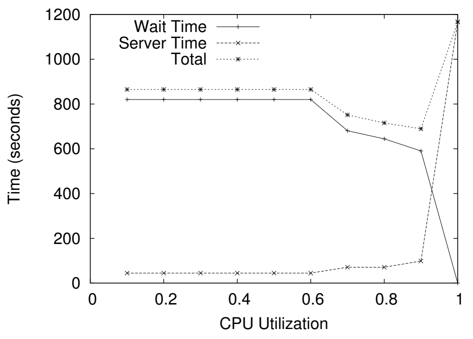
Figure 2: Makespan for single-task jobs.

of these graphs, there is a marked difference in behavior. As we can see from the figures, because the arrival rate has doubled, server congestion increases significantly when the effect of scheduler throttling is reduced by using a high utilization threshold. For example, the best utilization threshold range is [0.1, 0.4] for Bonnie++, Nbench, and Dbench.

Figures 4(a) and 4(b) depict similar graphs for MapReduce [24] jobs with different CPU and disk service demands (small and large). These are multi-task jobs typical of MapReduce applications. Table 2 shows the number of tasks per job and their service demands. This experiment assumed 12 servers and was run with the assumption that all the tasks of all the jobs are available for scheduling immediately, a typical scenario for map tasks in MapReduce. The graphs of Figs. 4(a) and 4(b) show that both types of jobs exhibit the lowest execution time for a CPU utilization threshold of 90%. These graphs illustrate very clearly the tradeoffs between spending time waiting at the scheduler because of the utilization threshold versus spending more time at the server due to a heavily congested server.



(a) Timing vs. CPU utilization threshold for small MapReduce jobs



(b) Timing vs. CPU utilization threshold for large MapReduce jobs

Figure 4: Average execution time, waiting time, server time vs. CPU utilization threshold for MapReduce jobs.

Figures 5(a) and 5(c) show the CPU utilization for a typical server in the cluster using RR scheduling for workloads 1 and 2, respectively. Figures 5(b) and 5(d) show similar data for LMUF-T scheduling with a 70% threshold. Remember that according to Table 1, the workload intensity is the same for workloads 1 and 2. But, service demands for workload 2 are twice as high as those for workload 1. Note that in the RR case, workload 1 (Fig. 5(a)) shows a moderate average CPU utilization with a few peaks reaching 100%. But, for workload 2 (Fig. 5(c)) the CPU utilization quickly reaches 100% and stays there. Let us now contrast the RR case with the LMUF-T case (Figs. 5(b) and 5(d)). Here we see that for both workloads, the average CPU utilization stays at a lower level and never stays fixed at 100% (an undesirable situation) as in Fig. 5(c). This is because of the 70% threshold for CPU utilization used by LMUF-T in this case.

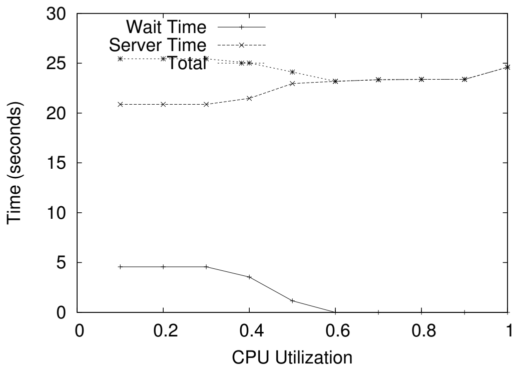
4.3 Scheduling Multi-task Jobs on a Heterogeneous Cluster

This subsection discusses the results obtained by executing various MapReduce jobs consisting of map tasks only on a cluster with heterogeneous servers. A job is considered finished only when all its tasks finish. The workloads used in this subsection are described in Table 2 and they differ in terms of their CPU and disk service demands. We assume that all the jobs arrive at the same time so their tasks need to be all scheduled at time zero as is customary for MapReduce jobs. Based on the number of tasks per job and the number of jobs in Table 2, there are 50 small job tasks, 50 medium job tasks, and 50 large job tasks. We assume a cluster with 12 nodes, six of which are half as fast as the other six. The CPU and disk service demands in Table 2 correspond to the faster machines. The corresponding values for the slow machines are twice the values in that table.

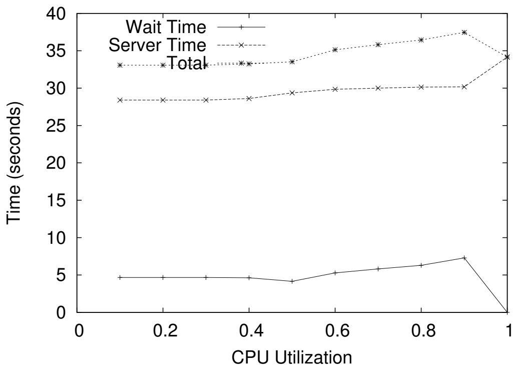
Most of the time, each task in a MapReduce job is given exclusive use of a CPU core so that at most one task is scheduled to run at a given core at any given time. We explore here the impact of using LMFU-T using three different CPU utilization threshold values:

- 0 %: at most one task per core is allowed. This is similar to the conventional Hadoop scheduling approach described above.
- 70%: a task is not scheduled into a core if its utilization exceeds 70%, and
- 100%: tasks do not have to wait in the scheduler queue and can be assigned to any core. This is equivalent to LMFU. Note that the 70% and 100% cases correspond to CPU oversubscribing, not typical in Hadoop environments.

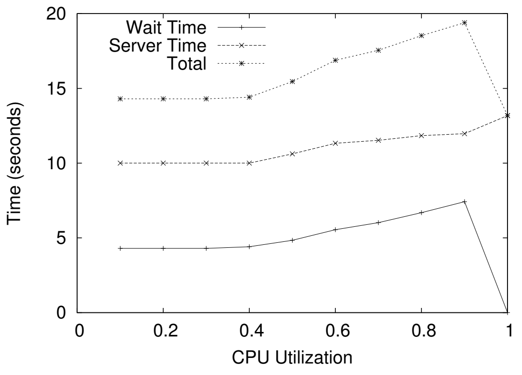
Tables 3-5 show data for experiments using the three utilization thresholds, respectively. Each table shows the average waiting time at the scheduler queue, the average time spent at the server, and the average total time. Additionally, the overall makespan is shown. Four scenarios are shown in each table: (1) Any job can be scheduled on any machine, (2)



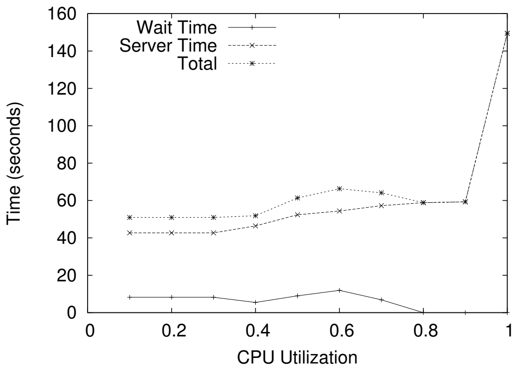
(a) Timing vs. CPU utilization threshold for Bonnie++ and workload 1



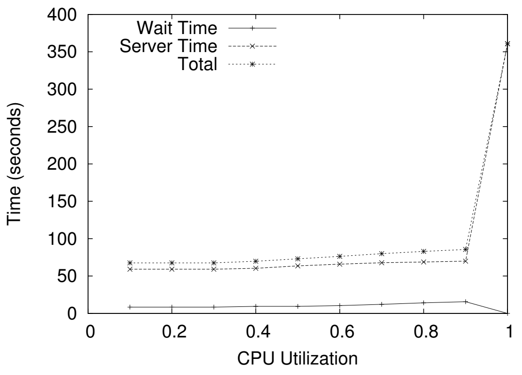
(b) Timing vs. CPU utilization threshold for Nbench and workload 1



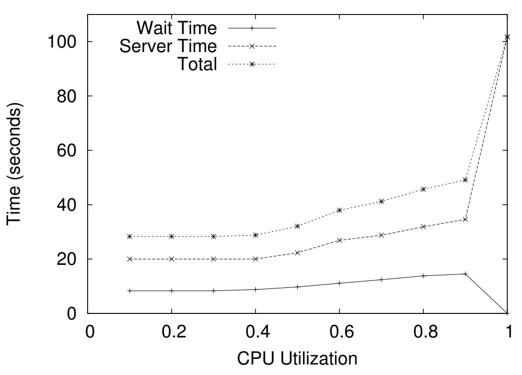
(c) Timing vs. CPU utilization threshold for Dbench and workload 1



(d) Timing vs. CPU utilization threshold for Bonnie++ and workload 3



(e) Timing vs. CPU utilization threshold for Nbench and workload 3



(f) Timing vs. CPU utilization threshold for Dbench and workload 3

Figure 3: Average execution time, wait time, and server time vs. CPU utilization threshold.

Job Type	No. Tasks	No. Jobs	CPU Demand (sec)	Disk Demand (sec)
Small	5	10	5	1
Medium Jobs	25	2	10	5
Large	25	2	30	15

Table 2: Multi-task job characteristics.

Large jobs are only scheduled on slow machines and the other jobs are scheduled on any machine, (3) Small and medium jobs are scheduled on the slow machines and large jobs on the fast machines only, and (4) Small jobs are scheduled to the slow machines and medium and large jobs are scheduled into any machine.

The following observations can be derived from these tables. First, for a given CPU utilization threshold value, the best makespan values are obtained either when any job can be scheduled to any machine (case 1) or when fast nodes are used exclusively by large jobs (case 3). Second, the tables show the clear impact of the threshold on server congestion. For example, Table 3, which is similar to the Hadoop approach of granting exclusive access to CPU cores to each task, shows very small server contention and large waiting times at the scheduler queue. For example, large jobs spend between 73% and 79% of their total time at the scheduler queue in any of the four scenarios. When the utilization threshold is 70% (Table 4), there is a slight increase in server time due to added server congestion and a decrease in waiting time at the scheduler queue. As a consequence, the total times are lower than when the utilization threshold is zero (Table 3). Third, the case in which there is no queuing at the scheduler (Table 5) favors small jobs over the other threshold values for all cases except for case 3. This is expected because in case 3 small and medium jobs can only use the slower machines and in the case of Table 5 there is significant contention at these nodes because there is no admission control at the scheduler. On the other hand, large jobs have a much higher total time under the 100% threshold. This is due to the very high contention at the server nodes when compared with the other threshold values. In fact the slowdown (i.e., total time divided by total service demand) for large jobs is 6.3, 12.2, 5.7, and 8, for cases 1-4, respectively for the 100% utilization threshold and the corresponding values for the 70% threshold are 4.3, 7.9, 4.6, and 4.4.

Even though we have shown results for three different utilization threshold values and for four scenarios for allocating tasks to the different types of nodes, the approach presented in this report can easily be applied to a wide variety of "what-if" scenarios. The reason is that the scheduler is implemented and the servers are modeled using closed queuing networks. The integration between the scheduler implementation and the closed QN models is done through the Epochs [14] algorithm. A significant advantage of the TDAM approach is that it allows for any scheduling discipline to be assessed for any size and type of cluster given that the servers in the cluster are modeled analytically.

5 Related Work

There is a significant body of work on scheduling for single queues. Harchol-Balter brings an excellent survey of analytic scheduling results for M/G/1 queues in Part VII of her recent book [10]. In [10], Harchol-Balter also looks at the problem of immediate dispatching of arriving tasks to a server farm. She considers that each server in the server farm is modeled as a single queue, i.e., the CPU and disk resources of the server are not individually modeled as we do here. Also, the work in [10] does not consider the possibility of queuing at the scheduler, as is done in LMFU-T. Several papers on job scheduling for parallel environments appear in [7] and [8]. For example, the paper in [22] uses a simulation study to analyze failure-aware scheduling strategies in cluster scheduling.

During the last half a decade or so, performance analysis and modeling of MapReduce jobs has received significant attention and several different approaches have been presented [11, 18]. A representative group from HP Labs and collaborating researchers have published numerous papers on how to improve the resource allocation of MapReduce programs and have presented proposals on how to model the map, reduce, and shuffle phases of these jobs [19, 20, 23].

In [11], the authors discuss a query system to answer cluster sizing problems using a combination of job profiles and estimations. The authors in [18] built a cost function based on the complexity of the map and reduce tasks. The profile of these tasks is calculated on a testbed environment. We also measure job characteristics (i.e., service demands) on a testbed environment. However, we do not require that the testbed be sized as the intended production site. In fact, a single node is sufficient to measure service demands. The ARIA paper [19] provides a solid foundation on how to analyze the different phases of MapReduce jobs. The authors first create job profiles from which they can ascertain the optimum allocation of map and reduce slots. Consequently, they create an SLO scheduler that incorporates the aforementioned model. However, that work does not consider resource contention due to multiple tasks running on the same node, as done here. In [1], the authors discuss a tool called Tresa that helps system administrators predict execution times when consolidating workloads in a data center. The tool automates workload characterization and uses MVA to predict execution times. There is no mention to the effect of scheduling policies on job execution times.

However, none of the above referenced papers and no other studies, to the best of our knowledge, consider the effects of resource contention when predicting the effect of scheduling

Table 3: Timing for small, medium, large MapReduce jobs and CPU utilization threshold = 0

LMUF-T ($U_{\text{cpu}} \leq 0.0$)	Avg. Wait Time	Avg. Server Time	Avg. Total Time	Overall makespan
(1) Any job to any machine				
Small Jobs	148.0	8.2	156.2	391.0
Medium Jobs	150.2	19.0	169.2	
Large Jobs	152.4	55.8	208.2	
(2) Large jobs to slow machines, other jobs to any machine				
Small Jobs	270.7	6.1	276.8	675.0
Medium Jobs	270.9	15.2	286.1	
Large Jobs	281.8	75.0	356.8	
(3) Small/Medium jobs to slow machines, large jobs to fast machines				
Small Jobs	168.2	11.0	179.2	405.0
Medium Jobs	169.0	25.0	194.0	
Large Jobs	175.2	45.0	220.2	
(4) Small jobs to slow machines, medium/large jobs to any machine				
Small Jobs	159.2	11.0	170.2	413.0
Medium Jobs	160.3	17.2	177.5	
Large Jobs	161.7	56.4	218.1	

Table 4: Timing for small, medium, large MapReduce jobs and CPU utilization threshold = 0.7

LMUF-T ($U_{\text{cpu}} \leq 0.7$)	Avg. Wait Time	Avg. Server Time	Avg. Total Time	Overall makespan
(1) Any job to any machine				
Small Jobs	118.9	10.2	129.0	344.0
Medium Jobs	120.8	23.8	144.6	
Large Jobs	122.7	71.7	194.3	
(2) Large jobs to slow machines, other jobs to any machine				
Small Jobs	269.8	6.8	276.5	675.0
Medium Jobs	269.8	18.6	288.4	
Large Jobs	281.8	75.0	356.8	
(3) Small/Medium jobs to slow machines, large jobs to fast machines				
Small Jobs	136.9	11.0	149.9	342.9
Medium Jobs	141.0	25.0	166.0	
Large Jobs	141.0	63.6	204.7	
(4) Small jobs to slow machines, medium/large jobs to any machine				
Small Jobs	125.6	11.0	136.6	347.0
Medium Jobs	126.5	23.1	159.6	
Large Jobs	127.9	69.8	197.7	

Table 5: Timing for small, medium, large MapReduce jobs and CPU utilization threshold = 1.0

LMUF-T ($U_{\text{cpu}} \leq 1.0$)	Avg. Wait Time	Avg. Server Time	Avg. Total Time	Overall makespan
(1) Any job to any machine				
Small Jobs	0.0	81.6	81.6	373.4
Medium Jobs	0.0	147.1	147.1	
Large Jobs	0.0	283.8	283.8	
(2) Large jobs to slow machines, other jobs to any machine				
Small Jobs	0.0	74.9	74.9	611.9
Medium Jobs	0.0	125.6	125.6	
Large Jobs	0.0	548.4	548.4	
(3) Small/medium jobs to slow machines, large jobs to fast machines				
Small Jobs	0.0	165.8	165.8	273.3
Medium Jobs	0.0	251.5	251.5	
Large Jobs	0.0	254.3	254.3	
(4) Small jobs to slow machines, medium/large jobs to any machine				
Small Jobs	0.0	86.9	86.9	461.9
Medium Jobs	0.0	163.8	163.8	
Large Jobs	0.0	361.5	361.5	

policies on job completion times.

Most clusters today are heterogeneous in nature due to the fact that machines are added to a compute cluster as the need grows. It is not uncommon to find a cluster with 8, 16 and 24 core CPU machines with either 8, 16 or 32 GB of RAM. In [21], the authors deal with speculative executions of tasks so that failing tasks do not degrade the running time of a job. The authors create a new scheduler called LATE (Longest Approximate Time to End) that is based on heuristics. Again, this work does not take into account resource contention at the node level. The identification of straggler jobs, at the heart of the work in [21], would benefit from knowing if a task is falling behind because the executing node is failing or if the job is having to contend with other high demand jobs. An analytical model could accurately predict the delay a task may experience due to resource contention thus improving the LATE scheduler heuristic.

The impact of scheduling policies has been studied on various additional domains. For example, in [9] the authors analyze the tradeoffs of different scheduling algorithms on achieving performance goals in multimedia storage systems. In [3] the authors discuss content-aware scheduling of virtual machines in cloud systems.

6 Concluding Remarks and Future Work

Job schedulers play a very important role in many large enterprise IT infrastructures. A myriad of distributed applications run on multi-node clusters of heterogeneous servers. It is important to test the efficacy of a particular scheduling scheme in an extensive manner before it is put in production. It is generally not feasible to do a live test of schedulers with real jobs and compute clusters. In this report, we proposed an assessment method that schedules job traces of varied workload characteristics. The approach explores the ability to experiment with complex schedulers, which are actually implemented to process a global job trace, with a server cluster that is modeled as a collection of closed queuing networks, one per server. Therefore, no actual cluster is needed to evaluate a given scheduler algorithm. The glue between the scheduler implementation and the server analytic models is the Epochs algorithm [14], which was validated experimentally using real jobs.

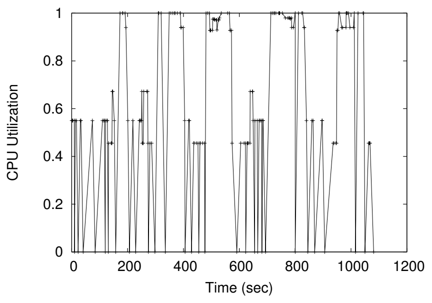
We plan to extend the TDAM approach into modeling memory contention generated by servers with a large number of cores sharing memory resources (e.g., caches and buses). Our hierarchical closed queuing network model [2] will be the basis of such extension.

We plan to extend the Mumak simulator [24], which comes bundled with Hadoop, with an analytical performance model backing its trace execution simulation. This will allow Mumak, which can handle job traces of MapReduce jobs, to analyze the makespan of these jobs more accurately and possibly shed light on the most appropriate scheduling schemes to use for a given workload.

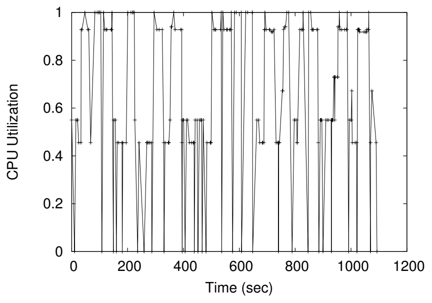
References

[1] D. Ansaloni, L.Y. Chen, E. Smirni, A. Yokokawa, and W. Binder, *Find your best match: predicting performance of consolidated workloads*, Proc. 3rd ACM/SPEC Int. Conf. Performance Engineering (ICPE 2012), Boston, Massachusetts, USA, 2012, pp. 243-244.

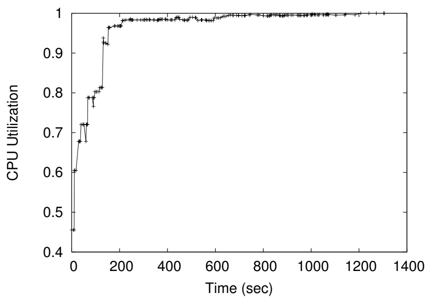
[2] S. Bardhan and D.A. Menascé, *Analytic Models of Applications in Multi-core Computers.* Proc. 2013 IEEE



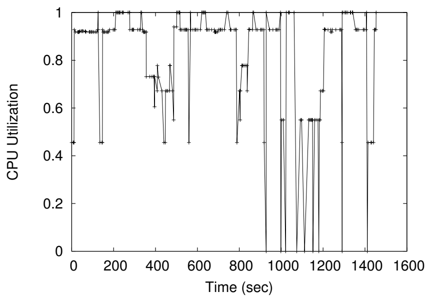
(a) CPU Utilization vs. time for a typical server in the cluster: RR for workload 1



(b) CPU Utilization vs. time for a typical server in the cluster: LMUF-T for workload 1



(c) CPU Utilization vs. time for a typical server in the cluster: RR for workload 2



(d) CPU Utilization vs. time for a typical server in the cluster: LMUF-T for workload 2

Figure 5: CPU Utilization vs. time for different workloads and scheduling schemes

21st Intl. Symp. Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2013), IEEE Computer Society, 2013.

[3] S. Bazarbayev, M.A. Hiltunen, K.R. Joshi, W.H. Sanders, and R.D. Schlichting, *Content-Based Scheduling of Virtual Machines (VMs) in the Cloud.*, ICDCS 2013, pp. 93-101.

[4] J.P. Buzen and P.J. Denning, *Operational Treatment of Queue Distributions and Mean-Value Analysis*, Computer Performance, IPC Press, Vol. 1, No. 1, June 1980, pp. 6-15

[5] J.P. Buzen and P.J. Denning, *Measuring and Calculating Queue Length Distributions*, IEEE Computer, April 1980, pp. 33-44

[6] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters*, Comm. ACM 51.1 (2008): 107-113.

[7] D. Feitelson et al., eds., *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, Springer Verlag, Vol. 3277, 2005.

[8] D. Feitelson et al., eds., *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, Springer Verlag, Vol. 3834, 2005.

[9] L. Golubchik, J.C.S. Lui, E. de Souza e Silva, H.R. Gail, *Performance Tradeoffs in Scheduling Techniques for Mixed Workloads*, Multimedia Tools Appl., 22(2), 2003, pp. 147–172.

[10] Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queuing Theory in Action*, Cambridge University Press, 2013.

[11] H. Herodotou, F. Dong, and S. Babu, *No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics*, Proc. 2nd ACM Symp. Cloud Computing, 2011.

[12] M. Isard et al., *Quincy: fair scheduling for distributed computing clusters*, Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles, 2009.

[13] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, 1991.

[14] D.A. Menascé and S. Bardhan, *Epochs: Trace-Driven Analytical Modeling of Job Execution Times*, Technical Report GMU-CS-TR-2014-01, Computer Science Department, George Mason University, March 2014, available at http://cs.gmu.edu.

[15] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, 2004.

[16] A.E. Ramirez, B. Morales, and T.M. King, *A self-testing autonomic job scheduler*, Proc. 46th Annual Southeast Regional Conf., ACM, 2008.

[17] M. Reiser and S. Lavenberg, Mean-Value Analysis of Closed Multichain Queuing Networks, J. ACM 27 (2), 1980.

[18] F. Tian and K. Chen, *Towards optimal resource provisioning for running MapReduce programs in public clouds*, 2011 IEEE Intl. Conf. Cloud Computing (CLOUD).

[19] A. Verma, L. Cherkasova, and R.H. Campbell, *ARIA: automatic resource inference and allocation for mapreduce environments*, Proc. 8th ACM Intl. Conf. Autonomic computing, 2011.

[20] A. Verma, L. Cherkasova, and R.H. Campbell, *Resource provisioning framework for mapreduce jobs with performance goals*, Middleware 2011, Springer, pp. 165–186.

[21] M. Zaharia et al., *Improving MapReduce Performance in Heterogeneous Environments*, 8th USENIX Symp. Operating Systems Design and Implementation (OSDI), Vol. 8, No. 4. 2008.

[22] Y. Zhang, M.S. Squillante, A. Sivasubramaniam, and R. Sahoo, *Performance Implications of Failures in Large-Scale Cluster Scheduling*, in Job Scheduling Strategies for Parallel Processing, Feitelson et al., eds., Lecture Notes in Computer Science, Vol. 3277, 2005, pp. 233-252.

[23] Z. Zhang et al., *Performance Modeling and Optimization of Deadline-Driven Pig Programs*, ACM Tr. Autonomous and Adaptive Systems (TAAS) 8.3 (2013): 14.

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

[25] http://www.coker.com.au/bonnie++/

[26] http://dbench.samba.org/

[27] http://www.tux.org/ mayer/linux/bmark.html

[28] https://hbase.apache.org/

[29] https://accumulo.apache.org/

[30] http://flume.apache.org/

[31] http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html

[32] http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html