A Contention-Aware Hybrid Evaluator for Schedulers of Big Data Applications in Computer Clusters

Shouvik Bardhan
Department of Computer Science
George Mason University
Fairfax, VA 22030
Email: sbardhan@masonlive.gmu.edu

Daniel A. Menascé
Department of Computer Science
George Mason University
Fairfax, VA 22030
Email: menasce@gmu.edu

***Abstract*—Large enterprises use clusters of computers to process heterogeneous Big Data workloads. The scheduling of jobs from such workloads has a significant impact on their execution times. This paper presents a Trace Driven Analytic Model (TDAM) methodology to assess the impact of different scheduling schemes on job execution times. The method combines scheduler implementations with queuing-theoretic analytic models of task contention at cluster nodes. The usefulness of the approach is demonstrated by showing the effect of four common schedulers on the execution times of jobs derived from well-known benchmarks. This method is then implemented in Mumak, a popular Hadoop job-trace simulator making it contention-aware. The original Mumak tool ignores contention for processors and I/O at each node of the cluster. Our contentiion-aware Mumak predicts job completion times at a significantly higher level of accuracy.**

***Keywords*-analytic models, Mean Value Analysis, experimentation, queuing theory, Hadoop, Mumak**

I. Introduction

Many enterprises today run Big Data applications on a cluster of heterogeneous machines. Apache Hadoop [24] and associated software stacks are examples of such software platforms and products. Large Internet organizations like Amazon, Yahoo, LinkedIn, and Facebook run thousands of Hadoop jobs on a routine basis on clusters comprising thousands of servers. These jobs have varied completion time requirements since some are ad hoc 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 made available several different schedulers over the years to schedule MapReduce jobs 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 completion times. 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 paper provides a hybrid method, called TDAM (Trace Driven Analytic Model), that integrates the implementation of the scheduler with well-known queuing-theoretic analytic models that account for job resource contention at the various nodes.

Many papers [18], [19], [20], [21] have proposed a variety of methods to predict the completion time of MapReduce jobs. However, they do not consider job resource contention at the nodes when more than one task executes concurrently. Especially with the notion of a variable number of slots on a TaskTracker [22], it is extremely important that resource contention be estimated correctly. The work described in this paper fills the gap in today's MapReduce research which considers the completion time of a task to be independent of the number of tasks executing at the same node.

TDAM, the main contribution of this paper, relies on the integration of the scheduler under evaluation with well-known analytic closed queuing network (QN) models [15] to assess resource contention at the cluster nodes. The implemented scheduler takes as input a synthetic or a real trace of jobs of various types (in terms of resource usage) and schedules them on "servers", which are modeled by analytical QNs. 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 an operational analysis formulation [4] of mean value analysis [15] equations for finite time intervals.

We applied TDAM to assess a few common cluster scheduling policies under different workload types. We implemented TDAM in Hadoop's Mumak [2]. Mumak is a job-trace simulator distributed with Hadoop. Implementing TDAM in Mumak makes it aware of the contention experienced by tasks while executing concurrently with other tasks at server nodes. That way, the contention-aware Mumak (CA-Mumak) is able to predict task and job completion times much more accurately instead of relying on Mumak's fixed task completion times provided in the trace it uses as input. The choice of Mumak as a trace simulator for Hadoop job traces is incidental and the TDAM method can be applied to other Hadoop job simulators. Even though we have discussed the applicability of the TDAM methodology with respect to Hadoop and MapReduce, the theory and the models used in TDAM are also applicable to any scenario where job schedulers are used in a cluster of computers.

The rest of this paper is organized as follows. Section II discusses the need for assessing job schedulers. Section III describes the components of CA-Mumak. Section IV uses CA-Mumak to assess the impact of actual Hadoop schedulers (e.g., FIFO, Capacity, and Fair) on a real Hadoop job under various scenarios in which cluster size and job characteristics vary. The following section discusses the results of using the TDAM method to assess four different scheduling policies on different types of workloads (Hadoop MapReduce job trace and also non-Hadoop synthetic job trace). Section VI discusses related work and lastly section VII presents some concluding remarks and future work.

II. The Need for Robust Job Scheduler Assessment

A big challenge in today's Hadoop and similar cluster platforms 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 efficiently 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 in this paper.

Hadoop was originally 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* [28] was developed for more efficient cluster sharing. The *Fair Scheduler* [29] was introduced to maintain fairness between different cluster users. Despite the presence of many schedulers, there is really no way to test with any accuracy the performance of the schedulers for a set of jobs with varying characteristics and possibly a compute cluster with heterogeneous servers without actually running the jobs on a real cluster. Many other Hadoop 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.

One of the primary goals of our research is to show how to apply the TDAM method to help enterprises assess the effectiveness of scheduling disciplines on the performance of Big Data applications. Towards that end, we have augmented Hadoop's Mumak simulator with the TDAM method and made Mumak contention aware.

III. The Components of CA-Mumak

This section describes the components of CA-Mumak (see Fig. 1) and explains how TDAM works. The workload produced by the *Workload Generator* is a stream of jobs consisting of the list of tasks that compose the jobs and a few other interesting metadata about the tasks, e.g., start and finish times. For example, MapReduce jobs are composed of map tasks and reduce tasks. Firstly, the job trace data is transformed into entities suitable for consumption by the *JobTracker Manager* and the *TaskTracker Manager* components of Mumak. Similar to what a standard Hadoop MapReduce framework does, Mumak uses JobTracker and TaskTracker classes to schedule tasks from a MapReduce job trace. However, unlike Hadoop where these components run as daemon processes, Mumak brings in the functionality of these pieces in its own process space. Mumak also tracks the finishing times of the tasks and eventually the entire job and records timings. The problem with the stock version of Mumak is that it has no awareness of the elongation of task execution times because it has no notion of task service demands (i.e., the total service time per task at each node resource) or the amount of resource contention that tasks undergo when multiple tasks execute on the same node. This is where the TDAM components come in. They bring in the knowledge of service demands (e.g., CPU and I/O) for the tasks of a particular job. Armed with this information, CA-

Mumak can now not only schedule tasks in a variety of ways based on scheduler configuration, but is also able to utilize the analytical model engine that is part of TDAM to accurately predict the makespan of a set of jobs.

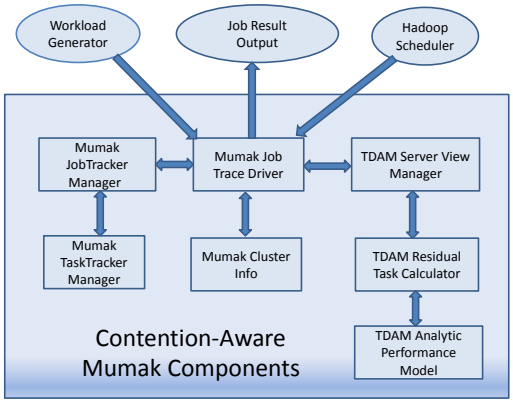

Figure 1. Components of TDAM-based CA-Mumak.

Continuing with the component diagram, the *Server View Manager* uses the well-known analytic closed queuing network (QN) algorithms [15], [4] and the server information to maintain current load state across the entire cluster as tasks come and leave worker nodes. The closed QN 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 task arrives and the end of an epoch is characterized by either the arrival of a new task or the completion of a task. The QN algorithms predict the start time of the next epoch on any cluster node by being aware of the residual times of each task as tasks start and finish executing on the nodes. The *Residual Task Calculator* component 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) [15] equations for closed QN models when flow balance, one-step behavior, and homogeneous service times are met [4]. The Epochs algorithm was validated experimentally using a micro-benchmark, real jobs from a Unix benchmark and also Hadoop MapReduce jobs [14]. The *Analytic Performance Model* block in the diagram 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.

As the simulation starts, Mumak reads in a Hadoop job-trace and the tasks for these jobs are executed virtually (since these jobs are not really executing on servers) and are scheduled to run on different nodes of a cluster utilizing the built-in JobTracker and TaskTracker inside of the simulator. Mumak also takes a cluster topology as input which consists of a certain number of TaskTrackers (execution nodes). Mumak then uses the configured Hadoop scheduler (e.g., FIFO) to schedule tasks on a TaskTracker with the understanding that as soon as the expected time of the task is complete, it would let the other components (e.g., JobTracker) know about the completion and thus be ready to receive the next task if required. The expected completion time of a scheduled task as determined from the trace by Mumak, is saved in the events to go-off after the timer expires in a typical event-calendar-driven simulator fashion. The idea behind CA-Mumak is to let Mumak schedule jobs from Hadoop job-traces as it would normally do but augment the calculation of the completion time more accurately when the execution environment (e.g., the mix of co-executing tasks) changes. The third set of inputs to CA-Mumak is a set of service demands (CPU and I/O) for various tasks of the jobs. These have to be measured by running the tasks in isolation and profiling resource demands.

Without the TDAM method, Mumak will simply consider the completion time noted in the trace for each particular task. But, the trace-specified completion time is only valid for a task if the same set of other tasks were running on the worker node when the task in question was in execution. In other words, if according to the trace file a task T1 took t minutes to execute on a machine, then this execution time t is valid only for the mix of tasks memorialized in the trace. If the task mix were to be different (either more or less time consuming) then the completion time t for task T1 is no longer valid. The TDAM method rectifies this situation because it keeps a complete view of all the tasks executing on each node in a dynamic fashion. Therefore, it is able to predict, as tasks start and finish on each worker node, the execution time of a task with a high degree of accuracy utilizing queuing theory principles. It is therefore possible to estimate the makespan of jobs on different size clusters and also analyze "what-if" scenarios using different schedulers and their configurations.

IV. MapReduce Job Scheduler Assessment with CA-Mumak

This section discusses the results of the job scheduler assessment using stock and CA-Mumak and compares the results with job execution time values obtained by running MapReduce jobs on a real cluster using three different Hadoop job schedulers under Mapreduce 1.0. We considered map-task-only jobs (we encounter these kind of MapReduce jobs in Big Data handling production environments performing bulk ingestion operations) for the results shown here. We assumed that all tasks are data local which is not the case in actuality for a large enough cluster. We have also not considered network contention. Additionally, the service demands will naturally change for a task when the input file to a job is smaller or larger than the size on which the

service demands were calculated. We have therefore made sure during the experiments on a real cluster that the number of map tasks created and the data locality of the tasks are maintained as far as possible.

We first discuss the results of simulation runs using the three main Hadoop schedulers (FIFO, Fair, and Capacity) with stock and CA-Mumak. Tables I-III show the results of several job simulation runs on a 1- and 2-node cluster with 20 map slots on each TaskTracker. There are two types of jobs (column 1) and their service demands are: 100 sec of CPU and no I/O demand for type 1 jobs and 70 sec of CPU and 30 sec of I/O for type 2 jobs. Column 2 shows the number of jobs of each type. It is assumed that the TaskTrackers run on single core computers. Note that for stock Mumak the total makespan for the job with 10 tasks (row 1 of each table) is just 131 sec, which is clearly not possible since 10 map tasks are running at the same time and contending for the core. CA-Mumak shows a total time of around 1000 sec which makes sense since 10 tasks each with 100 sec of CPU service demand sharing a 1-core machine will finish in about 1000 (100×10) seconds. The other rows of each table show makespans for various other combinations of cluster size, job size, and number of jobs. It can be seen that in each case stock Mumak results show very optimistic completion times.

Table I
MAKESPAN USING HADOOP'S FIFO SCHEDULER

Job Type	No. Jobs	Tasks per Job	No. Nodes	Mumak Time (sec)	CA-Mumak Time (sec)
1	1	10	1	131	1004
1	2	10,10	1	131, 161	1991,2003
1	2	10,10	2	116, 131	995, 1004
2	2,2	10,10	5	120, 126, 132, 138	795, 810, 816, 816

Table II
MAKESPAN USING HADOOP'S FAIR SCHEDULER

Job Type	No. Jobs	Tasks per Job	No. Nodes	Mumak Time (sec)	CA-Mumak Time (sec)
1	1	10	1	102	1002
1	2	10,10	1	102, 600	1989, 2001
1	2	10,10	2	102, 216	1002, 1155
2	2,2	10,10	5	102, 207, 351, 522	711, 849, 1002, 1062

Table III
MAKESPAN USING HADOOP'S CAPACITY SCHEDULER

Job Type	No. Jobs	Tasks per Job	No. Nodes	Mumak Time (sec)	CA-Mumak Time (sec)
1	1	10	1	131	1004
1	2	10,10	1	131, 161	1991, 2003
1	2	10,10	2	116, 131	995, 1004
2	2,2	10,10	5	107,113, 119,125	782,797, 803,803

We now discuss experiments based on a production level map-only MapReduce job ran on a 7-node 4-core per node cluster. This job consists of 200 map tasks, is not very memory intensive, and is used for ingesting data from various sources (e.g., twitter historical feed, flight information) into a specialized store for visualization and analysis. The job-trace file included the start and finish times of each map task for the job. We measured the task CPU and I/O service demands for the job by running a few map tasks in isolation (easily achieved by making the job run on a single TaskTracker node having only 1 map slot). The CPU and disk service demands for the map tasks were found to be about 98 sec and 25 sec, respectively.

Table IV shows the job's running time results obtained by running this job and by predicting these values through Mumak and CA-Mumak. We varied the number of TaskTrackers in the experiments (i.e., number of nodes in the cluster) from 3 to 5 to 7 and also varied the number of map slots available on each node from 2 to 4 to 8. The % relative error ([experiment - predicted]/experiment) is clearly better for CA-Mumak. For stock Mumak the % error range is between -26% and 51%, whereas for CA-Mumak the range is between 2% and 15%.

We experimented with two other similar jobs. One with 670 and another with 100 map tasks. The results are statistically similar to the results presented here. It is to be noted that during our experiments, when we varied the number of TaskTrackers, some of the tasks were not data-local anymore. We have not considered these variances and also have not considered network contention, which could be very high for a host of different MapReduce job types

V. SCHEDULER ASSESSMENT FOR NON-HADOOP JOBS USING TDAM

This section discusses the results of assessing four different scheduling policies using TDAM. The workload consists of streams of tasks derived from well-known Unix benchmark programs. The choice of the scheduling schemes is intended to showcase the variety of insights that one can glean regarding the performance metrics at compute nodes, such as throughput, response time, utilization and queue lengths, which are essential to predict completion times of individual tasks. The four non-preemptive scheduling policies discussed here are:

- 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 task is predicted to have the least response time. Since the scheduler has the state of all tasks running on all servers, it can predict the response time of the incoming task if it were added to any node in the cluster.

Table IV
CONTENTION AWARE MUMAK VS. STOCK MUMAK RESULTS WITH HADOOP FIFO SCHEDULER

No. TaskTrackers	Map slots Per TaskTracker	Mumak Makespan (sec)	CA-Mumak Makespan (sec)	Actual Job Running Time (sec)	%Error on Mumak	%Error on CA-Mumak
3	2	3791	4592	5200±93	27.1	11.7
3	4	1934	2807	3100±46	37.6	9.5
3	8	1202	2216	2454±36	51.0	9.7
5	2	3107	2702	2868±55	-8.3	5.8
5	4	1844	1652	1844±25	0.0	10.4
5	8	1028	1232	1454±29	29.3	15.3
7	2	2816	2027	2235±45	-26.0	9.3
7	4	1613	1332	1429±39	-12.9	6.8
7	8	1109	986	1010±32	-9.8	2.4

- Least Maximum Utilization First (LMUF): Assigns the incoming task to the server with the minimum utilization for the bottleneck resource (i.e., the resource with the highest utilization). The utilization of a resource at each server is calculated as a snapshot at the time the new task arrives to be scheduled.
- Least Maximum Utilization First-Threshold (LMUF-T): Similar to LMUF except that a task is not sent to a server if the utilization of its bottleneck resource exceeds a certain threshold. In that case, the task is queued at the scheduler. When a task completes at any node, the scheduler attempts to send the queued task again to one of the servers. The goal of LMUF-T is to bound the contention at each node by having tasks 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 created job streams by randomly selecting jobs from one of 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]. The job stream files created, along with the scheduling scheme and the number of servers in the cluster are the main input parameters to the experimental runs.

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 nodes 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 multi-task jobs on a heterogeneous cluster.

A. Effect of Scheduling Policy on the Makespan

This subsection considers how the scheduling policy affects the makespan, i.e., the time needed to execute all jobs in a job stream. Table V 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 V). 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 V, we obtained four different workloads by combining two service demand sets and two average inter-arrival time values.

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 V
(CPU, DISK) SERVICE DEMANDS (IN SEC) FOR BENCHMARK JOBS BONNIE++, NBENCH, AND DBENCH, AND TWO VALUES OF THE WORKLOAD INTENSITY

For each workload in Table V, we created 10 job streams by randomly selecting, with equal probability, the type of job at each arrival instant. Figure 2 depicts the makespan for all 10 streams for workload 1 (other workload charts depict similar results but are omitted to save space) and for each of the four scheduling disciplines. The utilization threshold used in LMUF-T is 70% for all workloads. This value is used here for illustrative purposes only. Note that the y-axis does not start at zero so that the differences between the schedulers are easier to visualize. The following conclusions can be drawn from the data. First, LMUF-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 LMUF-T is inferior than the other policies for a

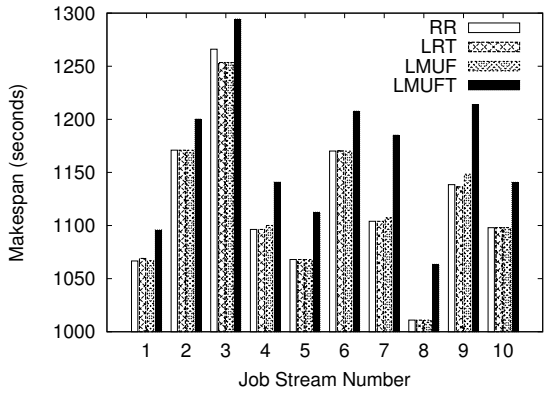

Figure 2. Makespan vs. job stream for single-task jobs in workload 1.

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 node and the utilization of its bottleneck device. Finally, RR is worse than LRT and LMUF for jobs 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 for workload 1 (and other workloads not shown). LMUF-T was shown to be significantly different at the 95% confidence level than the three other scheduling disciplines. We also applied the Tukey-Kramer procedure [13] to the three other scheduling disciplines and found statistically significant differences among them in several of the tested workload.

B. Threshold Effect on LMUF-T

We now discuss the effect of the bottleneck threshold value used by LMUF-T. In that case, the total execution time of a job consists of two components: wait time at the scheduler and time spent at a node, which includes time spent using resources and time waiting to use node resources.

Figures 3(a) and 3(b) show the waiting time at the scheduler, server time, and total time for workloads 1 and 3 versus the CPU utilization threshold for Bonnie++ and Nbench, respectively. These workloads contain a mix of the three types of jobs in Table V. However, each graph only shows average values for a specific job within the multi-job workload. Concentrating on Figure 3(a), as the CPU utilization threshold approaches 100%, the wait time (the line at the bottom of the chart) goes to zero as expected because no jobs will be queued at the scheduler. However, as seen in the figures (not all the graphs are shown to save space), 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 (not shown) the lowest execution time is reached for a CPU utilization threshold equal to 100%. Figure 3(b) uses the same service demands as in Figure 3(a) but with an average arrival rate twice as large. This corresponds to workload 3 in Table V. There is a marked difference in behavior between the graphs of Figure 3. 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 Nbench (as well for Bonnie++ and Dbench).

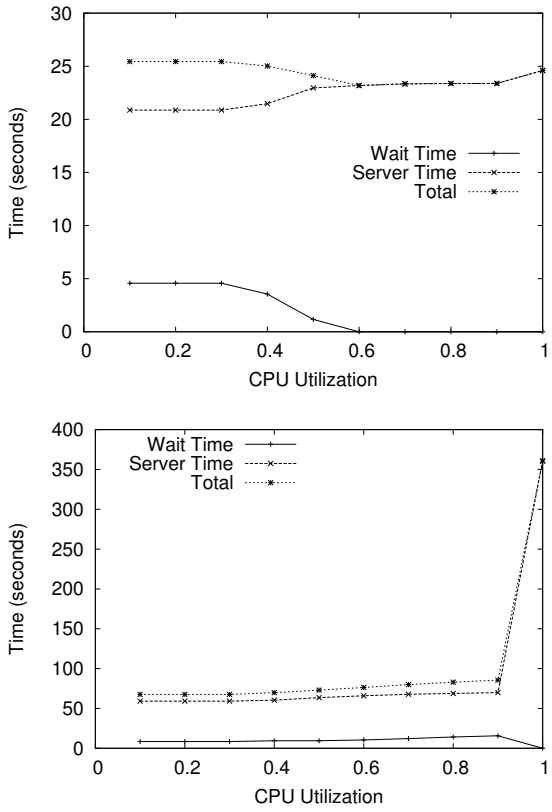

Figure 3. Average execution time, wait time, and server time vs. CPU utilization threshold. Top: (a) Bonnie++ and workload 1. Bottom: (b): Nbench and workload 3.

Figure 4(a) shows the CPU utilization for a typical server in the cluster using RR scheduling for workload 2. Figure 4(b) shows similar data for LMUF-T scheduling with a 70% threshold for the same workload. In the RR case, the CPU utilization quickly reaches 100% and stays there. However, in the LMUF-T case, the average CPU

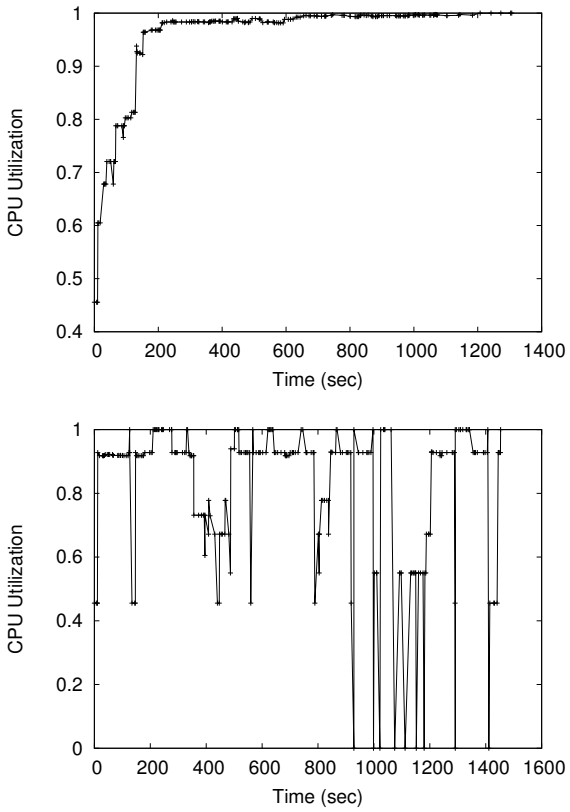

Figure 4. CPU utilization vs. time for workload 2. Top: (a) RR. Bottom: (b) LMUF-T.

utilization stays at a lower level and never stays fixed at 100% (an undesirable situation) as in Figure 4(a). This is because of the 70% threshold for CPU utilization used by LMUF-T. This sort of insights into the running of jobs with various characteristics is possible because TDAM facilitates the accurate analysis of "what-if" scenarios.

C. Assessing Synthetic MapReduce Job Traces

This subsection discusses the results from assessing a job trace consisting a set of MapReduce jobs on a cluster with heterogeneous servers.

A MapReduce job is considered finished only when all its tasks finish. Once a task is scheduled to run on a node, it can not be switched to another node (unless it fails and is retried by the framework). The workloads used are described in Table VI and they differ in terms of their CPU and disk service demands. We assume that all the jobs arrive as indicated in the job-trace file. Based on the number of tasks per job and the number of jobs in Table VI, 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 VI correspond to the faster machines. The corresponding values for the slow machines are twice the values depicted in the table.

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

Table VI
MULTI-TASK JOB CHARACTERISTICS.

Experiments were performed using LMUF-T scheduling using three utilization thresholds: 0.0, 0.7 and 1.0. Table VII shows the overall makespan for four different scheduling scenarios: (1) Any job can be scheduled on any machine, (2) 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 made from the data in Table VII (and additional data collected from the experiment and not shown here). First, for a given CPU utilization threshold value, the best makespan values are obtained either when any job can be scheduled on any machine (row 1) or when fast nodes are used exclusively by large jobs (row 3). Second, the data shows a clear impact of the threshold on server congestion. For example, for a CPU threshold of 0.0 (similar to the Hadoop approach of granting exclusive access to CPU cores to each task) there is 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% , 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. Third, the case in which there is no queuing at the scheduler 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 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.

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 paper can easily be applied to a wide variety of "what-if" scenarios. The reason is that the scheduler implementation is unchanged 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 methodology is that it allows for any scheduling discipline to be easily and accurately assessed for any size and type of cluster given that task/job contention for server resources is modeled analytically.

	Overall Makespan ($U_{\text{cpu}} \leq 0.0$)	Overall Makespan ($U_{\text{cpu}} \leq 0.7$)	Overall Makespan ($U_{\text{cpu}} \leq 1.0$)
Any job to any machine	391	344	373
Large jobs to slow machines, other jobs to any machine	675	675	612
Small/Medium jobs to slow machines, large jobs to fast machines	405	343	273
Small jobs to slow machines, medium/large jobs to any machine	413	347	462

Table VII
MAKESPAN (IN SEC) FOR SMALL, MEDIUM, LARGE MAPREDUCE JOBS FOR DIFFERENT CPU UTILIZATION THRESHOLDS

VI. 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 [9]. In [9], 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 [9] does not consider the possibility of queuing at the scheduler, as is done in LMUF-T. Several papers on job scheduling for parallel environments appear in [7] and [8].

During the last half decade or so, performance analysis and modeling of MapReduce jobs has received significant attention and several different approaches have been presented [10], [17]. 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 [18], [19], [23].

In [10], the authors discuss a query system to answer cluster sizing problems using a combination of job profiles and estimations. The authors in [17] 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 [18] 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 policies on job completion times.

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. In [11], the authors created a novel scheduler called "Maestro" that prevents excessive speculative executions of map tasks by scheduling the tasks in two waves which lead to a higher data-locality of the tasks. Again, these efforts do 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 authors in [5] created a MapReduce-based GIS workflow engine with its own scheduler called the MRGIS scheduler. This scheduler has a rather specialized way of scheduling tasks on Hadoop nodes and would greatly benefit by incorporating a performance model to track the congestion on worker nodes before tasks are actually scheduled.

VII. CONCLUDING REMARKS AND FUTURE WORK

Job schedulers play a very important role in many large enterprise IT infrastructures. Many 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 live runs with real jobs and large compute clusters to test scheduling scenarios. In this paper, 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.

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 machines with either 8, 16 or 32 GB of RAM. The approach presented here allows for the easy analysis of scheduling disciplines on Big Data jobs running on heterogeneous clusters.

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 [3] will be the basis of such extension. We also plan to model the other phases of a MapReduce jobs like shuffle, sort, and reduce.

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] Apache, Mumak: Map-Reduce Simulator, 2010. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-728

[3] S. Bardhan and D.A. Menascé, *Analytic Models of Applications in Multi-core Computers.* Proc. 2013 IEEE 21st Intl. Symp. Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2013), IEEE Computer Society, 2013.

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

[5] Q. Chen et. al, *MRGIS: A MapReduce-Enabled High Performance Workflow System for GIS*, IEEE 4th Intl. Conf. eScience, 2008.

[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*, LNCS, Springer Verlag, Vol. 3834, 2005.

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

[10] 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.

[11] S. Ibrahim, et. al, *Maestro: Replica-aware map scheduling for mapreduce,*12th IEEE/ACM Intl. Symp. Cluster, Cloud and Grid Computing (CCGrid), 2012.

[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/˜tr-admin/papers/GMU-CS-TR-2014-1.pdf

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

[18] 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.

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

[20] Verma, Abhishek, Ludmila Cherkasova, and Roy H. Campbell. "Play it again, SimMR!." Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, 2011.

[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] Tang, Shanjiang, Bu-Sung Lee, and Bingsheng He. "Dynamic slot allocation technique for MapReduce clusters." Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013.

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

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