Improving Short Job Latency Performance in Hybrid Job Schedulers with Dice

Wei Zhou
University of Virginia
Charlottesville, USA
wz5ad@virginia.edu

K. Preston White
University of Virginia
Charlottesville, USA
kpwhite@virginia.edu

Hongfeng Yu
University of Nebraska-Lincoln
Lincoln, USA
yu@cse.unl.edu

ABSTRACT

It is common to find a mixture of both long batch jobs and latency-sensitive short jobs in enterprise data centers. Recently hybrid job schedulers emerge as attractive alternatives of conventional centralized job schedulers.

In this paper, we conduct trace-driven experiments to study the job-completion-delay performance of two representative hybrid job schedulers (Hawk and Eagle), and find that short jobs still encounter long latency issues due to fluctuating bursty nature of workloads. To this end, we propose Dice, a general performance optimization framework for hybrid job schedulers, to alleviate the high job-completion-delay problem of short jobs. Dice is composed of two simple yet effective techniques: Elastic Sizing and Opportunistic Preemption. Both Elastic Sizing and Opportunistic Preemption keep track of the task waiting times of short jobs. When the mean task waiting time of short jobs is high, Elastic Sizing dynamically and adaptively increases the short partition size to prioritize short jobs over long jobs. On the other hand, Opportunistic Preemption preempts resources from long tasks running in the general partition on demand, so as to mitigate the "head-of-line" blocking problem of short jobs.

We enhance the two schedulers with Dice and evaluate Dice performance improvement in our prototype implementation. Experiment results show that Dice achieves 50.9%, 54.5%, and 43.5% improvement on 50th-percentile, 75th-percentile, and 90th-percentile job completion delays of short jobs in Hawk respectively, as well as 33.2%, 74.1%, and 85.3% improvement on those in Eagle respectively under the Google trace, at low performance costs to long jobs.

CCS CONCEPTS

• **Software and its engineering → Scheduling**.

KEYWORDS

big data, job scheduling, resource management

ACM Reference Format:
Wei Zhou, K. Preston White, and Hongfeng Yu. 2019. Improving Short Job Latency Performance in Hybrid Job Schedulers with Dice. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337851

1 INTRODUCTION

Big data analytics workloads in large-scale enterprise data centers tend to be more and more intensive, complex, and diverse, as evident in latest studies of job traces collected from production environments [3, 10, 11, 24, 26]. We observe a mixture of both long jobs and latency-sensitive short jobs in enterprise data centers. The latency of short jobs, that is, job completion delay, matters to users because most (if not all) short jobs are user-facing interactive applications like ad-hoc queries for interactive data analysis or personalized search. This fact demands modern-day big data jobs schedulers to be able to schedule large number of different types of jobs with corresponding resource and latency requirements from a variety of analytics frameworks like Hadoop MapReduce [30], Dremel [22], Impala [19], Storm [28], Spark [31], and Flink [7] in timely fashion.

Centralized job schedulers, like YARN [29], Mesos [17], Omega[25], achieve high resource efficiency since they usually have a global view of cluster resource allocations and demands of batch jobs. However, their job scheduling delay becomes non-trivial when scheduling a great amount of latency-sensitive short jobs. Distributed job schedulers like Apollo [5] and Sparrow [23] successfully minimize job scheduling delay with parallel and independent scheduling decision-making of multiple schedulers but they fail to allocate resources efficiently.

Recognizing the mixed nature of long batch jobs and latency-sensitive short jobs, Hawk [16], Eagle [14], Phoenix [27] hybrid job schedulers, which in general consist of one centralized scheduler for long jobs and multiple distributed schedulers for short jobs, have emerged as promising alternatives of existing job schedulers for better latencies of short jobs and cluster utilization.

Although the existing hybrid job schedulers have attempted to address the high-latency issue of short jobs, our experiment studies show that they still cannot fully handle intermittent burstiness of workloads, which in turn causes short jobs to suffer from long latency issues (see Section 2.2 for more detail). To this end, we propose a general performance optimization approach to hybrid job schedulers, called "Dice", to mitigate the long job-completion-delay issue of short jobs with two simple yet effective techniques: Elastic Sizing and Opportunistic Preemption. Elastic Sizing and Opportunistic Preemption continuously monitor the task waiting time of short jobs. Elastic Sizing dynamically and adaptively increases the dedicated short partition size to prioritize short jobs over long jobs when the mean task waiting time of short jobs is detected high. On the other hand, Opportunistic Preemption preempts resources from long tasks running in the general partition on demand, so

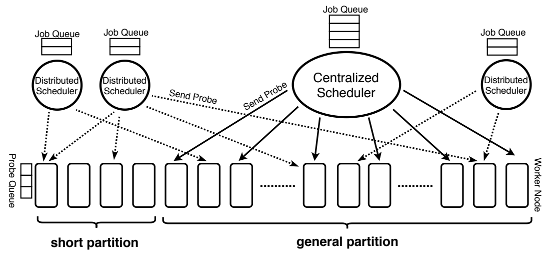


Figure 1: A general architecture of hybrid job schedulers

as to mitigate the "head-of-line" blocking problem of short jobs. Because the two schemes of Dice are orthogonal to the general architecture of existing hybrid job schedulers, Dice can be easily integrated into them. To the best of our knowledge, Dice is the first general performance optimization framework for hybrid job schedulers.

We enhance the Hawk and Eagle job schedulers with Dice in a widely-used trace-driven simulator, and evaluate the job-completion-delay performance improvement on short jobs by Dice with three representative traces of enterprise production workloads. Extensive experiment results show that Dice is able to significantly improve latencies of short jobs, at a relatively low cost by marginally increasing the latencies of long jobs. For example, under the Google trace, Dice achieves 50.9%, 54.5%, and 43.5% improvement on 50th-percentile (P50), 75th-percentile (P75), and 90th-percentile (P90) job completion delays of short jobs in Hawk respectively, as well as 33.2%, 74.1%, and 85.3% improvement on those metrics in Eagle respectively, while lengthening the P50 job completion delay of long jobs by 4.9% in Hawk and the P75 job completion delay of long jobs by 14.6% in Eagle respectively.

In summary, we make the following contributions in this paper:

- We identify that long task waiting time of short jobs for hybrid job schedulers is one main cause of the long job completion delay of shorts jobs under fluctuating bursty workloads;
- We propose the Elastic Sizing scheme to dynamically and adaptively increase the dedicated short partition size to prioritize short jobs over long jobs on demand;
- We propose the Opportunistic Preemption scheme to preempt resources from long tasks running in the general partition on demand, so as to mitigate "head-of-line" blocking problem of short jobs;
- We conduct extensive trace-driven experiments and validate the effectiveness of the two schemes individually and combinatorially.

2 BACKGROUND AND MOTIVATIONS

2.1 Background

Recently, hybrid job schedulers have been proposed to deliver both high cluster utilization and responsiveness of short jobs under mixed workloads in large-scale data centers. Hawk [16], Eagle [14], and Phoenix [27] are representative of state-of-the-art work in this field. Figure 1 illustrates a general architecture of hybrid job schedulers. As shown in the figure, a cluster consists of many worker nodes where each node maintains one probe queue of tasks. For hybrid job schedulers, the cluster is split into two exclusive partitions: *general partition* and *short partition*. The centralized job scheduler is meant to schedule only long jobs on the general partition, and the size of the general partition is determined by the portion of time cluster resources are spent on long jobs. The short partition is dedicated for executing short jobs, while multiple Sparrow [23]-like distributed schedulers are responsible for queuing and placing only short jobs on both partitions in parallel. Like Sparrow, each distributed scheduler uses the "batch sampling" scheme to place probes of short tasks onto the least loaded of randomly chosen worker nodes, and assigns short tasks to the worker nodes only until they are ready to run the tasks, which is called "late binding". Because of the high parallelism of job placement by distributed schedulers, short jobs are expected to bear very short job queueing delay.

To compensate occasional poor scheduling decisions by distributed schedulers, Hawk [16] employs "randomized task stealing", which steals tasks of short jobs behind running and waiting tasks of long jobs from busy nodes to idle nodes in the general partition. Eagle [14] proposes the "Succinct State Sharing" (SSS) mechanism to convey the information about locations where long jobs are executing among distributed schedulers, so as to avoid the head-of-line blocking for short jobs. Different from Sparrow, a probe in Eagle represents not one single task of a job, but the whole job. As a result, Eagle introduces "Sticky Batch Probing" (SBP) to keep the probe of a short job stay in the queue until all the remaining tasks of the corresponding job are completed. Phoenix [27] is an extension to the Eagle scheduler to take job placement constraints into considerations.

2.2 Motivations

As Hawk [16] is shown to improve the P50 and P90 job-completion-delay performance of short jobs by 80% and 90% respectively compared with Sparrow, and Eagle [14] further performs up to 80% better than Hawk, our question is raised: is short job-completion-delay performance good enough under latest hybrid job schedulers?

In order to understand performance behaviors of short jobs under hybrid job schedulers, we conduct a trace-driven experimental study with the open-source Eagle simulator, which is able to simulate both Hawk and Eagle schedulers [13]. In our experiments, we try to mimic the same configuration parameters used in Eagle. In particular, we simulate a cluster of 4000 nodes while 2% of nodes are reserved for the short partition because task-seconds of short jobs account for 2% overall task-seconds in the Yahoo trace [11]. Then we feed the Yahoo trace to the simulator as input workload. The Yahoo trace includes 24262 jobs in total where short jobs account for 90.6% (the jobs with the mean task runtime of smaller than 90.58 seconds are defined as short jobs for the Yahoo trace, that is "cutoff task runtime" to distinguish short and long jobs).

We are especially interested in understanding the impact of the task waiting time on job completion delay of short jobs. Therefore, for every 60-second time window in a simulation run, we first collect and report the ratio of job completion delay to mean task runtime of its corresponding short job. Considering an example case where there are two short jobs with 50-second and 5-second

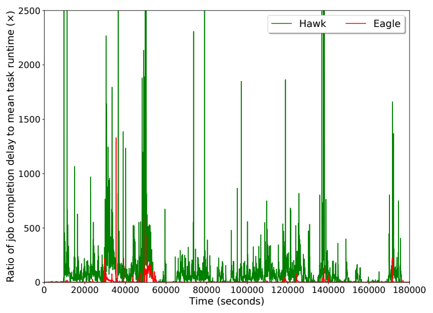


Figure 2: Ratio of job completion delay to mean task runtime for short jobs under the Yahoo trace

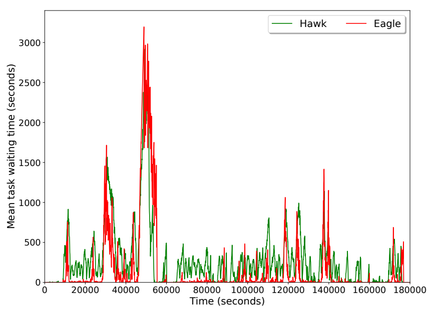


Figure 3: Mean task waiting time of short jobs under the Yahoo trace

mean task runtime respectively, the same job completion delay of 100 seconds may result in totally different user experience. Hence we believe this ratio, instead of the absolute value of job completion delay, is a better indicator of lags caused by resource contentions. Figure 2 illustrates the ratio of job completion delay to mean task runtime of short jobs for Hawk and Eagle schedulers under the Yahoo trace.

Second, we also collect and report the mean task waiting time of short jobs until all the jobs are completed for every 60-second time window. The task waiting time for a given task is defined as the duration from the time when its corresponding job is submitted to the time when the task is executed. In general, the task waiting time consists of task scheduling delay and probe queueing delay on the worker node. With multiple and parallel distributed schedulers dedicated for scheduling short jobs, task scheduling delay is guaranteed to be negligible. So the task waiting time of short jobs is actually determined by probe queueing delay. In case the task waiting time for a job outweighs mean task runtime, the task waiting time thus dominates job completion delay. Figure 3 plots the mean task waiting time of short jobs under Hawk and Eagle schedulers under the Yahoo trace.

From Figures 2 and 3, we have two observations: (1) spikes of the mean task waiting time are correlated and contribute to spikes of job completion delay for short jobs; and (2) spikes of the mean task waiting time can be as high as more than 3000 seconds, which is (3000/90.58 = 33.1) times cutoff task runtime for short jobs.

The above observations clearly dictate that shortening the task waiting time is key and imperative to improve job completion delay of short jobs. As we know for hybrid job schedulers a dedicated short partition is used to ensure low latency of short jobs, our first idea is to rethink the sizing of the short partition for better latency performance of short jobs. On the other hand, since short tasks could be affected by the head-of-line blocking in the general partition, our second idea is to explore the task preemption option.

3 ELASTIC SIZING

In this section, we first quantitatively evaluate the impact of the short partition size on job completion delay as well as cluster utilization. Then we discuss how to strike a good balance between job-completion-delay performance of short jobs and cluster utilization with Elastic Sizing.

3.1 Impact of Short Partition Size

Intuitively, a straightforward way to shorten the task waiting time and resulting job completion delay of short jobs is to increase the size of the dedicated short partition. Therefore, we evaluate job-completion-delay performance and cluster utilization as a function of different short partition sizes with the aforementioned simulator. Table 1 shows P50, P75, and P90 job completion delays of short and long jobs in both Hawk and Eagle schedulers with the short partition sizes of 2%, 4%, 6%, and 8% under the Yahoo trace. One can see that for Hawk, P50, P75, and P90 job completion delays of short jobs can be improved by 57.8%, 70.3%, and 77.9% respectively if the short partition size is increased from 2% to 8%. On the other hand, affected by fewer worker nodes available for long jobs, P50, P75, and P90 job completion delays of long jobs for the 8% short partition size are 48.0%, 30.0%, and 19.0% higher than those for 2% short partition size respectively. It clearly implies that a bigger size of the dedicated short partition contributes to significant performance improvement on job completion delay of short jobs, with a non-trivial cost to job completion delay of long jobs. We observe a similar pattern from experiment results for Eagle as well.

Let's then take a close look at cluster utilization. Figure 4 plots cluster utilization trends in Eagle under Yahoo trace with 2%, 4%, 6%, and 8% short partition sizes. It is clear that cluster utilization is inversely proportional to the short partition size within a certain range. In particular, for the 8% short partition size (that is, 92% general partition size), cluster utilization ranges from 92% to approximately 94% during most of the time, with sometimes 100% peaks. This is expected considering task-seconds of short jobs account for 2% overall task-seconds for the Yahoo trace. Therefore, a bigger size of the short partition could result in lower cluster utilization.

3.2 Elastic Sizing

Motivated by the above observations and implications, we propose *Elastic Sizing*, which dynamically and adaptively adjusts the short partition size according to the task waiting time of short jobs, so as to improve job-completion-delay performance of short

Table 1: Job completion delays under the Yahoo trace as a function of different short partition sizes

Short Partition	Job Completion Delay (seconds)					
	Short Jobs			Long Jobs		
Hawk	P50	P75	P90	P50	P75	P90
2%	327.7	687.1	1226.4	3145.2	6837.4	10089.1
4%	191.2	322.3	457.7	3455.5	7467.1	10754.7
6%	157.0	243.8	331.6	3947.6	8168.3	11447.7
8%	138.2	204.3	270.8	4654.6	8891.4	12010.3
Eagle	P50	P75	P90	P50	P75	P90
2%	29.8	59.1	147.3	3160.0	6837.7	10070.3
4%	23.2	42.0	66.0	3455.6	7469.1	10764.3
6%	22.4	40.4	62.9	3943.5	8166.6	11391.0
8%	22.1	39.9	62.1	4650.2	8901.9	12025.4

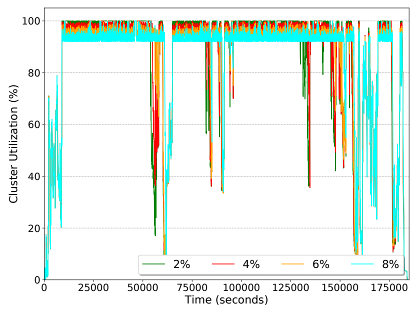


Figure 4: Cluster utilization in Eagle under the Yahoo trace

jobs, while minimizing adverse impacts on the performance of long jobs and overall cluster utilization. Elastic Sizing enforces sizing adjustment of the short partition by converting a number of the general-partition nodes into the short-partition nodes throughout consecutive time windows. In particular, the basic workflow of Elastic Sizing is as follows:

- At the start of a time window, the centralized scheduler of hybrid job schedulers collects the task waiting time of short jobs during last time window from all worker nodes and computes the mean task waiting time;
- The centralized scheduler then decides the number of nodes in the general partition should be temporally converted into the short-partition nodes during the current time window. Implementing node conversion is simple: the centralized scheduler puts the converted nodes onto a blacklist, and avoids scheduling probes of newly-arrived long jobs onto the blacklisted nodes during the current time window. Note that Elastic Sizing requires no changes to distributed schedulers;
- At the end of a time window, the centralized scheduler empties the blacklist.

We then discuss the algorithm to determine the number of nodes to convert. We define the lower bound of the short partition size as *MinShortPartitionSize* number of nodes, the upper bound of the

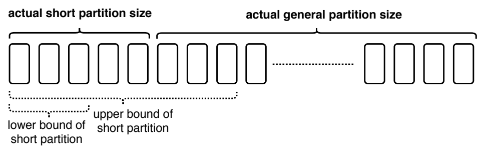

actual short partition size
actual general partition size
lower bound of short partition
upper bound of short partition

Figure 5: An illustrative example of Elastic Sizing

short partition size as *MaxShortPartitionSize* number of nodes. We also define the mean task waiting time during last time window as *CurrMeanTaskWaitingTime* and the corresponding maximum mean task waiting time of short jobs as *MaxTaskWaitingTime*. If *CurrMeanTaskWaitingTime* is greater than *MaxTaskWaitingTime*, (*MaxShortPartitionSize* − *MinShortPartitionSize*) general-partition nodes are converted into the short-partition nodes by Elastic Sizing. Otherwise, $p \times$ (*MaxShortPartitionSize* - *MinShortPartitionSize*) number of nodes will be converted, where $p \in [0.0, 1.0]$. Figure 5 illustrates an example case of Elastic Sizing.

We are interested in the relationship between the rate of node conversions and resulting performance gain. Therefore, we consider and evaluate the below 3 models to compute p because they reflect three different node-conversion strategies: linear conversion, slow-start conversion, and fast-start conversion respectively. This is achieved by leveraging the different responsiveness rate of functions $y = x$, $y = x^2$, and $y = \sqrt{x}$, where $x \in [0.0, 1.0]$.

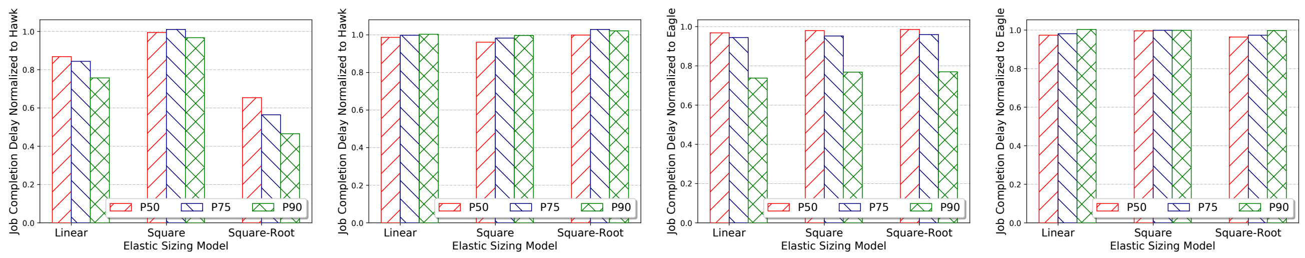
- Linear model: $p = \frac{CurrMeanTaskWaitingTime}{MaxTaskWaitingTime}$
- Square model: $p = (\frac{CurrMeanTaskWaitingTime}{MaxTaskWaitingTime})^2$
- Square-Root model: $p = \sqrt{\frac{CurrMeanTaskWaitingTime}{MaxTaskWaitingTime}}$

In summary, Elastic Sizing aims to prioritize short jobs over long jobs when short jobs face high task waiting time, by proactively constraining the number of nodes available for scheduling long jobs and thus allocating more resources to short jobs.

3.3 Searching Key Parameter Space

In this subsection, we first conduct experiments to understand the relationship between the performance impact and node conversion models. Secondly, we study the performance improvement by Elastic Sizing with different upper bounds of the short partition size.

In the first experiment, we configure the upper bound of the short partition size to 10% and the maximum mean task waiting time of short jobs to 1000 seconds. Then we run the simulations of Hawk and Eagle schedulers with different models in Elastic Sizing under the Yahoo trace. Figure 6 depicts the job-completion-delay performance in Hawk and Eagle schedulers enhanced with the three node-conversion models of Elastic Sizing. We can see from Figures 6a and 6b, Hawk with Elastic Sizing's Square-Root model is able to shorten P50, P75, and P90 job completion delays of short jobs by 34.6%, 43.6%, and 53.5% respectively compared with the original Hawk scheduler. However, this is achieved at the cost of 2.9% and 2.1% longer P75 and P90 job completion delays of long jobs respectively. It is expected because Square-Root model opts to aggressively convert the general-partition nodes into the short-partition nodes. As a counterpart, Square model responds



(a) Short jobs in Hawk **(b) Long jobs in Hawk** **(c) Short jobs in Eagle** **(d) Long jobs in Eagle**

Figure 6: Job completion delays for Elastic Sizing with different models normalized to Hawk and Eagle

to the increase of the task waiting time of short jobs so slowly that insufficient nodes are converted in time, which is evident in that trivial improvement on job completion delay performance of short jobs is observed for Square model. In contrast, Elastic Sizing's Linear model achieves 13.2%, 15.6%, and 24.3% improvement on P50, P75, and P90 job completion delays for short jobs respectively, with a negligible impact on job completion delay for long jobs. In the meantime, we observe from Figures 6c and 6d, Eagle is insensitive to different node-conversion models of Elastic Sizing and all the models are able to deliver more than 4% and 23% improvement on P75 and P90 job completion delays for short jobs.

In the second experiment, we vary the upper bounds of the short partition size from 4%, 6%, 8%, 10%, 12%, 14% to 16% with a step of 2% (Note that the lower bound of the short partition size is 2% under Yahoo trace by default), and use the Linear model. Figure 7 plots the job-completion-delay performance of Hawk and Eagle enhanced with Elastic Sizing as a function of different upper bounds of the short partition size. One can observe from Figures 7a and 7c: (1) Elastic Sizing with 16% upper bound of the short partition size improves P50, P75, and P90 job completion delay performance of short jobs by 18.3%, 22.3%, and 33.3% respectively for Hawk, and improves them by 2.9%, 5.3%, and 26.4% respectively for Eagle; (2) with the increase of the upper bound of the short partition size, Elastic Sizing is able to translate higher node conversion into lower job completion delays of short jobs; and (3) Hawk with Elastic Sizing is more sensitive to the upper bound of the short partition size than Eagle with Elastic Sizing.

4 OPPORTUNISTIC PREEMPTION

In this section, we first introduce the background of task preemption in the context of big data job scheduling. Then we present the basic idea of Opportunistic Preemption and explore its key parameter space.

4.1 Task Preemption

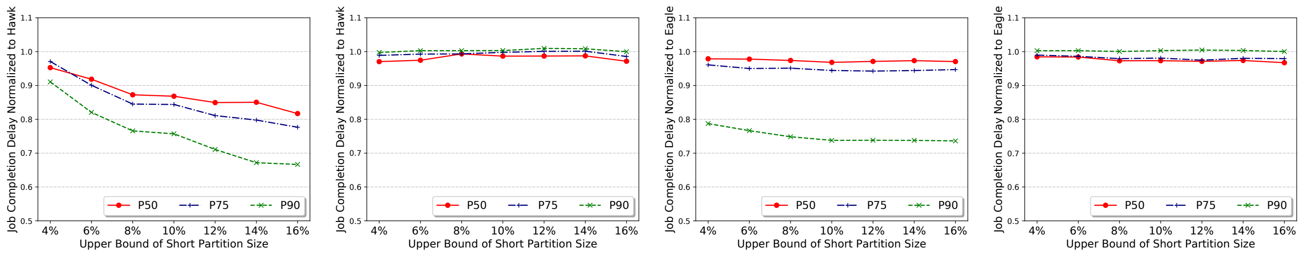
Process preemption is a commonly-used mechanism to enforce the time-slice quota of running processes and/or the prioritization of higher-priority processes over lower-priority processes in modern operating systems. Recently big data job schedulers have employed task preemption for fair resource sharing and job prioritization enforcement as well. Killing tasks is a simple but costly way to implement preemption because made progress of the tasks is lost and the killed tasks need to be restarted from scratch. On the other hand, job schedulers like Amoeba [4], Natjam [12], etc. checkpoint tasks' progress periodically to save intermediate results to persistent storage. This allows the tasks to be suspended and resumed when needed, which is usually called "checkpointing-based preemption" [20]. Thus whether to enable preemption in job scheduling is mainly determined by the efficiency and overhead of task suspension and resumption with checkpointing. Via lightweight container-based virtualization, Big-C [9] implements immediate preemption and graceful preemption strategies to make tasks preemptive with low cost and latency. Based on these two strategies, Big-C develops a preemptive fair share scheduler to preempt resources from long jobs when short jobs arrive. Inspired by Big-C, Kairos [15] implements time sharing on all worker nodes through container-based task preemption.

Although low latency of suspending and saving task context is achieved with container-based preemption under general workloads, Big-C's experiment results show that Spark tasks with iterative computation are susceptible to high resumption overhead. More importantly, latest studies [8, 21] show that Java Virtual Machine (JVM) warm-up overheads, e.g. class loading and byte-code interpretation, play an important role in short job execution while many popular data analytic frameworks including Hadoop [30] and Spark [31] are built upon JVM. For example, compute-intensive workloads like Spark queries could take 21 seconds on average on JVM warm-up, while IO-intensive workloads like HDFS reads could also spend 33% execution time on warm-up [21].

Without careful considerations of resumption and JVM warm-up overheads, blind preemption, even with low-cost container-based preemption scheme like Big-C, may result in both lengthened job completion delay of short jobs and low cluster utilization.

4.2 Opportunistic Preemption

Keeping benefits and possible overheads of task preemption in mind, we propose *Opportunistic Preemption*, which judiciously preempts resources of long tasks only when the task waiting time of short jobs is high. Different from Big-C that always preempts resources from long jobs when short jobs arrive to enforce share fairness and Kairos that always preempts resources from running jobs to enforce quota-based time sharing, Opportunistic Preemption aims to mitigate long task waiting time for short jobs with preemption on demand, while avoiding high resumption overheads of the above "always-on" preemption schemes.



(a) Short jobs in Hawk (b) Long jobs in Hawk (c) Short jobs in Eagle (d) Long jobs in Eagle

Figure 7: Job completion delays for Elastic Sizing with different upper bounds of the short partition size normalized to Hawk and Eagle

Similar to Elastic Sizing, the centralized scheduler with Opportunistic Preemption enabled periodically collects and aggregates the task waiting time of short jobs during last time window from all worker nodes. When the computed mean task waiting time of short jobs becomes high, Opportunistic Preemption is activated. The question is: how many and what long tasks should be preempted? Considering a data center consisting of tens of thousands of worker nodes and the centralized scheduler does not have detailed information about the running task and the probe queue for every worker node, Opportunistic Preemption computes the total number of needed preemption candidates according to the extent of the mean task waiting time of short jobs and then opts to randomly select worker nodes in the general partition. In particular, the basic workflow of Opportunistic Preemption is as follows:

- At the start of a time window, the centralized scheduler for hybrid job schedulers collects the task waiting time of short jobs during last time window from all worker nodes and computes the mean task waiting time;
- The centralized scheduler then computes the number of nodes in the general partition for preemption candidacy (say n), and sends preemption requests to randomly-chosen n nodes;
- For a node receiving a preemption request, it will save intermediate results of and then suspend the running task if the following four conditions are all met: (1) a long task is running on the node; (2) no other long tasks are suspended for the node; (3) the number of suspensions for the running task is less than a predefined maximum number of suspensions for a task under preemption; and (4) there are probes of short jobs waiting in the probe queue;
- When the task suspension is completed, the total number of suspensions for the task is incremented by one. A timer that is used to stop suspension with a predefined timeout starts ticking. In the meantime, the node will fetch the probe of a short task according to local scheduling algorithms like SRTF (Shortest Remaining Time First) in Eagle or FIFO (First In First Out) in Hawk, and execute the chosen short task;
- When the timer expires, the node will resume the suspended task after the currently running task is completed.

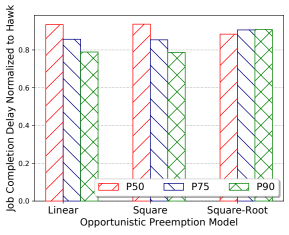
We then discuss the key parameter space of Opportunistic Preemption. Similar to Elastic Sizing, we define the current short partition size as *CurrShortPartitionSize* number of nodes, the mean task waiting time of short jobs during last time window as *CurrMeanTaskWaitingTime* and the corresponding maximum mean task waiting time of short jobs as *MaxTaskWaitingTime*. If *CurrMeanTaskWaitingTime* is greater than *MaxTaskWaitingTime*, Opportunistic Preemption sends preemption requests to *CurrShortPartitionSize* $\times$ *Multiplier* number of randomly-chosen nodes in the general partition. Otherwise, $p \times (\mathit{CurrShortPartitionSize} \times \mathit{Multiplier})$ preemption requests will be sent, where $p \in [0.0, 1.0]$. *Multiplier* is meant to compensate unfulfilled preemption requests due to the randomization nature of choosing preemption candidates. We consider the same Linear, Square, Square-Root models for p as Elastic Sizing.

In summary, Opportunistic Preemption aims to mitigate the head-of-line blocking issues caused by long tasks to lower long task waiting time of short jobs, with on-demand task preemption.

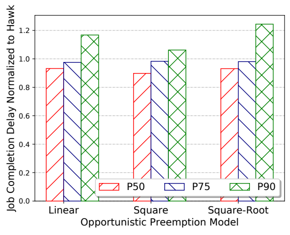
4.3 Searching Key Parameter Space

In the first experiment, we assume Opportunistic Preemption is also built on lightweight container-based virtualization, and configure task suspension delay to 3 seconds and task resumption delay to 10 seconds, as on par with experiment results in [8]. Then we run the simulations of Hawk and Eagle schedulers with different Opportunistic Preemption models under the Yahoo trace. We also configure the suspension duration to 100 seconds and the maximum number of allowed preemptions per task to 2. Figure 8 depicts the job-completion-delay performance of Hawk and Eagle schedulers with different Opportunistic Preemption models.

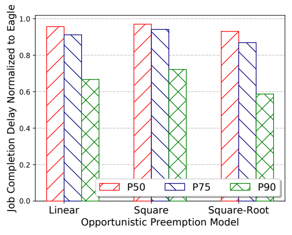
As shown in Figures 8a and 8c, Opportunistic Preemption is able to consistently improve job completion delays of short jobs under Hawk and Eagle schedulers. For example, Opportunistic Preemption with Square-Root model improves P90 job completion delay of short jobs under Eagle by up to 41.3% while Square model improves P90 job completion delay under Hawk by up to 21.3%. Moreover, experiment results also reveal different characteristics of Opportunistic Preemption compared with Elastic Sizing. First, Opportunistic Preemption is able to shorten job completion delay of short jobs under Eagle more significantly than Elastic Sizing. Second, Square-Root model is not always able to deliver the most performance gains for short jobs with Hawk (see Figure 8a) in spite of its aggressive nature, while it could cause the most performance loss for long jobs (20% as in Figure 8b). Third, Linear and Square



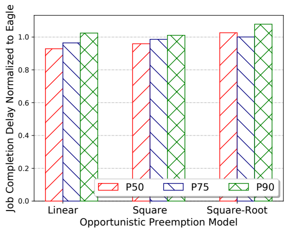
(a) Short jobs in Hawk



(b) Long jobs in Hawk

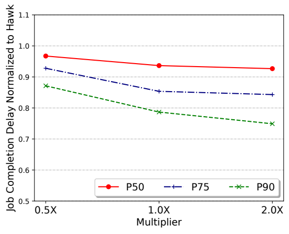


(c) Short jobs in Eagle

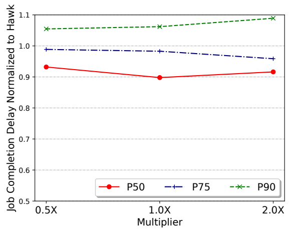


(d) Long jobs in Eagle

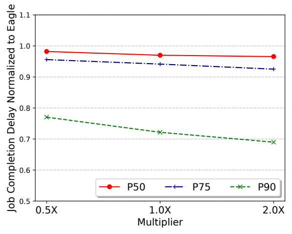
Figure 8: Job completion delays for Opportunistic Preemption with different models normalized to Hawk and Eagle



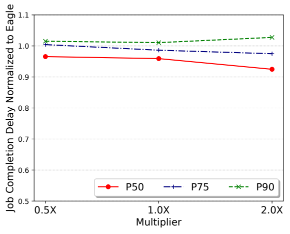
(a) Short jobs in Hawk



(b) Long jobs in Hawk



(c) Short jobs in Eagle



(d) Long jobs in Eagle

Figure 9: Job completion delays for Opportunistic Preemption with different multipliers normalized to Hawk and Eagle

models can achieve similar performance gains but Linear model tends to have more performance loss for long jobs.

The second experiment is to evaluate the impact of different multipliers. We vary the multiplier values from 0.5×, 1.0×, to 2.0×. Figure 9 depicts the job-completion-delay performance of Hawk and Eagle schedulers with different Opportunistic Preemption multipliers. As expected, with the increase of multiplier, we can see better improvement on the job-completion-delay performance of short jobs (especially under Eagle) as well as higher cost on job completion delay for long jobs (especially for Hawk). It suggests that Eagle needs an aggressive interference to alleviate long job completion delay of short jobs as its cluster resource is heavily utilized compared with Hawk. This implies that 1.0× multiplier strikes a balance between performance gains on short jobs and performance loss on long jobs.

5 PUTTING IT ALL TOGETHER: DICE

A natural idea is to enable both Elastic Sizing and Opportunistic Preemption, which becomes Dice. Dice leverages the commonality of monitoring the mean task waiting time of short jobs in both schemes and makes integration simple. Dice adds the logic into the centralized job scheduler to adjust the short partition size and preempt resources of long jobs when the mean task waiting time of short jobs is high during a time window. On the other hand, Dice recognizes the fact of conservative but nearly cost-free nature of Elastic Sizing and aggressive but potentially costly nature of Opportunistic Preemption, and supplements each other into one unified approach.

In essence, Dice introduces a feedback loop into hybrid job schedulers for performance optimizations. Latency awareness in Dice enables performance monitoring for short jobs, then actions are taken to activate the two proposed optimizations when performance is below expectation.

6 PERFORMANCE EVALUATIONS

6.1 Experimental Setup

We implement a Dice prototype and evaluate its performance in a trace-driven simulator, which is also used to evaluate Sparrow [23], Hawk [16], Eagle [14], Phoenix [27], and Kairos[15]. We feed the simulator with three traces representative of enterprise workloads in large data centers. In particular, we use the Yahoo and Cloudera traces [10, 11], as well as the Google trace [24, 26]. Table 2 shows the key job characteristics of the three traces, where the percentage of task-seconds of short jobs determines the lower bound of the short partition size.

In our experiments, we simulate clusters of 3000, 4000, and 5000 worker nodes using Yahoo trace and 11000, 12000, and 13000 worker nodes using Cloudera and Google traces to evaluate Dice performance under heavy, medium-heavy, and medium loads respectively. Each worker node has one single core and maintains one probe queue. Since job arrival timestamps are constant during trace replay in the simulations, varying the total number of worker nodes actually varies the workload intensity, and thus directly affects the task waiting time and job completion time.

We enhance Hawk and Eagle job schedulers with Dice, and compare P50, P75, and P90 job completion delay performance with the original Hawk and Eagle schedulers respectively. Every 60 seconds, Dice collects and aggregates the task waiting time of short jobs from worker nodes and computes the mean task waiting time. In Dice's configuration, Elastic Sizing employs Linear model and sets the upper bound of the short partition to 10%, 17%, and 25%

Table 2: Trace characteristics

Trace	Total jobs	Short jobs	Task-seconds of short jobs
Yahoo	24,262	90.6%	2%
Cloudera	21,030	95.0%	9%
Google	506,546	90.0%	17%

for Yahoo, Cloudera, Google traces respectively. This makes room for the short partition size adjustment be a fixed 8% of the cluster size. Opportunistic Preemption employs Square model and sets multiplier to 1.0×.

6.2 Results

Figure 10 plots normalized job-completion-delay performance for Hawk and Eagle schedulers enhanced with Dice under Yahoo trace as a function of varying numbers of nodes. From Figures 10a and 10c, we can see that Dice consistently and significantly improves the latencies of short jobs. First, for the 3,000-node cluster configuration, Dice achieves 91.4%, 83.6%, and 74.4% improvement on P50, P75, and P90 job completion delays of short jobs in Hawk respectively, as well as 97.4%, 82.3%, and 74.9% improvement on those in Eagle respectively. The reason for such a surprisingly significant improvement is that, the task waiting time of short jobs is kept extremely high under saturated load, and thus Dice has to activate both Elastic Sizing and Opportunistic Preemption during the simulation. In other words, high latency of short jobs dictates Dice to maximize the short partition size and proactively preempts resources from long tasks under heavy load to shorten job completion delay of short jobs. On the other hand, we can also see that P50 job completion delay of long jobs is lengthened by 14.3% in Hawk and 8.0% in Eagle. We believe, prioritizing short jobs over long jobs in Dice is imperative to guarantee responsiveness of short jobs under heavy or even saturated loads.

Second, under medium-heavy load (4000 nodes), Dice achieves 17.7%, 26.0%, and 38.3% improvement on P50, P75, and P90 job completion delays of short jobs in Hawk respectively, as well as 4.1%, 8.7%, and 34.0% improvement on those in Eagle respectively. We can also see that 8.9%/2.4% longer P90 job completion delay of long jobs in Hawk/Eagle respectively is traded for such performance improvement for short jobs.

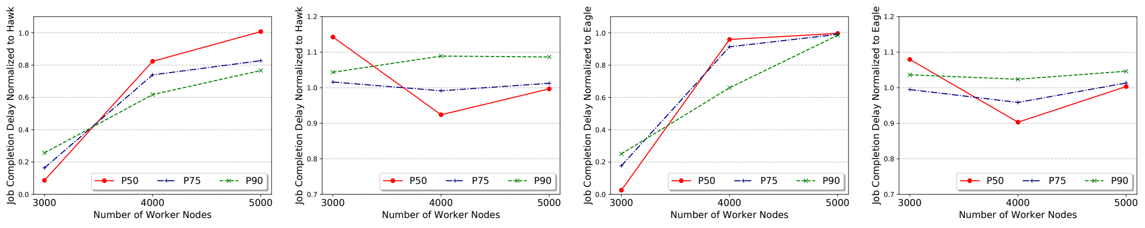
Third, it is clear that performance gains for short jobs decrease with the increase of cluster size. This is expected that lower workload intensity as a result of the bigger scale of a cluster reduces the chances of high task waiting time of short jobs, and Dice has fewer chances to activate Elastic Sizing and Opportunistic Preemption.

We can observe the same patterns from experiment results under the Cloudera and Google traces, as shown in Figures 11 and 12. More specifically, for a cluster of 12000 nodes under the Cloudera trace, Dice achieves 59.1%, 45.0%, and 14.4% improvement on P50, P75, and P90 job completion delays of short jobs in Hawk respectively, as well as 27.6%, 57.3%, and 11.5% improvement on those in Eagle respectively, while lengthening P90 job completion delay of long jobs by 26.8% in Hawk and 24.7% in Eagle respectively. For a cluster of 12000 nodes under the Google trace, Dice achieves 50.9%, 54.5%, and 43.5% improvement on P50, P75, and P90 job completion delays of short jobs in Hawk respectively, as well as 33.2%, 74.1%, and 85.3% improvement on those in Eagle respectively, while lengthening P50 job completion delay of long jobs by 4.9% in Hawk and P75 job completion delay of long jobs by 14.6% in Eagle respectively. Overall, we confirm that in Dice performance gains obtained for short jobs outweigh performance costs on long jobs.

In order to quantitatively evaluate how Elastic Sizing and Opportunistic Preemption individually and combinatorially contribute to performance gains, we conduct experiments of Hawk and Eagle schedulers enhanced with Elastic Sizing, Opportunistic Preemption, Dice under the three traces. Figure 13 compares their performance in terms of job completion delay of short jobs. One can see from Figure 13a, in the case of Hawk under the Yahoo trace, Dice achieves 5.2%, 12.4%, and 18.5% improvement on P50, P75, and P90 job completion delay of short jobs respectively compared with Elastic Sizing, as well as 12.1%, 13.4%, and 21.5% improvement respectively compared with Opportunistic Preemption. Similarly, we can also see from Figure 13d, in the case of Hawk under the Cloudera trace, Dice achieves 3.5%, 0.7%, and −1.4% improvement on P50, P75, and P90 job completion delays of short jobs respectively compared with Elastic Sizing, as well as 55.0%, 38.5%, and 15.3% improvement respectively compared with Opportunistic Preemption. It clearly indicates that using Elastic Sizing and Opportunistic Preemption combinatorially is able to deliver more performance gains than using them individually in most cases. We can observe the same pattern from experiment results for the Eagle scheduler and under the Google trace. Even in some cases, the combination of Elastic Sizing and Opportunistic Preemption causes performance loss, the performance loss is negligible (the maximum performance loss we observed from experiment results is −1.4%). This implies that it is empirically beneficial to deploy both Elastic Sizing and Opportunistic Preemption in hybrid job schedulers.

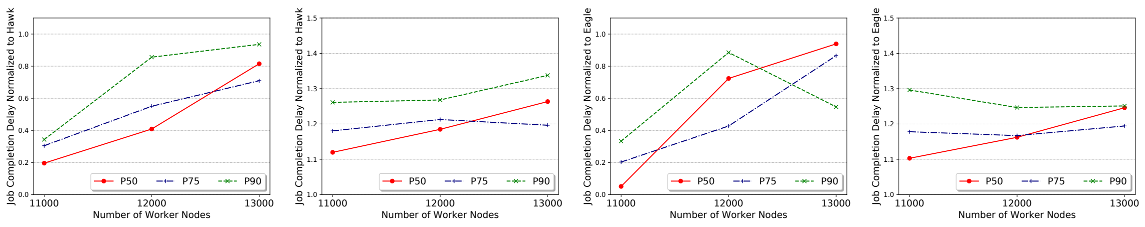
7 EXTENDED RELATED WORK

Elastic Sizing. In the context of resource management and job scheduling, the most similar work to cluster partitioning in hybrid job schedulers is the node label scheme [2] in YARN [29] and the floating partition scheme in Slurm [1, 18]. Each node in a cluster managed by YARN can be tagged with a label representing ownership or capacity (e.g., GPU support, memory size) and a group of nodes with the same label form a sub-cluster. A label can be set as *exclusive* or *non-exclusive*. A sub-cluster labeled as exclusive can run jobs with the same label only. In a sub-cluster labeled as non-exclusive, resources can be shared with any jobs in the cluster when idle resources are available. Similarly, a cluster in Slurm can be partitioned into disjoint sub-clusters as well, and Slurm's floating partition scheme can be used to share idle resources across partitions. Node label and floating partition are a static partitioning approach, and they are meant to share and leverage idle resources. In contrast, Elastic Sizing is a dynamic partitioning and node conversion approach, which dynamically and adaptively adjusts the short and general partition sizes according to the task waiting time of short jobs, even there are no idle resources. In addition, the basic idea of Elastic Sizing can be easily extended to YARN and Slurm.



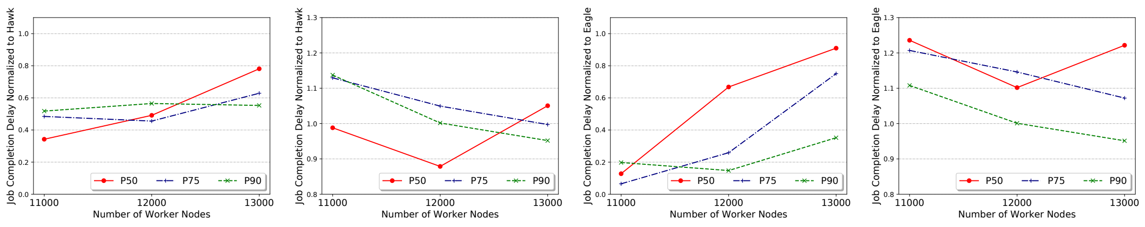
(a) Short jobs in Hawk (b) Long jobs in Hawk (c) Short jobs in Eagle (d) Long jobs in Eagle

Figure 10: Job completion delays for Dice normalized to Hawk and Eagle under Yahoo trace



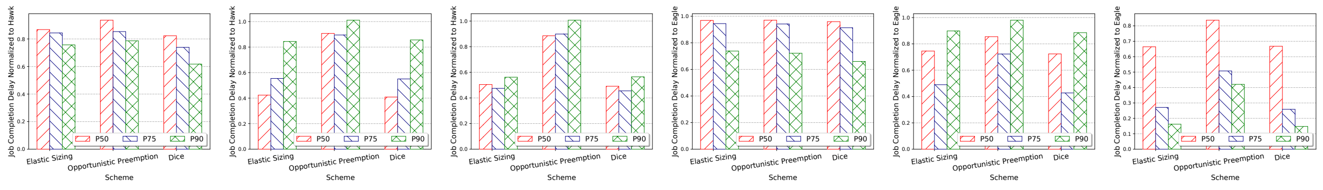
(a) Short jobs in Hawk (b) Long jobs in Hawk (c) Short jobs in Eagle (d) Long jobs in Eagle

Figure 11: Job completion delays for Dice normalized to Hawk and Eagle under Cloudera trace



(a) Short jobs in Hawk (b) Long jobs in Hawk (c) Short jobs in Eagle (d) Long jobs in Eagle

Figure 12: Job completion delays for Dice normalized to Hawk and Eagle under Google trace



(a) Short jobs in Hawk under Yahoo (b) Short jobs in Hawk under Cloudera (c) Short jobs in Hawk under Google (d) Short jobs in Eagle under Yahoo (e) Short jobs in Eagle under Cloudera (f) Short jobs in Eagle under Google

Figure 13: Job completion delays for Elastic Sizing, Opportunistic Preemption, and Dice normalized to Hawk and Eagle

Opportunistic Preemption. Task preemption has been widely used in major job schedulers and cluster managers like YARN [29], Mesos [17], Kubernetes [6], Slurm [1] for enforcement of job prioritization. For node label in YARN, a labeled job requesting labeled resources can preempt non-labeled jobs on labeled nodes. Big-C [9] implements a low-overhead task preemption mechanism via the container technique and then develops a preemptive job scheduler to prioritize short jobs over long jobs. Built on top of the container-based task preemption in Big-C, Kairos [15] proposes a two-layer scheduling framework to address head-of-line blocking problem of short jobs: one centralized scheduler for coarse-grained load balancing and local scheduler on every node for achieving Least Attained Service (LAS). In particular, local scheduler preempts the running task that has the highest LAS time when a new task arrives. When the time quota of the running task expires, local scheduler suspends the task and resumes a task with the least LAS time waiting in the task queue. Different from the always-on task preemption in Big-C and Kairos, Opportunistic Preemption is activated on demand, when the mean task waiting time of short jobs is high, to avoid the non-trivial task suspension and resumption overheads and JVM warmup overhead due to preemption.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose Dice, a general performance optimization approach for state-of-the-art hybrid job schedulers to address the high-latency issue of short jobs due to the bursty nature of workloads in large-scale enterprise data centers. Dice keeps track of the task waiting time of short jobs, and deploys Elastic Sizing and Opportunistic Preemption schemes to optimize job completion delays of short jobs. In particular, Elastic Sizing dynamically and adaptively increases the short partition size to accommodate long task waiting time of short jobs due to the shortage of resource reservation for short jobs, while Opportunistic Preemption preempts resources from long tasks running in the general partition on demand, so as to mitigate the head-of-line blocking problem. Trace-driven experiments show that Dice is able to achieve significant performance gains for short jobs with acceptable performance costs to long jobs.

In the future, we plan to implement a prototype of Dice on the open-source Hadoop YARN job scheduler to evaluate Dice performance in real-world environments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive suggestions on this paper. This research has been partially supported by the National Science Foundation through grants IIS-1423487, ICER-1541043, and IIS-1652846. The contents do not necessarily reflect the views and policies of the funding agencies and do not mention of trade names or commercial products constitute endorsement or recommendation for use.

REFERENCES

[1] 2019. Slurm Workload Manager. http://slurm.schedmd.com/.
[2] 2019. YARN Node Labels. https://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site/NodeLabel.html.
[3] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the diversity of cluster workloads and its impact on research results. In *USENIX Annual Technical Conference (USENIX ATC '18)*.
[4] Ganesh Ananthanarayanan, Christopher Douglas, Raghu Ramakrishnan, Sriram Rao, and Ion Stoica. 2012. True Elasticity in Multi-Tenant Data-Intensive Compute Clusters. In *ACM Symposium on Cloud Computing (SoCC '12)*.
[5] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*.
[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *ACM Queue* 14 (2016).
[7] Paris Carbone, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2015).
[8] Chen Chen, Wei Wang, and Bo Li. 2018. Performance-Aware Fair Scheduling: Exploiting Demand Elasticity of Data Analytics Jobs. In *IEEE International Conference on Computer Communications (INFOCOM '18)*.
[9] Wei Chen, Jia Rao, and Xiaobo Zhou. 2017. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization. In *USENIX Annual Technical Conference (USENIX ATC '17)*.
[10] Yanpei Chen, Sara Alspaugh, and Randy Katz. 2012. Interactive Query Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. In *International Conference on Very Large Data Bases (VLDB '12)*.
[11] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. 2011. The Case for Evaluating MapReduce Performance Using Workload Suites. In *IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '11)*.
[12] Brian Cho, Muntasir Rahman, Tej Chajed, Indranil Gupta, Cristina Abad, Nathan Roberts, and Philbert Lin. 2013. Natjam: Design and Evaluation of Eviction Policies For Supporting Priorities and Deadlines in Mapreduce Clusters. In *ACM Symposium on Cloud Computing (SoCC '13)*.
[13] Pamela Delgado. 2017. Hawk/Eagle simulator. https://github.com/epfl-labos/eagle/tree/master/simulation.
[14] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. 2016. Job-aware Scheduling in Eagle: Divide and Stick to Your Probes. In *ACM Symposium on Cloud Computing (SoCC '16)*.
[15] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. 2018. Kairos: Preemptive Data Center Scheduling Without Runtime Estimates. In *ACM Symposium on Cloud Computing (SoCC '18)*.
[16] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *USENIX Annual Technical Conference (USENIX ATC '15)*.
[17] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *8th USENIX Conference on Networked Systems Design and Implementation (NSDI '11)*.
[18] Morris Jette and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *ClusterWorld Conference and Expo*.
[19] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, Ishaan Joshi, Lenni Kuff, Dileep Kumar, Alex Leblang, Nong Li, Ippokratis Pandis, Henry Robinson, David Rorke, Silvius Rus, John Russell, Dimitris Tsirogiannis, Skye Wanderman-Milne, and Michael Yoder. 2015. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *7th Biennial Conference on Innovative Data Systems Research (CIDR '15)*.
[20] Jack Li, Calton Pu, Yuan Chen, Vanish Talwar, and Dejan Milojicic. 2015. Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters. In *ACM/IFIP/USENIX Middleware Conference (Middleware '15)*.
[21] David Lion, Adrian Chiu, Hailong Sun, Xin Zhuang, Nikola Grcevski, and Ding Yuan. 2016. DonâĂŹt Get Caught in the Cold, Warm-up Your JVM: Understand and Eliminate JVM Warm-up Overhead in Data-Parallel Systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.
[22] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. In *36th International Conference on Very Large Data Bases (VLDB '10)*.
[23] Kay Ousterhout, PatrickWendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: Distributed, Low Latency Scheduling. In *24th ACM Symposium on Operating Systems Principles (SOSP '13)*.
[24] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *ACM Symposium on Cloud Computing (SoCC '12)*.
[25] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *ACM European Conference on Computer Systems (EuroSys '13)*.
[26] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. 2011. Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters. In *ACM Symposium on Cloud Computing (SoCC '11)*.
[27] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2017. Phoenix: A Constraint-aware Scheduler for Heterogeneous Datacenters. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS '17)*.
[28] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. 2014. Storm@twitter. In *ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*.
[29] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Gravesy, Jason Lowey, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *ACM Symposium on Cloud Computing (SoCC '13)*.
[30] Tom White. 2015. *Hadoop: The Definitive Guide, 4th Edition.* O'Reilly Media Inc.
[31] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*.