Reducing Job Slowdown Variability for Data-Intensive Workloads

Bogdan Ghiț[†] and Dick Epema[†§]
[†]Delft University of Technology, the Netherlands
[§]Eindhoven University of Technology, the Netherlands
{b.i.ghit,d.h.j.epema}@tudelft.nl

***Abstract*—A well-known problem when executing data-intensive workloads with such frameworks as MapReduce is that small jobs with processing requirements counted in the minutes may suffer from the presence of huge jobs requiring hours or days of compute time, leading to a job slowdown distribution that is very variable and that is uneven across jobs of different sizes. Previous solutions to this problem for sequential or rigid jobs in single-server and distributed-server systems include priority-based FeedBack Queueing (FBQ), and Task Assignment by Guessing Sizes (TAGS), which kills and restarts from scratch on another server jobs that exceed the local time limit. In this paper, we derive four scheduling policies that are rightful descendants of existing size-based scheduling disciplines (among which FBQ and TAGS) with appropriate adaptations to data-intensive frameworks. The two main mechanisms employed by these policies are partitioning the resources of the datacenter, and isolating jobs with different size ranges. We evaluate these policies by means of realistic simulations of representative MapReduce workloads from Facebook and show that under the best of these policies, the vast majority of short jobs in MapReduce workloads experience close to ideal job slowdowns even under high system loads (in the range of 0.7-0.9) while the slowdown of the very large jobs is not prohibitive. We validate our simulations by means of experiments on a real multicluster system, and we find that the job slowdown performance results obtained with both match remarkably well.**

I. INTRODUCTION

The ever-growing amounts of data collected and processed by clusters and datacenters cause large disproportions between the sizes of large-scale data-analytics jobs and short interactive queries executed in single systems. As is well known, in the face of a skewed or even heavy-tailed job size distribution, achieving fairness across jobs of different sizes is difficult as small jobs may be stuck behind large ones. For sequential jobs, this problem can be addressed in single-server systems by feedback queuing [20] or processor sharing [7] and in distributed-server systems by having each server process jobs of sizes in a certain range [8], [11]. In some of the policies in the latter case, when job sizes are not known apriori, jobs are restarted from scratch elsewhere, thus wasting processing capacity, when they exceed a local time limit. In contrast, data-analytics jobs using such programming models as MapReduce, Dryad, and Spark have much inherent parallelism, there is no natural way of splitting up resources of a datacenter for specific job size ranges, and jobs may be so large that wasting resources spent on partial executions is not acceptable. In this paper, we propose and simulate four scheduling policies which are rightful descendants of existing size-based disciplines for single-server and distributed-server systems with appropriate adaptations to data-intensive frameworks.

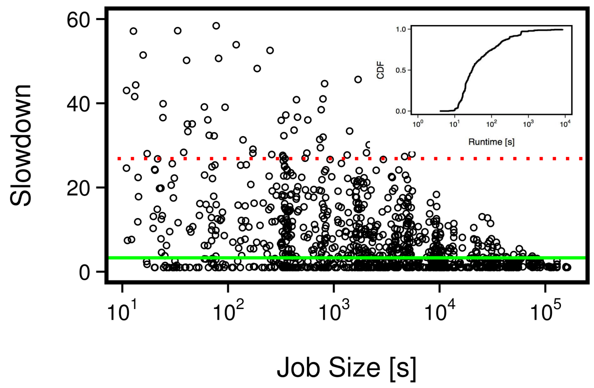


Fig. 1: A scatter plot of the job slowdown versus the job size for a heavy-tailed MapReduce workload from Facebook (the small figure shows the CDF of the runtimes) with the FIFO scheduler under a system load of 0.7 (the horizontal lines indicate the median and the 95^{th} percentile of the slowdown).

Fairness, in both single-server and distributed-server systems, and more recently in clusters and datacenters, can be considered to be satisfied when jobs experience delays that are proportional to their sizes, which in this paper is defined as their total processing requirements. Traditionally, the performance of scheduling disciplines with respect to fairness has been measured using job slowdown as a metric. In fact, two dimensions of this metric are relevent—policies designed for highly variable workloads are considered to be fair to the extent that the total distribution of the job slowdown has a low variability, and that it is not biased for certain job size ranges. Therefore, in this paper the targets are to reduce the *variability of the slowdown* defined as the ratio of the 95^{th} percentile and the median of the job slowdown distribution without significantly increasing the median slowdown, and to even the job slowdowns across the whole range of job sizes.

As an example of the phenomenon we want to tackle, in Figure 1 we show the slowdowns versus the sizes for the jobs in a Facebook workload that is scheduled with the standard MapReduce FIFO scheduler. Here, the median and the job slowdown variability (as just defined) are 3 and 9, and clearly, the small and medium-sized jobs experience the higher and more variable job slowdowns. The inset of the figure shows the CDF of job runtimes, and shows a difference of 3 orders of magnitude between the smallest and the largest jobs. Further, we find that less than 7% of the jobs in the Facebook workload account for almost half of the total load.

The common denominator of policies for isolating the performance of jobs of different sizes that have been studied in the past is splitting the workload across multiple queues that only serve jobs (or parts of jobs) with processing requirements in certain ranges. Indeed, multi-level FeedBack Queueing (FBQ) [20] is a priority-based single-server time-sharing policy that relies on preemption of jobs without loss of work already done. In contrast, the Task Assignment by Guessing Sizes (TAGS) policy [11] is designed for distributed server systems, where jobs get killed and are restarted from scratch when being moved to the next queue when they exceed a time limit. Similarly, when job sizes are known (or estimated) apriori, jobs can be immediately dispatched to the appropriate queue upon arrival as in Size-Interval Task Assignment (SITA) [12]. Another interesting way to do the same without knowing job sizes is done by the COMP policy that compares the estimated size of an arriving job to the sizes of some number of the last previously departing jobs [18].

Data-intensive frameworks such as MapReduce have a job model that is very flexible. Jobs consist of many tasks with loose synchronisation points between successive stages (e.g., map and reduce), which makes them malleable or elastic. The shared distributed file system of MapReduce allows any task to run on any processor in the datacenter. So the opportunity exists to run multiple tasks of a single job in parallel and to run multiple jobs simultaneously, as opposed to the rigid job model supported by FBQ and TAGS in single and distributed servers. Therefore, we have the option to partition the resources of a datacenter across queues, mimicking the operation of distributed-server systems, or to have all queues share the whole non-partitioned datacenter. Moving jobs from one partition/queue to another may be done without killing them by keeping the work previously completed in the distributed file system.

With the mechanisms employed by our policies the vast majority of short jobs in MapReduce workloads experience close to ideal job slowdowns even under high system loads (in the range of 0.7-0.9), at the expense of higher slowdowns for a relatively small fraction of large jobs (less than 5%). Further, our policies consistently improve the slowdown variability over FIFO by a factor of 2.

The main contributions of this paper are:

1) We derive four multi-queue size-based policies for data-intensive workloads with skewed, unknown job sizes that isolate jobs of similar sizes either by migrating them across different queues or partitions without loss of previously completed work, or by judiciously selecting the queue to join (Section IV).
2) With a set of real-world experiments, we show that our simulations are remarkably accurate even at high percentiles of the job slowdown distribution (Section V). With a comprehensive set of simulations, we analyse and compare the effectiveness of our scheduling policies in reducing the slowdown variability of heavy-tailed MapReduce workloads (Section VI).

II. MapReduce Model

MapReduce [9] and its open-source implementation Hadoop are widely used in clusters and datacenters for processing large, regular datasets by exploiting the parallelism of the applications involved. With this model, jobs that process very large datasets (terabytes of data) can be easily executed on clusters.

Internally, MapReduce jobs are structured in *multiple phases*, each of which having a homogeneous set of parallel tasks, and so the number of tasks of jobs is proportional to the size of their input data. A running MapReduce job goes through three successive possibly overlapping phases: the map phase that runs a user-defined function on each data block of the input data set and generates key-value pairs, the shuffle phase that sorts the output of the map phase and divides it among the reduce tasks, and finally, the reduce phase that runs a user-defined function to aggregate the intermediate data. The shuffle phase may start transferring intermediate data to reduce tasks when a predefined fraction of the map tasks have completed (by default 0.05 in Hadoop). However, reduce tasks can only start their actual processing after the entire map phase is finished. This precendece constraint may pollute the job runtime with significant delays between consecutive phases of MapReduce.

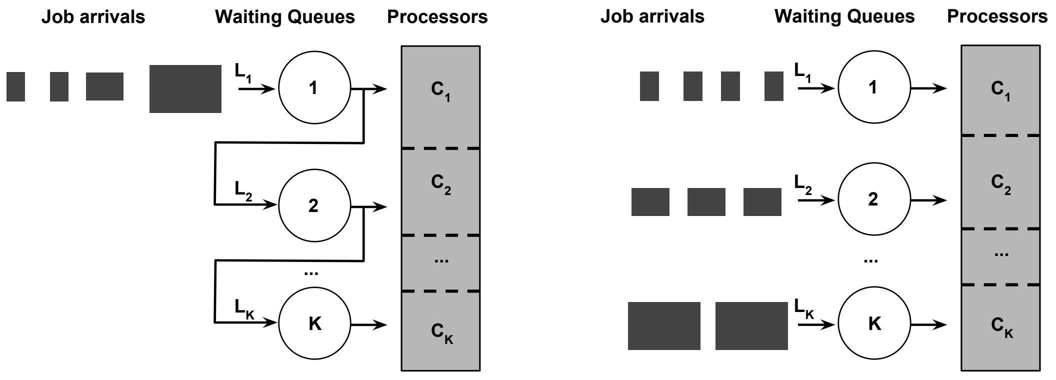
A MapReduce framework is implemented using a master-worker model and requires the input dataset of jobs to be distributed across a shared distributed filesystem. For instance, the well-known high-throughput Hadoop Distributed File System (HDFS) can easily be deployed on standard hardware and is suitable for applications with very large data sets. The data are stored in blocks of fixed size (e.g., 128 MB) that are replicated in the HDFS for fault tolerance. The internal job scheduler in MapReduce uses FIFO with five priority levels. To assign tasks to an idle worker, jobs are selected in order of their arrival time. Once a job is selected, the scheduler executes either a map task with data closest to the worker, or the next reduce task in line.

An interesting element of data-intensive frameworks is the underlying shared filesystem, which enables fine-grained resource sharing across different (sets of) jobs. In this way, users may run their jobs across disjoint datasets, without replicating their data across clusters. MapReduce jobs are flexible parallel jobs that may be *paused*, their intermediate results saved in the distributed filesystem, and later gracefully *resumed* without losing previously completed work. Thus, the underlying shared file system in our model enables a work-conserving approach to preemption, which is an important assumptions in the design of our scheduling policies.

III. Job Slowdown Variability

In this paper we define the *processing requirement* or the *size of a job* to be the sum of the its task runtimes. Clusters and datacenters running frameworks for big-data applications such as MapReduce, Dryad, and Spark are consistently facing workloads with high job size variability, thus raising concerns with respect to large and/or imbalanced delays across the executed jobs [24], [29]. The tension between fast service and fair performance has been an important design consideration in many computer systems such as web servers and super-computers [10], [22], which are known to execute workloads containing jobs with processing requirements characterized by heavy-tailed distributions.

Whereas users may tolerate long delays for jobs that process large data sets, but most likely expect short delays for small interactive queries, job slowdown, that is, the sojourn time of a job in a system normalized by its runtime in an empty system, is widely used for assessing system performance. The question then is, what statistic of the job slowdown distribution to use. In this paper, in order to characterise *fair performance* in clusters and datacenters with data-intensive workloads, we use a metric that we call the *job slowdown variability*. Let F be



(a) FBQ and TAGS *with* feedback – *unknown* job sizes. (b) SITA and COMP *without* feedback – *known* job sizes.

Fig. 2: Two general queueing models for reducing job slowdown variability with a single partition or multiple partitions.

the cumulative distribution function of the job slowdown when executing a certain workload in a system, and let $F^{-1}(q)$ be the q^{th} percentile of this distribution. Then the *job slowdown variability at the q^{th} percentile*, denoted by $V_F(q)$, is defined as the ratio of the q^{th} percentile of F and the median job slowdown, that is:

$$V_F(q) = \frac{F^{-1}(q)}{F^{-1}(50)}. \tag{1}$$

Intuitively, the slowdown variability at a certain percentile captures some subrange of the slowdowns of all jobs. In the ideal case, $V_F(q) = 1$ for all values of q between 0 and 100, meaning that all jobs have equal slowdowns. Then the policy employed can be called *strictly fair* (for this workload), although that notion has been previously defined when equality of slowdowns holds in expectation [27]. Our target is to minimize the job slowdown variability at different percentiles q, in particular, at $q = 95$, while keeping the median job slowdown low. In this paper we call $V_F(95)$ the (*overall*) *job slowdown variability* of the workload.

To put fairness in large complex systems a bit in perspective, in an M/G/1 system with load $\rho < 1$, the expected slowdown for any job size under the processor-sharing (PS) discipline is $1/(1-\rho)$ [28]. Further, there is no policy that is both strictly fair and has slowdown strictly less than $1/(1-\rho)$ [27]. Obviously, such strict fairness guarantees lead to performance inefficiency when compared with the Shortest-Remaining-Processing-Time (SRPT) discipline. Not only is SRPT response time-optimal, but the improvement over PS with respect to the mean sojourn time is at least a factor of 2 [4]. Interestingly, despite the general concern of starving large jobs, the degree of unfairness under SRPT is relatively small when the job sizes are drawn from heavy-tailed distributions.

IV. Scheduling Policies

In order to reduce the job slowdown variability in data-intensive frameworks with jobs that have highly variable processing requirements, in this section we will present four scheduling policies that are inspired by multi-level scheduling with feedback in single-server systems and by size-based scheduling in distributed-server systems.

A. Mechanisms and Queueing Models

The two mechanisms used by our policies are *logical partitioning* and/or *system feedback*. With the former mechanism, we allocate the compute resources (processors or slots) across disjoint partitions and we restrict for each such partition the amount of service offered to jobs. With the latter mechanism, we use job preemption in a *work-conserving* way by pausing a running job, saving its completed work in the distributed file system, and later gracefully resuming its execution from where it left off. In Figure 2 we show the two queueing models for that result. The main difference between the two models is whether job sizes are unknown or known, possibly by means of predictions. We propose four policies by combining the two mechanisms in all possible ways.

All our policies have as a parameter some number $K, K > 1$, of waiting queues that serve jobs in FIFO order. Each queue $k = 1, 2, \ldots, K$ has an associated *time limit* L_k, which is set to the total amount of service that jobs may receive while they are in queue k; L_K is set to ∞. To satisfy our goal, it is crucial that jobs of similar sizes reach the same queue, either through feedback to lower queues (Figure 2a) or immediately upon arrival (Figure 2b). In addition, each queue k may be associated with a *partition* of size C_k of the total set of resources (hence the dotted lines in the figures); if so, jobs from queue k are restricted to using resources in the corresponding partition.

B. The FBQ Policy

Our version of FeedBack Queuing (FBQ) is an extension of multi-level feedback scheduling for the M/G/1 queue [20] to a data-intensive framework running in a datacenter or cluster. It uses the queueing model of Figure 2(a) with feedback but without resource partitioning.

An arriving job is appended to queue 1, where it is entitled to an amount L_1 of service. If its processing requirement does not exceed L_1, it will depart the system from queue 1, otherwise it will be appended to queue 2, etc. When processors become available because a task completes, the next tasks to run are selected from the jobs waiting in the highest priority (lowest index number) non-empty queue.

Our FBQ policy for data clusters enables multiplexing so that multiple jobs at potentially different priorities may run simultaneously. Even with Processor Sharing as the queueing discipline in every queue, the latter is impossible with multi-level feedback scheduling in a single-server queue.

C. The TAGS Policy

Our version of the TAGS (*task assignment by guessing size*) policy is similar to our FBQ policy with the exception that now each queue has its own resource partition where its jobs have to run. As a consequence, whereas with FBQ jobs at different priority levels *may* run simultaneously (when the jobs in the highest non-empty queue cannot fill the complete system), with TAGS, jobs of different sizes *will* indeed run simultaneously in separate resource partitions.

Unlike its predecessor for distributed servers, our TAGS policy for data clusters does not require killing jobs when they are kicked out from one queue to another. Instead, jobs are allowed to gracefully resume their execution without redoing previously completed work. This is an essential design element which eliminates concerns related to inefficiencies of TAGS under higher loads.

D. The SITA Policy

Similarly to TAGS, SITA does employ per-queue resource partitions, but it does not use feedback. Unlike both FBQ and TAGS, SITA requires a way to predict the sizes of jobs upon their arrival, based on which they are dispatched to the queue of jobs of similar size. A job with predicted size between L_k and L_{k+1} is appended to queue $k+1$. Consequently, the queue time limits have a different role as in the previous policies. Whereas in FBQ and TAGS they are used to keep track of the amount of processing consumed by a job, in the case of SITA they are used as *cutoffs* in the job size distribution for directly dispatching them to the appropriate queue, where they run till completion, even if the prediction is wrong.

Job sizes can be estimated by building job profiles by running (a fraction of) their tasks and collecting samples of the average task duration and the size of the intermediate data they generate. This method has been adopted with good results in previous work for MapReduce jobs when assumptions of uniform task executions within different phases of a job can be made [25]. In practice, we find that a much simpler way of predicting jobs sizes based on their correlation with the job input size is very effective (see Section V-C).

E. The COMP Policy

All previous policies require setting a number of parameters that is proportional to the number of queues, which may be prohibitive in a real system deployment. We will now present a policy called COMP which is an adaptation to MapReduce of a policy that has been studied before [18]. Similar to FBQ, COMP does not use partitioning of the resources, and similar to SITA, COMP does require job size predictions to send an arriving job immediately to the appropriate queue where it runs to completion. However, in contrast to both SITA and FBQ, COMP does not use queue time limits. Upon the arrival of a job, COMP compares its estimated size to those of the last $K-1$ jobs that have been completed. If the new job is estimated to be larger than exactly m of those jobs for some $m, m = 0, 1, \ldots, K-1$, it is appended to queue $m+1$.

F. Contrasting the Policies

Despite the common goal of reducing the job slowdown variability, we observe in Table I the main differences between our policies, which may divide the system capacity across multiple partitions (e.g., TAGS and SITA), may use preemption and relegation to another queue (e.g., FBQ and TAGS), or may use some form of prediction to anticipate job sizes (e.g., SITA and COMP). Although our policies resemble the structure of former scheduling disciplines for single-server and distributed-server systems, there are three key elements in which their correspondents for datacenters are different.

Policy	Queues	Partitions	Feedback	Job Size	Param.
FIFO	single	no	no	unknown	0
FBQ	multiple	no	yes	unknown	K
TAGS	multiple	yes	yes	unknown	$2K-1$
SITA	multiple	yes	no	predicted	$2K-1$
COMP	multiple	no	no	compared	1

TABLE I: Our policy framework for scheduling data-intensive jobs in datacenters.

First, the original policies were designed for a single or distributed-server model in which each host is a single multi-processor machine that can only serve one job at a time. In contrast, we target a datacenter environment in which the system capacity may be divided across partitions with many resources. As a result, instead of only having the time cutoffs as parameters, in our model we also have the partition capacities as parameters.

Secondly, original TAGS and SITA assume simple, rigid (sequential or parallel) non-preemptible jobs that may only run on a single host until completion. In contrast, our (MapReduce) job model is more complex as there are intra-job data precedence constraints (map before reduce) and data locality preferences (of map tasks), and as jobs are elastic (or malleable) and can run simultaneously, taking any resources they can get when it is their turn.

Thirdly, original TAGS does not preserve the state of a job when it moves it from one server to the next. Hence, long jobs will get killed at every server except at the one where they run to completion, at every step losing all the work performed and thus wasting CPU time. Instead, we take a work-conserving approach by allowing jobs that are being moved from one partition to the next to retain their work and to gracefully resume their executions without redoing previously completed work. This way of operation is facilitated by the file system that is shared across the whole framework, with the intermediate results of tasks executed within any partition being persistent and visible after a job has been moved to another partition.

V. Setup

We evaluate and compare our policies with the job slowdown variability at different percentiles as the main metric for representative MapReduce workloads. Given the large space of policy configurations (e.g., the number of queues, the partition sizes, and the queue time limits), in this paper we take an experimental approach through realistic simulations of MapReduce to completely understand the impact of our policies on the slowdown variability in a MapReduce framework when jobs have very different sizes.

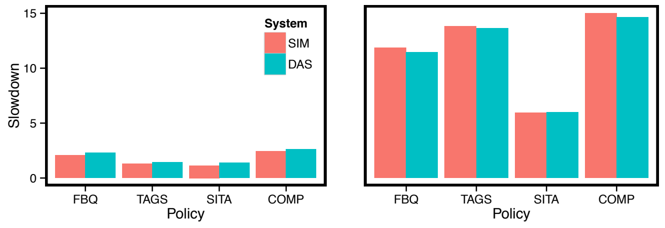
A. Simulator

We have modified Apache's open-source MapReduce simulator Mumak [19] to include our scheduling policies. Although many discrete-event simulators for traditional queueing models exist, we chose Mumak [19] for its compatibility with the popular open-source MapReduce implementation Hadoop. Thus, Mumak reproduces closely the internals of Hadoop by simulating with a discrete time scale the MapReduce job scheduling and task allocation. Furthermore, Mumak can employ without changes the standard Hadoop schedulers (e.g., FIFO, Capacity [2]).

A subtle point in simulating MapReduce is to appropriately adjust the reduce task runtimes based on the shuffle phase duration. Mumak schedules reduce tasks only when a predefined fraction of map tasks have finished (the default value is 5%).

Applications	Maps	Reduces	Job Size [s]	SIM [s]	DAS [s]	Jobs
GREP	2	1	63.14	36.10	43.26	26
SORT	4	1	60.20	32.70	39.97	4
WCOUNT	4	1	126.14	42.04	49.73	4
GREP	50	5	155.32	42.83	53.18	4
WCOUNT	100	10	3,790.46	86.80	93.62	3
SORT	200	20	5,194.64	149.92	156.89	3
GREP	400	40	15,697.18	233.63	239.21	3
WCOUNT	600	60	26,662.53	579.73	589.02	3

TABLE II: The characterisitics of the jobs used in the validation, and the job runtime in the simulations (SIM) and in the real deployment (DAS).



(a) Median (b) 95th percentile

Fig. 3: The slowdown performance of our policies for the workload summarized in Table II using both simulations (SIM) and real-world experiments (DAS).

Although Mumak allows reduce tasks to occupy slots during the map phase, their runtime duration is simulated from the moment when all map tasks are finished. As reduce tasks may still be in their shuffle phase even after all map tasks have finished, Mumak conspicuously underestimates the job completion time. To reproduce closely the shuffle phase of the MapReduce job execution, we changed the simulator and incorporated the remaining duration of the shuffle phase in the reduce task runtimes.

Mumak is not completely event-based but has time-based elements, as the hearbeats through which tasks are assigned to idle slots in the framework are simulated periodically at fixed intervals. Although useful in practice to implement the interaction between the components of the MapReduce framework (e.g., JobTracker and TaskTrackers), this artifact pollutes the simulation results by leaving slots idle between two consecutive heartbeats, thus reducing the utilization of the framework. Therefore, we changed the simulator and removed the periodic hearbeat by forcing slots to initiate a hearbeat whenever they become idle.

In all our simulations (except the validation experiments in Section V-B), we use a cluster consisting of 100 nodes on which one MapReduce framework is installed, and we configure each node with 6 map slots and 2 reduce slots. This size of our MapReduce framework is comparable to production deployments of MapReduce frameworks [3], [29]. For each simulation, we report the averages of our performance metrics over three repetitions.

B. Validation

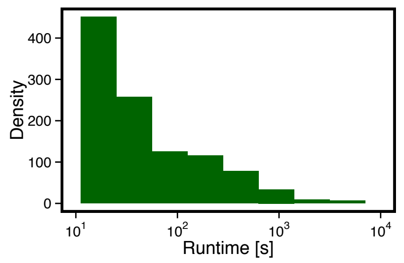
With a set of both simulations and real-world experiments, we assess the accuracy and the robustness of our modified simulator. We run real-world experiments on the Dutch six-cluster wide-area computer system DAS-4 [1]. The TU Delft cluster, which we use for this validation, has 24 dual-quad-core compute nodes, with 24 GiB memory per node and 50 TB total storage, connected within the cluster through 1 Gbit/s Ethernet (GbE) and 20 Gbit/s QDR InfiniBand (IB) networks. In our real-world experiments, we use our prototype of Hadoop-1.0.0, which includes the implementations of the four policies. For both our simulations and real-world deployment, we configure 10 nodes with 6 map slots and 2 reduce slots.

To evaluate how accurately our simulator approximates real-world executions of single MapReduce jobs, we compare the simulated runtimes with the job runtimes in the real system for 8 different job types. As we show in Table II, these jobs are instances of three well-known MapReduce benchmarks (Grep, Sort, and Wordcount) with very different numbers of tasks (between 3 and 660) and variable total processing requirements (between 1 minute and 7.4 hours). Our simulator estimates in most cases the job run times with less than 10% error (which is comparable to the error in SimMR [26] developed at HP Labs). In fact, the difference between the job runtimes in our simulator and in the real-world system is always at most 7-10 s (columns 5 and 6 in Table II). This difference represents the overhead of setting up and cleaning up a MapReduce job in Hadoop, which we do not account for in simulation.

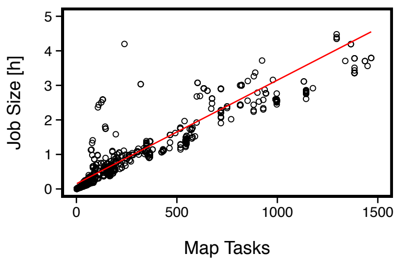
To assess the robustness of our simulator with complete MapReduce workloads, we compare the slowdown performance of our policies in simulation and real-world deployment. The workload we use in this evaluation has 50 jobs in total and consists of different fractions of job types, as indicated in Table II. We simulate and execute the workload in our cluster using each of the four policies with $K = 2$ queues. We set the size of partition 1 to 30% for both TAGS and SITA. The time limits we find to be the best for TAGS, SITA, and FBQ are 120, 160, and 160 seconds. Figure 3 shows the slowdown performance of our policies in both the simulator and the real-world deployment when job arrivals are Poisson and the system load imposed is 0.7. Our simulator is remarkably accurate and delivers slowdown performance that is consistent with the real-world experiments. The relative error between the simulations and the real-world experiments is less than 1% for both the median job slowdown and the job slowdown at the 95^{th} percentile.

C. Workloads

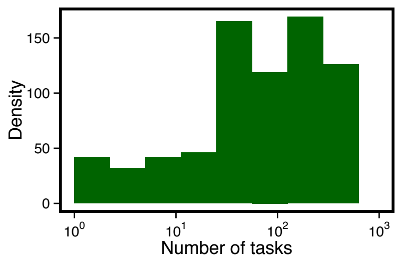
The workloads we impose on the system in our simulations are based on a MapReduce trace from Facebook that spans 6 months, from May 2009 to October 2009, and contains roughly 1 million jobs. This trace is public and contains the size in bytes for every job for each MapReduce job phase. We employ the SWIM benchmark [6] which uses the records in the trace to generate synthetic jobs. We create and execute a set of 1121 such synthetic jobs on a real 20-node Hadoop cluster and record for each job all relevant information (e.g., task runtimes) so that it can be executed in our simulator. We generate different workloads of 1121 jobs by randomly selecting jobs without repetition from this set of synthetic jobs. The job interarrival time has an exponential distribution. In order to compute the slowdowns experienced by the jobs in our simulations, we determine the reference runtime of each job in an empty 100-node simulated MapReduce framework from when its first task starts until its last task terminates. The sum of the reference runtimes of all jobs accounts for approximately 60 h of simulated time. In Figure 4a, we show the distribution of these reference job runtimes of the jobs in our workloads, which is very variable, as its squared coefficient of variation is equal to 16.35. As reported for other data-intensive workloads



(a) The density of the reference job runtimes (horizontal axis in log scale).



(b) The job size versus the job input size.



(c) The density of the number of tasks per job (horizontal axis in log scale).

Fig. 4: Properties of the workload generated with SWIM measured in a simulated empty 100-node system: a histogram of the reference job runtimes (a), the correlation between the input size and the processing requirement of jobs (b), and a histogram of the number of tasks per job (c).

at Facebook, Google, Cloudera, Yahoo! [5], the distribution of the job sizes is skewed, with fewer than 8% of the jobs in our workloads responsible for almost half of the total load. In contrast with the reference runtimes, the job sizes (total processing requirement) are less variable, with the squared coefficient of variation equal to 3.32. Further, we find in our workloads a strong correlation between the total job input size and the total processing requirement (the size) of the job (see Figure 4b), which is used by SITA and COMP to assign arriving jobs to queues. Finally, jobs in our workload may achieve very different levels of parallelism: from less than ten tasks to more than 10,000 tasks (Figure 4c).

VI. Evaluation

In this section we first investigate the impact of the key parameters on the slowdown variability (Section VI-A). Further, we analyse the effect of load unbalancing across partitions on the performance of TAGS and SITA (Section VI-B), the performance under heavy traffic (Section VI-C), and the degree of unfairness in our scheduling policies (Section VI-D). Finally, we analyze the performance of FBQ and COMP with more than two queues (Section VI-E).

A. Parameter Sensitivity

All our policies assume a single MapReduce framework with some number K of queues. Although having multiple partitions may reduce the job slowdown variability by starting short jobs faster rather than having them wait behind relatively large jobs, in practice, configuring many partitions and setting the corresponding time limits may be difficult—when set incorrectly, system fragmentation and poor utilization of partitions may result. Fortunately, as we show in our analysis, having only two queues (and partitions) already significantly reduces the job slowdown variability and is sufficient to reach our goal. Thus, in our simulations we set for all policies $K = 2$. In this case, for FBQ, only one parameter has to be set, which is the time limit of queue 1. In contrast with the previous implementations in single-server and distributed-server systems, our TAGS and SITA policies require an additional parameter to be set for each queue, which is the partition capacity—in our case only the capacity of partition 1. We don't have to consider COMP here as it operates without partitioning and without time limits. We seek the optimal values of the parameters (capacity and/or queue time limit of partition 1) for which our policies achieve the lowest job slowdown variability.

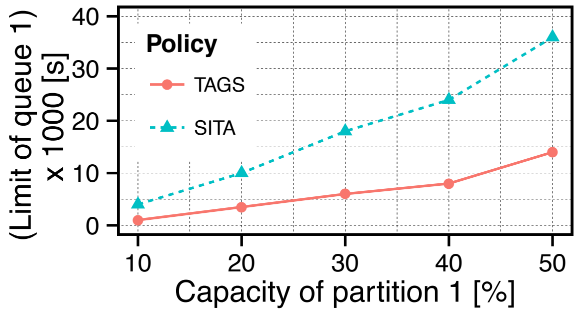
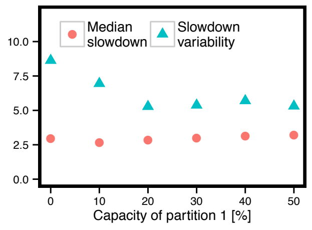
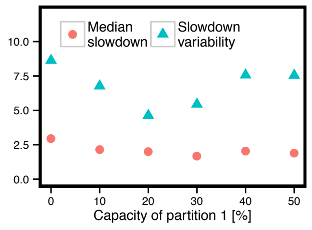


Fig. 5: The optimal time limit of partition 1 for each capacity between 10-50% at a system load of 0.7.



(a) TAGS, $\rho = 0.7$

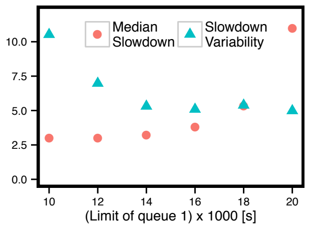


(b) SITA, $\rho = 0.7$

Fig. 6: The median job slowdown and the job slowdown variability versus the capacity of partition 1 under a system load of 0.7 (capacity 0% corresponds to FIFO).

We will first investigate the relation between the partition size and the queue time limit for TAGS and SITA. To this end, we show in Figure 5 the optimal time limit of partition 1 for a range of sizes of partition 1. Obviously, SITA has a higher time limit of partition 1 than TAGS, as jobs with SITA run to completion. Therefore, we expect TAGS to operate well even at high capacities of partition 1, but we want SITA to utilise a smaller partition 1.

In Figure 6 we show how TAGS and SITA actually perform in terms of the job slowdown when the capacity allocated to the first partition varies between 0-50% and the queue time limit for each size of partition 1 is set to the optimal value as indicated by Figure 5. As a hint to reading this and later similar figures, the values at 0% capacity of partition 1 should be interpreted as having a 95th percentile of the job slowdown distribution


(a) TAGS, $\rho = 0.7$

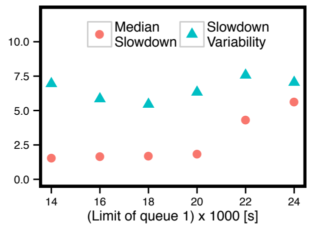
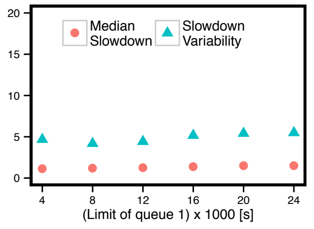

(b) SITA, $\rho = 0.7$

Fig. 7: The median job slowdown and the job slowdown variability versus the time limit in queue 1 at a system load of 0.7 (the horizontal axes have different scales). The size of partition 1 is set to 50% (TAGS) and 30% (SITA).


(a) FBQ, $\rho = 0.7$

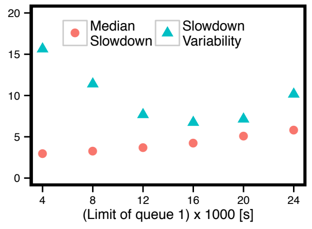
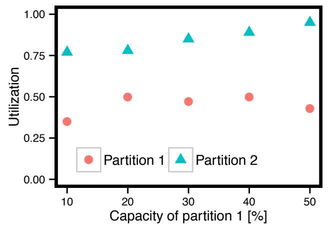

(b) FBQ, $\rho = 0.9$

Fig. 8: The median job slowdown and the job slowdown variability versus the time limit in queue 1 at a system load of 0.7 and 0.9.

of about 23.8 (2.8 x 8.5). We observe that the impact of the partition size is relatively small with TAGS over a wide range of sizes—a capacity of partition 1 ranging from 20% to 50% is fine. In contrast, with SITA, setting the partition sizes is much more critical; the job slowdown is significantly better when the capacity of partition 1 is 20% or 30%, depending on whether the median or the job slowdown variability is considered more important. Outside that range, the job slowdown variability is much higher.

Finally for the TAGS and SITA policies, we investigate the setting of the time limit of queue 1 having the capacity of partition 1 set to 50% and 30%, respectively. Figure 7 depicts the slowdown statistics for large ranges of queue time limits. TAGS has relatively low median and high variability at low queue time limits, and the other way around at high limits. SITA is relatively stable in the range of 14-20 x 1000 seconds, but for higher values of the time limit, the median value gets very poor. The time limits we consider to be the best for TAGS and SITA are 14,000 and 18,000 seconds, respectively (as already indicated in Figure 5 for the partition sizes considered here). Interestingly, the fraction of work that gets completed in partition 1 is 35% for both policies with these optimal cutoffs.

Further, we investigate the setting of the time limit of queue 1 for the FBQ policy. The slowdown statistics in Figure 8 show two important things. First, we see from Figure 8a that under a system load of 0.7, FBQ is very insensitive to the queue time limit, in contrast to both TAGS and SITA. However, in Figure 8b we show that going to 0.9 system load, FBQ becomes more sensitive. We analyse in more detail the performance of all policies under heavy traffic in Section VI-C. Secondly, Figure 8 shows that FBQ is by far the best performing policy with respect to both the median slowdown and the slowdown variability across the whole range of queue time limits considered. The time limit we consider to be the best for FBQ is 12,000 seconds.


(a) TAGS, $\rho = 0.7$

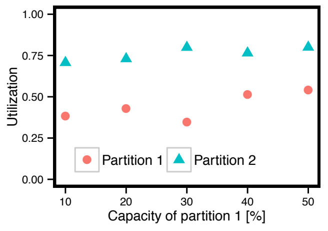

(b) SITA, $\rho = 0.7$

Fig. 9: The utilizations of partition 1 and 2 versus the capacity of partition 1 at a system load of 0.7.

B. *Load Unbalancing*

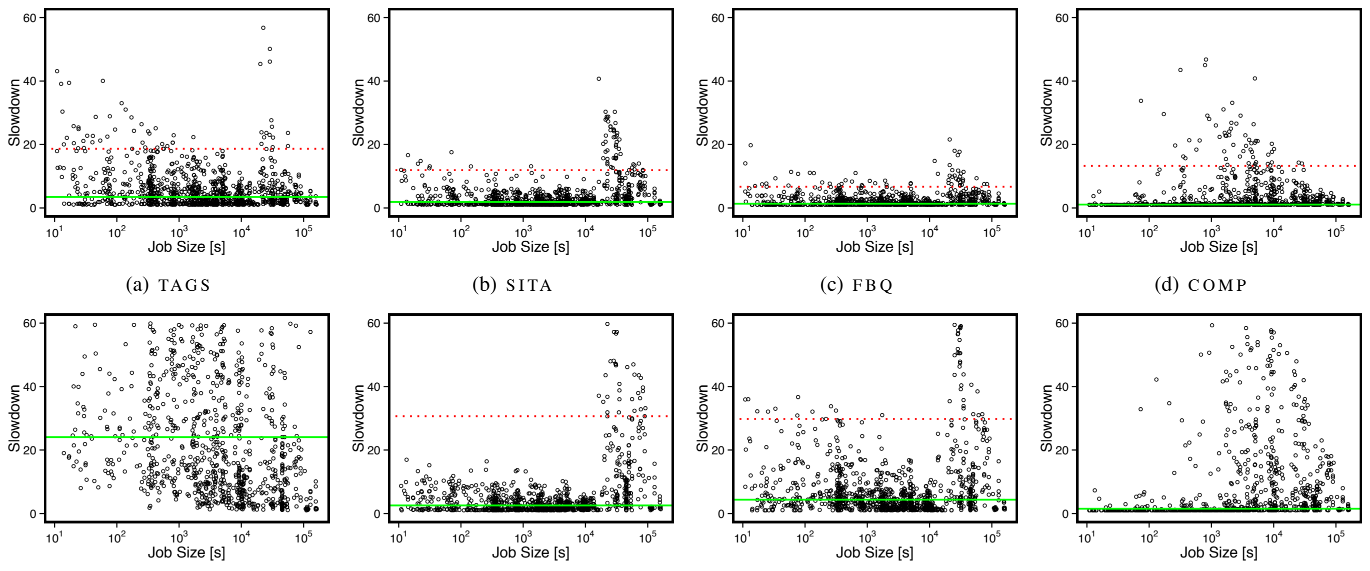
In previous studies of the TAGS and SITA policies for distributed server systems, it has been shown that choosing the queue time limits (or cutoffs) to balance the expected load across partitions can lead to suboptimal median slowdown [8], [11]. Counterintuitively, load unbalancing optimizes fairness when the workload has a heavy-tailed distribution. The intuition behind this strategy is that a large majority of the jobs get to run at a reduced load, thus reducing the median slowdown.

We investigate now whether load unbalancing has the same effect in our MapReduce use case. Figure 9 shows the utilizations of partitions 1 and 2 for different capacities of partition 1 with the corresponding optimal queue time limits as we have determined in Section VI-A. Indeed, we observe for TAGS and SITA that for any capacity of partition 1 in the range 10-50%, partition 1 is assigned significantly less load than partition 2.

Nevertheless, it seems that when we try to achieve both low median job slowdown and low job slowdown variability, only SITA is comparable to FBQ, while TAGS seems to be hitting a wall. The former two policies have very close slowdown variability when they operate at their optimal time limits – 18,000 seconds for SITA (see Figure 7b) and 12,000 seconds for FBQ (see Figure 8a). In contrast, in Figure 9 we show that TAGS has a very high utilisation in partition 2 at the optimal partition size of 50% and the optimal time limit of 14,000 seconds. This can be explained by what is the major difference between TAGS and SITA. Whereas SITA runs each job till completion in its "own" partition (wrong job size predictions can be made), TAGS moves jobs across multiple partitions through preemption until they reach the proper queues where they can complete. So SITA provides better isolation for short jobs than TAGS does. As a consequence, with TAGS the slowdowns of short jobs may be significantly higher than with SITA, as very large jobs with high levels of parallelism may monopolize the entire partition 1, no matter what its capacity is set to. Thus, TAGS with its optimal time limit of 14,000 s may achieve a lower job slowdown variability than SITA with its optimal time limit of 18,000 s, but it does so at the expense of a significantly higher median job slowdown.

C. *Heavy-Traffic Performance*

One major criticism of the TAGS and SITA policies in distributed server systems has been the strong dependence of their performance on the system load [11], [22]. As the load increases, unbalancing the load across the two servers



(a) TAGS (b) SITA (c) FBQ (d) COMP

(e) TAGS (f) SITA (g) FBQ (h) COMP

Fig. 10: Scatter plots of the job slowdown versus the job size for all policies under system loads of 0.7 (top) and 0.9 (bottom). The horizontal lines show the median and the 95th percentile of the job slowdown distribution. The horizontal lines that shows the 95th percentile of TAGS and COMP at system load of 0.9 are higher than 60.

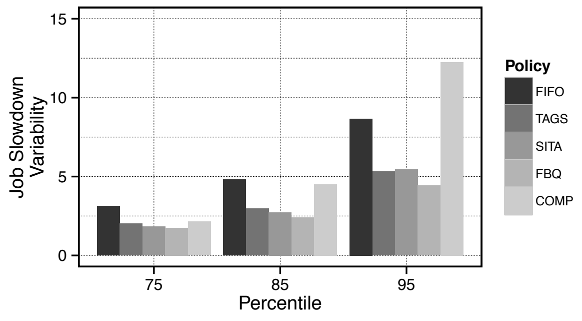


Fig. 11: The job slowdown variability at different percentiles.

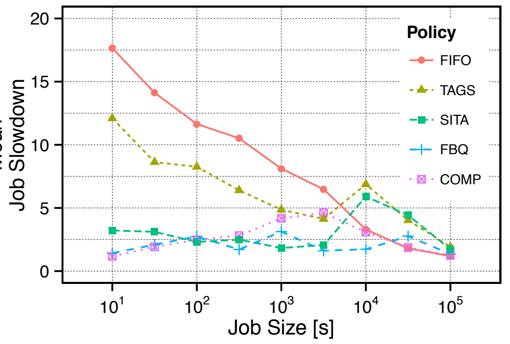


Fig. 12: The mean job slowdown per job-size subrange of the complete range of job sizes at a system load of 0.7 (horizontal axis in log scale).

(partitions) is difficult to achieve without overloading the second server (partition). In Figure 10 we show scatter plots of the job slowdown versus the job size for all policies with the optimal parameters. We observe that under system load of 0.9, the performance of TAGS is very poor, with the median job slowdown being twice as large as with FIFO. Further, with both TAGS and COMP the job slowdown at the 95th percentile is higher than 60. Despite having a relatively lower median job slowdown than TAGS and twice as low as FIFO, SITA hurts significantly the top 5% large jobs in our workload.

Although one would expect this issue to be removed with FBQ, it turns out this is not necessarily true for MapReduce. We observe that even for FBQ, balancing between the two optimisation goals, i.e., the median job slowdown and the job slowdown variability, is difficult, especially when the system is under heavy-traffic. To explain this, we have to look at the internal structure of MapReduce. As we have explained in our model, when jobs are being preempted, they are allowed to finish their running tasks no matter their given priority at that time. This has a negative effect on both the system utilisation and the slowdowns of large jobs, which may have reduce tasks unnecessarily occupying slots, while their map tasks are waiting for higher-priority queues to become empty.

D. Fairness Analysis

Recent mathematical analysis has shown that scheduling policies that are biased against long jobs by giving priority to relatively small jobs, are not only optimal with respect to mean response time or mean slowdown, but are also fair [4], [14], [15]. In addition, as MapReduce was originally created for jobs processing terabytes of data, scheduling disciplines that allow short jobs to preempt large ones have never been used in practice for MapReduce workloads for fear of hurting the very large jobs. In this section we will present summary evidence to what extent our policies help in the two dimensions of job slowdown variability: reduced job slowdown variability (the ratio of the 95th and 50th percentiles of the job slowdown distribution) and even slowdowns across the complete range of job sizes.

In Figure 11 we show the job slowdown variability at different percentiles (as defined in Section III) under a system load of 0.7. COMP provides gains over FIFO only up to the 95^{th} percentile. It is obvious that here FBQ is superior to TAGS, SITA, and that all these policies achieve a significant improvement over FIFO. FBQ consistently improves the slowdown variability at all percentiles by a factor of 2 over FIFO. Although TAGS and SITA are very close to FBQ, they seem to shift the slowdown variability to the second partition, where the largest jobs with thousands of tasks experience slowdowns because of being confined to smaller partitions. In fact, it is not the truly large jobs that suffer, but those jobs that are not sufficiently small to be completed in partition 1 and therefore end up in partition 2.

We next compare how the mean job slowdown varies across different job sizes with each policy. In Figure 12, we show the mean slowdowns of jobs in logarithmically spaced subranges of the complete range of job sizes (so, the first job-size range covers sizes between 10 and $10\sqrt{10}$, the second between $10\sqrt{10}$ and 100, etc.) at a utilization of 0.7. The mean slowdowns with FBQ, SITA, and COMP are considerably lower and less variable than with TAGS and FIFO, with FBQ being somewhat more stable. Thus, in particular FBQ manages to achieve a very even job slowdown across the complete job size range, which is exactly what we wanted.

E. More than two queues

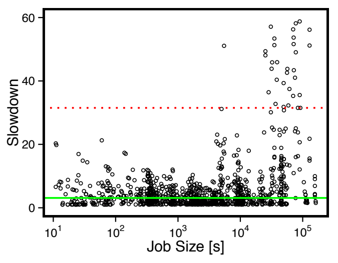
When configured with $K = 2$ queues, both FBQ and SITA consistently improve the job slowdown at the 95^{th} percentile by a factor of 2 over FIFO for system loads between 0.7 and 0.9. However, under system loads of 0.9 or higher, a relatively small fraction of large jobs (less than 5%) experience considerably larger slowdowns than the median. Therefore, we want to assess whether having more queues may reduce even more the job slowdown variability. As for SITA, an additional queue comes with the burden of partitioning and assigning appropriate capacities to those partitions, we compare only FBQ and COMP.

We will first explain the way we set the time limits of the queues for FBQ. Recall from Figure 10g that the job slowdowns offered by FBQ in optimal setting for $K = 2$ have relatively different values across three ranges of job sizes. The job slowdowns are somewhat stable for job sizes between 4,000 seconds and 10,000 seconds, but are very variable for job sizes outside that range. Thus, we set the time limits of queue 1 and queue 2 to 4,000 seconds and 10,000 seconds, respectively. As for queue 3, we set the time limit to 36,000 seconds, which is size of the job that suffers the highest job slowdown in the optimal setting for $K = 2$. Even though there may be other better values, setting the time limits is very critical for FBQ with multiple queues at system loads of 0.9 or higher.

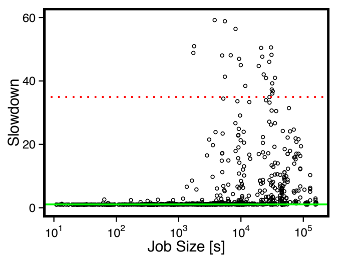
In Figure 13 we show scatter plots of the job slowdown versus the job size for FBQ and COMP when the number of queues K is set to 4. Two points stand out. First, FBQ with $K = 4$ offers considerable gains when the system is under heavy load. The median slowdown improves by 30% and no job of relatively small size (below 4,000 seconds) is over the 95^{th} percentile of the job slowdown distribution. Secondly, in contrast with FBQ, COMP has lower median job slowdown and more variable job slowdowns for large job sizes.

VII. RELATED WORK

A large body of both theoretical and experimental work exists on the evaluation of task assignment policies that are biasing towards jobs with small sizes. However, despite



(a) FBQ



(b) COMP

Fig. 13: Scatter plots of the job slowdown versus the job size for FBQ and COMP with $K = 4$ queues under a system load of 0.9. The horizontal lines show the median and the 95^{th} percentile of the job slowdown distribution.

the superior performance of size-based policies, datacenter schedulers have not yet incorporated them, for fear of unfairly hurting very large jobs.

It is well-known that the Shortest-Remaining-Processing-Time (SRPT) discipline is the best policy with respect to mean sojourn time given any arrival sequence and job size distributions [21], [23]. Another policy with comparable performance to SRPT is the Shortest-Processing-Time-Product (SPTP), which serves jobs according to the minimum product of initial size and remaining service time. In an M/G/1 setting, it has been shown that SPTP is optimal with respect to mean slowdown [17].

In contrast to the general premise that SRPT may treat unfairly large jobs which may starve given an adversarial arrival sequence, recent work has shown that such concerns are not realistic in the average case. More specifically, it can be proved that for an M/G/1 model and workloads which are heavy-tailed, all jobs experience a lower response time under SRPT when compared to the more popular processor-sharing (PS) discipline [4], [15]. SRPT achieves similar performance even for general job size distributions under moderate loads, and is not prohibitive even under higher loads. As a follow-up to the previous result, it has been shown that in a more practical setting of a web server, SRPT-based scheduling is to be preferred over the de-facto FAIR scheduler which fairly allocates fractions of resources to handle incoming requests [14]. However, SRPT is rarely used in current schedulers because of the practical consideration of possibly unknown job sizes. Luckily, the former drawback of SRPT discipline can be alleviated in several ways. First, many computer workloads exhibit strong correlations between the job input size and the processing time (or size) of the job. This phenomenon has been noticed in the case of web server requests, and is valid for MapReduce workloads as well. Secondly, when job sizes cannot be anticipated in any way, SRPT may be approximated in an M/G/1 model by employing multi-level scheduling with a large (infinite) number of feedback queues [20].

One representative policy for the problem of task assignment for server farms is Size-Interval-Task-Interval-Assignment or SITA [12], which isolates all jobs with sizes within a predefined size interval to a single server. SITA was specifically designed to reduce variability at each queue by dividing the job size distribution so that each queue handles a less variable portion of the original distribution. Although it may seem that the optimal cutoff points in the distributions should be chosen in such a way that the load is balanced across different servers (of a server farm), counterintuitively, it has been proved that SITA is more effective at reducing the mean slowdown when exactly the opposite is done [16]. Thus, SITA unbalances the load across the hosts and allows smaller tasks to run at lighter-loaded servers. It has been shown that with SITA, the more heavy-tailed the workload is, the better is the load unbalanced across hosts, thus greatly improving the mean slowdown [8]. Further, as the job size variability increases, SITA is by far superior to the Least-Work-Left (LWL) policy, which sends jobs to the server with the least remaining work. Despite the general belief of being the better policy, it seems SITA may in fact be inferior to the previous greedy policy, for particular job size distributions [13]. Derived from the same fundamental idea of load unbalancing, TAGS achieves comparable performance to SITA in the more challenging case of unknown job sizes [11]. Closest to our work, there exists a thorough simulation-based analysis of SITA with respect to fairness under supercomputing workloads [22].

VIII. Conclusions

Reducing job slowdown variability is an attractive yet challenging target for MapReduce workloads with variable job size distributions. Towards this end, we have presented four multi-queue size-based scheduling policies for data-intensive workloads derived from previous solutions to this problem for sequential or rigid jobs in single-server and distributed-server systems. The basic mechanisms employed by our policies are partitioning of resources of the datacenter, and system feedback by means of preemption in a work-conserving way. Hence, jobs with different ranges are isolated in separate queues or partitions, either by means of preemption from one queue to another (the TAGS and FBQ policies), or through some form of prediction (the SITA and COMP policies).

We analyse these policies with an extensive set of realistic simulations of MapReduce workloads and show close to ideal improvement for the vast majority of short jobs even in unfavourable load conditions, while only a relatively small fraction of large jobs suffer (less than 5%). We show that TAGS operates well for capacities of partition 1 between 20-50%, and that SITA offers considerable better slowdown performance when the size of partition 1 is small. FBQ is comparable to SITA, but at both lower median slowdown and slowdown variability, and unlike SITA, it is very insensitive to the queue time limit. Under heavy load, TAGS and COMP are by far the worst performing policies, while FBQ and SITA hurt significantly only the top 5% largest jobs in our workload. However, FBQ with 4 queues achieves a 30% improvement in median slowdown over the 2-queue setting.

We find that FBQ consistently improves the slowdown variability over FIFO by a factor of 2 under system loads between 0.7 and 0.9. Thus, FBQ may be the best policy in practice, as it not only comes with the advantage of requiring the setting of fewer parameters than both TAGS and SITA, but also offers very even job slowdowns across the complete range of job sizes.

Finally, we deploy our policies on a multicluster system and show that the relative error between the simulations and the real-world experiments is less than 1% for both the median job slowdown and the job slowdown at the 95^{th} percentile.

References

[1] "The Distributed ASCI Supercomputer 4," http://www.cs.vu.nl/das4.

[2] "Hadoop Capacity Scheduler," http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.

[3] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming Stragglers in Approximation Analytics," *NSDI*, 2014.

[4] N. Bansal and M. Harchol-Balter, "Analysis of SRPT Scheduling: Investigating Unfairness," *SIGMETRICS*, 2001.

[5] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," *VLDB*, 2012.

[6] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance using Workload Suites," *MASCOTS*, 2011.

[7] E. Coffman, R. Muntz, and H. Trotter, "Waiting Time Distributions for Processor-Sharing Systems," *Journal of the ACM*, vol. 17, no. 1, 1970.

[8] M. E. Crovella, M. Harchol-Balter, and C. D. Murta, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load," *SIGMETRICS PER*, vol. 26, no. 1, 1998.

[9] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, vol. 51, no. 1, 2008.

[10] E. J. Friedman and S. G. Henderson, "Fairness and Efficiency in Web Server Protocols," *SIGMETRICS PER*, vol. 31, no. 1, 2003.

[11] M. Harchol-Balter, "Task Assignment with Unknown Duration," *Distributed Computing Systems*, 2000.

[12] M. Harchol-Balter, M. E. Crovella, and C. D. Murta, "On Choosing a Task Assignment Policy for a Distributed Server System," *JPDC*, 1999.

[13] M. Harchol-Balter, A. Scheller-Wolf, and A. R. Young, "Surprising Results on Task Assignment in Server Farms with High-Variability Workloads," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, 2009.

[14] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-Based Scheduling to Improve Web Performance," *TOCS*, vol. 21, no. 2, 2003.

[15] M. Harchol-Balter, K. Sigman, and A. Wierman, "Asymptotic Convergence of Scheduling Policies with Respect to Slowdown," *Performance Evaluation*, vol. 49, no. 1, 2002.

[16] M. Harchol-Balter and R. Vesilo, "To Balance or Unbalance Load in Size-Interval Task Allocation," *Probability in the Engineering and Informational Sciences*, vol. 24, no. 02, 2010.

[17] E. Hyytiä, S. Aalto, and A. Penttinen, "Minimizing Slowdown in Heterogeneous Size-Aware Dispatching Systems," *SIGMETRICS*, vol. 40, no. 1, 2012.

[18] P. R. Jelenkovic, X. Kang, and J. Tan, "Adaptive and Scalable Comparison Scheduling," *SIGMETRICS*, vol. 35, no. 1, 2007.

[19] A. Murthy, "Mumak: Map-Reduce Simulator," *MAPREDUCE-728, Apache JIRA*, 2009.

[20] L. Schrage, "The Queue M/G/1 with Feedback to Lower Priority Queues," *Management Science*, vol. 13, no. 7, 1967.

[21] L. E. Schrage and L. W. Miller, "The Queue M/G/1 with the Shortest Remaining Processing Time Discipline," *Operations Research*, vol. 14, no. 4, 1966.

[22] B. Schroeder and M. Harchol-Balter, "Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness," *Cluster Computing*, vol. 7, no. 2, 2004.

[23] D. R. Smith, "A New Proof of the Optimality of the Shortest Remaining Processing Time Discipline," *Operations Research*, vol. 26, no. 1, 1978.

[24] J. Tan, X. Meng, and L. Zhang, "Delay Tails in MapReduce Scheduling," *SIGMETRICS*, 2012.

[25] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," *ACM Autonomic computing*, 2011.

[26] ——, "Play It Again, SimMR!" *IEEE CLUSTER*, 2011.

[27] A. Wierman and M. Harchol-Balter, "Classifying Scheduling Policies with Respect to Unfairness in an M/GI/1," *SIGMETRICS*, 2003.

[28] R. W. Wolff, "Stochastic Modelling and the Theory of Queues," *Englewood Cliffs, NJ*, 1989.

[29] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *EuroSys*, 2010.