On the diversity of cluster workloads and its impact on research results

George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson
Carnegie Mellon University
Elisabeth Baseman, Nathan DeBardeleben
Los Alamos National Laboratory

Abstract

Six years ago, Google released an invaluable set of scheduler logs which has already been used in more than 450 publications. We find that the scarcity of other data sources, however, is leading researchers to overfit their work to Google's dataset characteristics. We demonstrate this overfitting by introducing four new traces from two private and two High Performance Computing (HPC) clusters. Our analysis shows that the private cluster workloads, consisting of data analytics jobs expected to be more closely related to the Google workload, display more similarity to the HPC cluster workloads. This observation suggests that additional traces should be considered when evaluating the generality of new research.

To aid the community in moving forward, we release the four analyzed traces, including: the longest publicly available trace spanning *all* 61 months of an HPC cluster's lifetime and a trace from a 300,000-core HPC cluster, the largest cluster with a publicly available trace. We present an analysis of the private and HPC cluster traces that spans job characteristics, workload heterogeneity, resource utilization, and failure rates. We contrast our findings with the Google trace characteristics and identify affected work in the literature. Finally, we demonstrate the importance of dataset plurality and diversity by evaluating the performance of a job runtime predictor using all four of our traces and the Google trace.

1 Introduction

Despite intense activity in the areas of cloud and job scheduling research, publicly available cluster workload datasets remain scarce. The three major dataset sources today are: the Google cluster trace [58] collected in 2011, the Parallel Workload Archive [19] of High Performance Computing (HPC) traces collected since 1993, and the SWIM traces released in 2011 [10]. Of these, the Google trace has been used in more than 450 publications making it the most popular trace by far. Unfortunately, this 29-day trace is often the only one used to evaluate new research. By contrasting its characteristics with newer traces from different environments, we have found that the Google trace alone is insufficient to accurately prove the generality of a new technique.

Our goal is to uncover overfitting of prior work to the characteristics of the Google trace. To achieve this, our first contribution is to introduce four new traces: two from the private cloud of Two Sigma, a hedge fund, and two from HPC clusters located at the Los Alamos National Laboratory (LANL). Our Two Sigma traces are the longest, non-academic private cluster traces to date, spanning 9 months and more than 3 million jobs. The two HPC traces we introduce are also unique. The first trace spans the entire 5-year lifetime of a general-purpose HPC cluster, making it the longest public trace to date, while also exhibiting shorter jobs than existing public HPC traces. The second trace originates from the 300,000-core current flagship supercomputer at LANL, making it the largest cluster with a public trace, to our knowledge. We introduce all four traces, and the environments where they were collected, in Section 2.

Our second contribution is an analysis examining the generality of workload characteristics derived from the Google trace, when our four new traces are considered. Overall, we find that the private Two Sigma cluster workloads display similar characteristics to HPC, despite consisting of data analytics jobs that more closely resemble the Google workload. Table 1 summarizes all our findings. For those characteristics where the Google workload is an outlier, we have surveyed the literature and list affected prior work. In total, we surveyed 450 papers that reference the Google trace study [41] to identify popular workload assumptions, and we constrast them to the Two Sigma and LANL workloads to detect violations. We group our findings into four categories: job characteristics (Section 3), workload heterogeneity (Section 4), resource utilization (Section 5), and failure analysis (Section 6).

Our findings suggest that evaluating new research using the Google trace alone is insufficient to guarantee generality. To aid the community in moving forward, our third contribution is to publicly release the four traces introduced and analyzed in this paper. We further present a case study on the importance of dataset plurality and diversity when evaluating new research. For our demonstration we use JVuPredict, the job runtime predictor of the JamaisVu scheduling system [51]. Originally, JVuPredict was evaluated using only the Google trace [51]. Evaluating its performance with our four new traces, however, helped us identify features that make it easier to detect related and recurring jobs with predictable behavior. This enabled us to quantify the importance of individual trace fields in runtime prediction.

Section	Characteristic	Google	Two Sigma	Mustang	OpenTrinity
Job Characteristics (§3)	Majority of jobs are small	✔	✘	✘	✘
	Majority of jobs are short	✔	✘	✘	✘
Workload Heterogeneity (§4)	Diurnal patterns in job submissions	✘	✔	✔	✔
	High job submission rate	✔	✔	✘	✘
Resource Utilization (§5)	Resource over-commitment	✔	✘	✘	✘
	Sub-second job inter-arrival periods	✔	✔	✔	✔
	User request variability	✘	✔	✔	✔
Failure Analysis (§6)	High fraction of unsuccessful job outcomes	✔	✔	✘	✔
	Jobs with unsuccessful outcomes consume significant fraction of resources	✔	✔	✘	✘
	Longer/larger jobs often terminate unsuccessfully	✔	✘	✘	✘

Table 1: Summary of the characteristics of each trace. Note that the Google workload appears to be an outlier.

We describe our findings in Section 7.

Finally, we briefly discuss the importance of trace length in accurately representing a cluster's workload in Section 8. We list related work studying cluster traces in Section 9, before concluding.

2 Dataset information

We introduce four sets of job scheduler logs that were collected from a general-purpose cluster and a cutting-edge supercomputer at LANL, and across two clusters of Two Sigma, a hedge fund. The following subsections describe each dataset in more detail, and the hardware configuration of each cluster is shown in Table 2.

Users typically interact with the cluster scheduler by submitting commands that spawn multiple processes, or *tasks*, distributed across cluster nodes to perform a specific computation. Each such command is considered to be a *job* and users often compose scripts that generate more complex, multi-job schedules. In HPC clusters, where resources are allocated at the granularity of physical nodes similar to Emulab [4, 16, 27, 57], tasks from different jobs are never scheduled on the same node. This is not necessarily true in private clusters like Two Sigma.

2.1 Two Sigma clusters

The private workload traces we introduce originate from two datacenters of Two Sigma, a hedge fund firm. The workload consists of data analytics jobs processing financial data. A fraction of these jobs are handled by a Spark [49] installation, while the rest are serviced by home-grown data analytics frameworks. The dataset spans 9 months of the two datacenters' operation starting in January 2016, covering a total of 1313 identical compute nodes with 31512 CPU cores and 328TB RAM. The logs contain 3.2 million jobs and 78.5 million tasks, collected by an internally-developed job scheduler running on top of Mesos [28]. Because both datacenters experience the same workload and consist of homogeneous nodes, we collectively refer to both data sources as the *TwoSigma* trace, and analyze them together.

Platform	Nodes	CPUs	RAM	Length
LANL Trinity	9408	32	128GB	3 months
LANL Mustang	1600	24	64GB	5 years
TwoSigma A	872	24	256GB	9 months
TwoSigma B	441	24	256GB	
Google B	6732	0.50*	0.50*	29 days
Google B	3863	0.50*	0.25*	
Google B	1001	0.50*	0.75*	
Google C	795	1.00*	1.00*	
Google A	126	0.25*	0.25*	
Google B	52	0.50*	0.12*	
Google B	5	0.50*	0.03*	
Google B	5	0.50*	0.97*	
Google C	3	1.00*	0.50*	
Google B	1	0.50*	0.06*	

Table 2: Hardware characteristics of the clusters analyzed in this paper. For the Google trace [41], (*) signifies a resource has been normalized to the largest node.

We expect this workload to resemble the Google cluster more closely than the HPC clusters, where long-running, compute-intensive, and tightly-coupled scientific jobs are the norm. First, unlike LANL, job runtime is not budgeted strictly; users of the hedge fund clusters do not have to specify a time limit when submitting a job. Second, users can allocate individual cores, as opposed to entire physical nodes allocated at LANL. Collected data include: timestamps for job stages from submission to termination, job properties such as size and owner, and the job's return status.

2.2 LANL Mustang cluster

Mustang was an HPC cluster used for *capacity computing* at LANL from 2011 to 2016. Capacity clusters such as Mustang are architected as cost-effective, general-purpose resources for a large number of users. Mustang was largely used by scientists, engineers, and software developers at LANL and it was allocated to these users at the granularity of physical nodes. The cluster consisted of 1600 identical compute nodes, with a total of 38400 AMD Opteron 6176 2.3GHz cores and 102TB RAM.

Our Mustang dataset covers the entire 61 months of the machine's operation from October 2011 to November 2016, which makes this the longest publicly available cluster trace to date. The Mustang trace is also unique because its jobs are shorter than those in existing HPC traces. Overall, it consists of 2.1 million multi-node jobs submitted by 565 users and collected by SLURM [45], an open-source cluster resource manager. The fields available in the trace are similar to those in the TwoSigma trace, with the addition of a time budget field per job, that if exceeded causes the job to be killed.

2.3 LANL Trinity supercomputer

In 2018, Trinity is the largest supercomputer at LANL and it is used for *capability computing*. Capability clusters are a large-scale, high-demand resource introducing novel hardware technologies that aid in achieving crucial computing milestones, such as higher-resolution climate and astrophysics models. Trinity's hardware was stood up in two pre-production phases before being put into full production use and our trace was collected before the second phase completed. At the time of data collection, Trinity consisted of 9408 identical compute nodes, a total of 301056 Intel Xeon E5-2698v3 2.3GHz cores and 1.2PB RAM, making this the largest cluster with a publicly available trace by number of CPU cores.

Our Trinity dataset covers 3 months from February to April 2017. During that time, Trinity was operating in OpenScience mode, i.e., the machine was undergoing beta testing and was available to a wider number of users than it is expected to have after it receives its final security classification. We note that OpenScience workloads are representative of a capability supercomputer's workload, as they occur roughly every 18 months when a new machine is introduced, or before an older one is decommissioned. The dataset, which we will henceforth refer to as *OpenTrinity*, consists of 25237 multi-node jobs issued by 88 users and collected by MOAB [1], an open-source cluster scheduling system. The information available in the trace is the same as that in the Mustang trace.

2.4 Google cluster

In 2012, Google released a trace of jobs that ran in one of their compute clusters [41]. It is a 29-day trace consisting of 672074 jobs and 48 million tasks, some of which were issued through the MapReduce framework, and ran on 12583 heterogeneous nodes in May 2011. The workload consists of both long-running services and batch jobs [55]. Google has not released the exact hardware specifications of each cluster node. Instead, as shown in

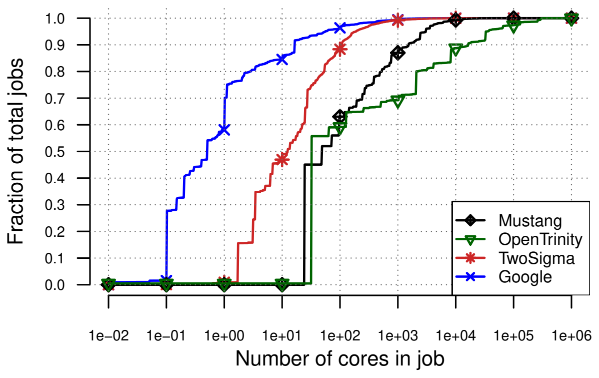

Figure 1: CDF of job sizes based on allocated CPU cores.

Table 2, nodes are presented through anonymized platform names representing machines with different combinations of microarchitectures and chipsets [58]. Note that the number of CPU cores and RAM for each node in the trace have been normalized to the most powerful node in the cluster. In our analysis, we estimate the total number of cores in the Google cluster to be 106544. We derive this number by assuming that the most popular node type (Google B with 0.5 CPU cores) is a dual-socket server, carrying quad-core AMD Opteron Barchelona CPUs that Google allegedly used in their datacenters at the time [26]. Unlike previous workloads, jobs can be allocated fractions of a CPU core [46].

3 Job characteristics

Many instances of prior work in the literature rely on the assumption of heavy-tailed distributions to describe the size and duration of individual jobs [2, 8, 13, 14, 40, 50]. In the LANL and TwoSigma workloads these tails appear significantly lighter.

Observation 1: *On average, jobs in the TwoSigma and LANL traces request 3 - 406 times more CPU cores than jobs in the Google trace. Job sizes in the LANL traces are more uniformly distributed.*

Figure 1 shows the Cumulative Distribution Functions (CDFs) of job requests for CPU cores across all traces, with the x-axis in logarithmic scale. We find that the 90% of smallest jobs in the Google trace request 16 CPU cores or fewer. The same fraction of TwoSigma jobs request 108 cores, and 1-16K cores in the LANL traces. Very large jobs are also more common outside Google. This is unsurprising for the LANL HPC clusters, where allocating thousands of CPU cores to a single job is not uncommon, as the clusters' primary use is to run massively parallel scientific applications. It is interesting to note, however, that while the TwoSigma clusters contain fewer cores than the other clusters we examine (3 times fewer

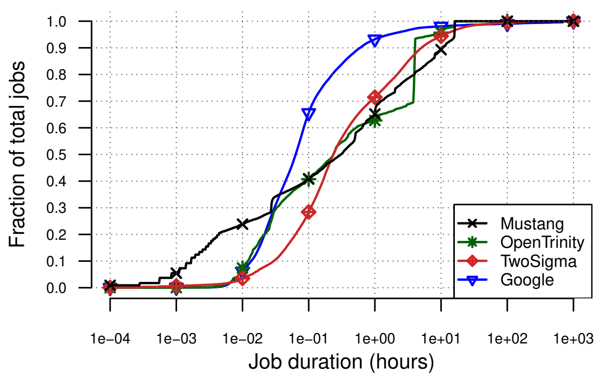

Figure 2: CDF of the durations of individual jobs.

than the Google cluster), its median job is more than an order of magnitude larger than a job in the Google trace. An analysis of allocated memory yields similar trends.

Observation 2: *The median job in the Google trace is 4-5 times shorter than in the LANL or TwoSigma traces. The longest 1% of jobs in the Google trace, however, are 2-6 times longer than the same fraction of jobs in the LANL and TwoSigma traces.*

Figure 2 shows the CDFs of job durations for all traces. We find that in the Google trace, 80% of jobs last less than 12 minutes *each*. In the LANL and TwoSigma traces jobs are at least an order of magnitude longer. In TwoSigma, the same fraction of jobs last up to 2 hours and in LANL, they last up to 3 hours for Mustang and 6 hours for OpenTrinity. Surprisingly, the tail end of the distribution is slightly shorter for the LANL clusters than for the Google and TwoSigma clusters. The longest job is 16 hours on Mustang, 32 hours in OpenTrinity, 200 hours in TwoSigma, and at least 29 days in Google (the duration of the trace). For LANL, this is due to hard limits causing jobs to be indiscriminately killed. For Google, the distribution's long tail is likely attributed to long-running services.

Implications. These observations impact the immediate applicability of job scheduling approaches whose efficiency relies on the assumption that the vast majority of jobs' durations are in the order of minutes, and job sizes are insignificant compared to the size of the cluster. For example, Ananthanarayanan et al. [2] propose to mitigate the effect of stragglers by duplicating tasks of smaller jobs. This is an effective approach for Internet service workloads (Microsoft and Facebook are represented in the paper) because the vast majority of jobs can benefit from it, without significantly increasing the overall cluster utilization. For the Google trace, for example, 90% of jobs request less than 0.01% of the cluster each, so duplicating them only slightly increases cluster utilization. At the same time, 25-55% of jobs in the LANL and TwoSigma traces *each* request *more than* 0.1% of the cluster's cores, decreasing the efficiency of the approach and suggesting replication should be used judiciously. This does not consider that LANL tasks are also tightly-coupled and the entire job has to be duplicated.

Another example is the work by Delgado et al. [14], which improves the efficiency of distributed schedulers for short jobs by dedicating them a fraction of the cluster. This partition ranges from 2% for Yahoo and Facebook traces, to 17% for the Google trace where jobs are significantly longer, to avoid increasing job service times. For the TwoSigma and LANL traces we have shown that jobs are even longer than for the Google trace (Figure 2), so larger partitions will likely be necessary to achieve similar efficiency. At the same time, jobs running in the TwoSigma and LANL clusters are also larger (Figure 1), so service times for long jobs are expected to increase unless the partition is shrunk. Other examples of work that is likely affected include task migration of short and small jobs [50] and hybrid scheduling aimed on improving head-of-line blocking for short jobs [13].

4 Workload heterogeneity

Another common assumption about cloud workloads is that they are characterized by heterogeneity in terms of resources available to jobs, and job interarrival times [7, 23, 31, 46, 56]. The private and HPC clusters we study, however, consist of homogeneous hardware (see Table 2) and user activity follows well-defined diurnal patterns, even though the rate of scheduling requests varies significantly across clusters.

Observation 3: *Diurnal patterns are universal. Clusters received more scheduling requests and smaller jobs at daytime, with minor deviations for the Google trace.*

In Figure 3 we show the number of job scheduling requests for every hour of the day. We choose to show metrics for the median day surrounded by the other two quartiles because the high variation across days causes the averages to be unrepresentative of the majority of days (see Section 8). Overall, diurnal patterns are evident in every trace and user activity is concentrated at daytime (7AM to 7PM), similar to prior work [38]. An exception to this is the Google trace, which is most active from midnight to 4AM, presumably due to batch jobs leveraging the available resources.

Sizes of submitted jobs are also correlated with the time of day. We find that longer, larger jobs in the LANL traces are typically scheduled during the night, while shorter, smaller jobs tend to be scheduled during the day. The reverse is true for the Google trace, which prompts our earlier assumption on nightly batch jobs. Long, large jobs are also scheduled at daytime in the TwoSigma clusters, despite having a diurnal pattern similar to LANL

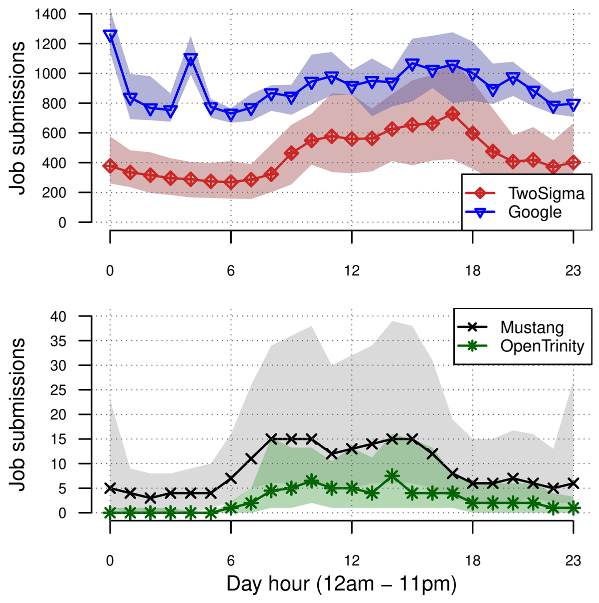

Figure 3: Hourly job submission rates for a given day. The lines represent the median, while the shaded region shows the distance between the 25th and 75th percentiles.

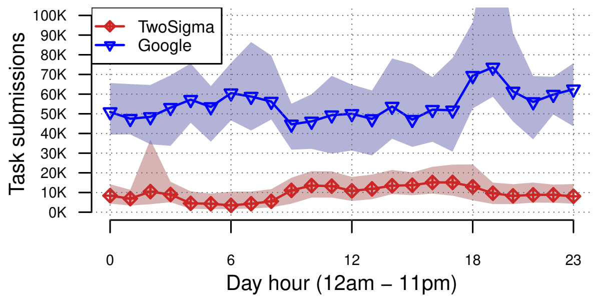

Figure 4: Hourly task placement requests for a given day. The lines represent the median, while the shaded region shows the distance between the 25th and 75th percentiles.

clusters. This is likely due to TwoSigma's workload consisting of financial data analysis, which bears a dependence on stock market hours.

Observation 4: *Scheduling request rates differ by up to 3 orders of magnitude across clusters. Sub-second scheduling decisions seem necessary in order to keep up with the workload.*

One more thing to take away from Figure 3 is that the rate of scheduling requests can differ significantly across clusters. For the Google and TwoSigma traces, hundreds to thousands of jobs are submitted every hour. On the other hand, LANL schedulers never receive more than 40 requests on any given hour. This could be related to the workload or the number of users in the system, as the Google cluster serves 2 times as many user IDs as the Mustang cluster and 9 times as many as OpenTrinity.

Implications: Previous work such as Omega [46] and ClusterFQ [56] propose distributed scheduling designs especially applicable to heterogeneous clusters. This does not seem to be an issue for environments such as LANL and TwoSigma, which intentionally architect homogeneous clusters to lower performance optimization and administration costs.

As cluster sizes increase, so does the rate of scheduling requests, urging us to reexamine prior work. Quincy [31] represents scheduling as a Min-Cost Max-Flow (MCMF) optimization problem over a task-node graph and continuously refines task placement. The complexity of this approach, however, becomes a drawback for large-scale clusters such as the ones we study. Gog et al. [23] find that Quincy requires 66 seconds (on average) to converge to a placement decision in a 10,000-node cluster. The Google and LANL clusters we study already operate on that scale (Table 2). We have shown in Figure 3 that the average frequency of job submissions in the LANL traces is one job every 90 seconds, which implies that this scheduling latency may work, but this will not be the case for long. Trinity is currently operating with 19,000 nodes and, under the DoE's Exascale Computing Project [39], 25 times larger machines are planned within the next 5 years. Note that when discussing scheduling so far we refer to *jobs*, since HPC jobs have a gang scheduling requirement. Placement algorithms such as Quincy, however, focus on *task* placement.

An improvement to Quincy is Firmament [23], a centralized scheduler employing a generalized approach based on a combination of MCMF optimization techniques to achieve sub-second task placement latency on average. As Figure 4 shows, sub-second latency is paramount, since the rate of task placement requests in the Google and TwoSigma traces can be as high as 100K requests per hour, i.e. one task every 36ms. Firmament's placement latency, however, increases to several seconds as cluster utilization increases. For the TwoSigma and Google traces this can be problematic.

5 Resource utilization

A well-known motivation for the cloud has been resource consolidation, with the intention of reducing equipment ownership costs. An equally well-known property of the cloud, however, is that its resources remain underutilized [6, 15, 35, 36, 41]. This is mainly due to a disparity between user resource requests and actual resource usage, which recent research efforts try to alleviate through workload characterization and aggressive consolidation [15, 33, 34]. Our analysis finds that user resource requests in the LANL and TwoSigma traces are characterized by higher variability than in the Google trace. We also look into job inter-arrival times and how they are

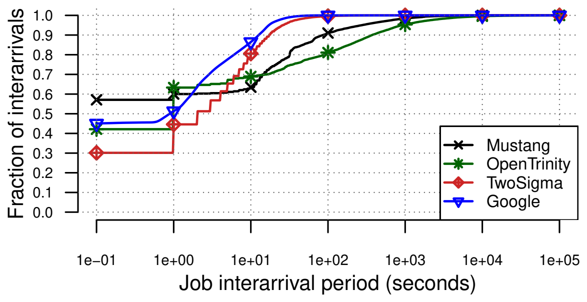

Figure 5: CDF of job interarrival times.

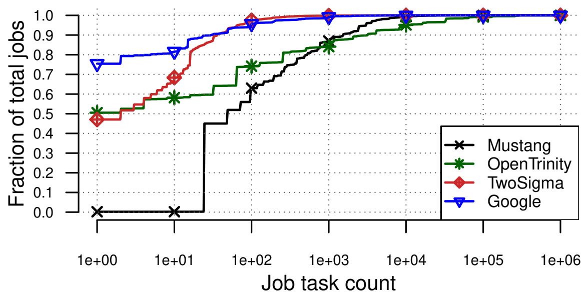

Figure 6: CDF of the number of tasks per job.

approximated when evaluating new research.

Observation 5: *Unlike the Google cluster, none of the other clusters we examine overcommit resources.*

Overall, we find that the fraction of CPU cores allocated to jobs is stable over time across all the clusters we study. For Google, CPU cores are over provisioned by 10%, while for other clusters unallocated cores range between 2-12%, even though resource overprovisioning is supported by their schedulers. Memory allocation numbers follow a similar trend. Unfortunately, the LANL and TwoSigma traces do not contain information on actual resource utilization. As a result, we can neither confirm, nor contradict results from earlier studies on the imbalance between resource allocation and utilization. What differs between organizations is the motivation for keeping resources utilized or available. For Google [41], Facebook [10], and Twitter [15], there is a tension between the financial incentive of maintaining only the necessary hardware to keep operational costs low and the need to provision for peak demand, which leads to low overall utilization. For LANL, clusters are designed to accommodate a predefined set of applications for a predetermined time period and high utilization is planned as part of efficiently utilizing federal funding. For the TwoSigma clusters, provisioning for peak demand is more important, even if it leads to low overall utilization, since business revenue is heavily tied to the response times of their analytics jobs.

Observation 6: *The majority of job interarrivals periods are sub-second in length.*

Interarrival periods are a crucial parameter of an experimental setup, as they dictate the load on the system under test. Two common configurations are second-granularity [15] or Poisson-distributed interarrivals [29], and we find that neither characterizes interarrivals accurately. In Figure 5 we show the CDFs for job interarrival period lengths. We observe that 44-62% of interarrival periods are sub-second, implying that jobs arrive at a faster rate than previously assumed. Furthermore, our attempts to fit a Poisson distribution on this data have been unsuccessful, as Kolmogorov-Smirnov tests [37] reject the null hypothesis with p-values $< 2.2 \times 10^{-16}$. This result does not account for a scenario where there is an underlying Poisson process with a rate parameter changing over time, but it suggests that caution should be used when a Poisson distribution is assumed.

Another common assumption is that jobs are very rarely big, i.e., made up of multiple tasks [29, 56]. In Figure 6 we show the CDFs for the number of tasks per job across organizations. We observe that 77% of Google jobs are single-task jobs, but the rest of the clusters carry many more multi-task jobs. We note that the TwoSigma distribution approaches that of Google only for larger jobs. This suggests that task placement may be a harder problem outside Google, where single-task jobs are common, exacerbating the evaluation issues we outlined in Section 4 for existing task placement algorithms.

Observation 7: *User resource requests are more variable in the LANL and TwoSigma traces than in the Google trace.*

Resource under-utilization can be alleviated through workload consolidation. To ensure minimal interference, applications are typically profiled and classified according to historical data [15, 33]. Our analysis suggests that this approach is likely to be less successful outside the Internet services world. To quantify variability in user behavior we examine the Coefficient of Variation[1] (CoV) across all requests of individual users. For the Google trace we find that the majority of users issue jobs within 2x of their average request in CPU cores. For the LANL and TwoSigma traces, on the other hand, 60-80% of users can deviate by 2-10x of their average request.

Implications: A number of earlier studies of Google [41], Twitter [15], and Facebook [10] data have highlighted the imbalance between resource allocation and utilization. Google tackles this issue by over-committing resources, but this is not the case for LANL and TwoSigma. Another proposed solution is Quasar [15], a system that consolidates workloads while guaranteeing

[1]The Coefficient of Variation is a unit-less measure of spread, derived by dividing a sample's standard deviation by its mean.

a predefined level of QoS. This is achieved by profiling jobs at submission time and classifying them as one of the previously encountered workloads; misclassifications are detected by inserting probes in the running application. For LANL, this approach would be infeasible. First, jobs cannot be scaled down for profiling, as submitted codes are often carefully configured for the requested allocation size. Second, submitted codes are too complex to be accurately profiled in seconds, and probing them at runtime to detect misclassifications can introduce performance jitter that is prohibitive in tightly-coupled HPC applications. Third, in our LANL traces we often find that users tweak jobs before resubmitting them, as they re-calibrate simulation parameters to achieve a successful run, which is likely to affect classification accuracy. Fourth, resources are carefully reserved for workloads and utilization is high, which makes it hard to provision resources for profiling. For the TwoSigma and Google traces Quasar may be a better fit, however, at the rate of 2.7 jobs per second (Figure 3), 15 seconds of profiling [15] at submission time would result in an expected load of 6 jobs being profiled together. Since Quasar requires 4 parallel and isolated runs to collect sufficient profiling data, we would need resources to run at least 360 VMs concurrently, with guaranteed performance isolation between tham to keep up with the average load. This further assumes the profiling time does not need to be increased beyond 15 seconds. Finally, Quasar [15] was evaluated using multi-second inter-arrival periods, so testing would be necessary to ensure that one order of magnitude more load can be handled (Figure 5), and that it will not increase the profiling cost further.

Another related approach to workload consolidation is provided by TSF [56], a scheduling algorithm that attempts to maximize the number of task slots allocated to each job, without favoring bigger jobs. This ensures that the algorithm remains starvation-free, however it results in significant slowdowns in the runtime of jobs with 100+ tasks, which the authors define as big. This would be prohibitive for LANL, where jobs must be scheduled as a whole, and such "big" jobs are much more prevalent and longer in duration. Other approaches for scheduling and placement assume the availability of resources that may be unavailable in the clusters we study here, and their performance is shown to be reduced in highly-utilized clusters [25, 29].

6 Failure analysis

Job scheduler logs are often analyzed to gain an understanding of job failure characteristics in different environments [9, 17, 21, 22, 43]. This knowledge allows for building more robust systems, which is especially important as we transition to exascale computing systems where failures are expected every few minutes [48], and cloud computing environments built on complex software stacks that increase failure rates [9, 44].

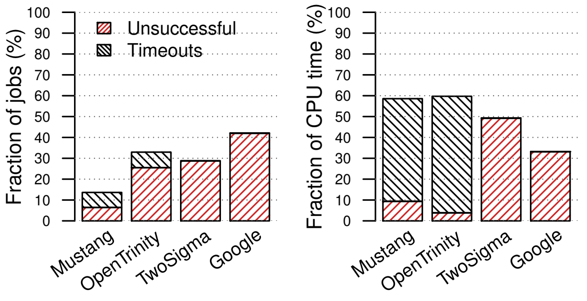

Figure 7: Breakdown of the total number of jobs, as well as CPU time, by job outcome.

Definitions. An important starting point for any failure analysis is defining what constitutes a failure event. Across all traces we consider, we define as *failed jobs* all those that end due to events whose occurrence was not intended by users or system administrators. We do not distinguish failed jobs by their root cause, e.g., software and hardware issues, because this information is not reliably available. There are other job termination states in the traces, in addition to success and failure. For the Google trace, jobs can be killed by users, tasks can be evicted in order to schedule higher-priority ones, or have an unknown exit status. For the LANL traces, jobs can be cancelled intentionally. We group all these job outcomes as *aborted jobs* and collectively refer to failed and aborted jobs as ***unsuccessful jobs***.

There is another job outcome category. At LANL, users are required to specify a runtime estimate for each job. This estimate is treated as a time limit, similar to an SLO, and the scheduler kills the job if the limit is exceeded. We refer to these killings as ***timeout jobs*** and present them separately because they can produce useful work in three cases: (a) when HPC jobs use the time limit as a stopping criterion, (b) when job state is periodically checkpointed to disk, and (c) when a job completes its work before the time limit but fails to terminate cleanly.

Observation 8: *Unsuccessful job terminations in the Google trace are 1.4-6.8x higher than in other traces. Unsuccessful jobs at LANL use 34-80% less CPU time.*

In Figure 7, we break down the total number of jobs (left), as well as the total CPU time consumed by all jobs by job outcome (right). First, we observe that the fraction of unsuccessful jobs is significantly higher (1.4-6.8x) for the Google trace, than for the other traces. This comparison ignores jobs that timeout for Mustang, because as we explained above, it is unlikely they represent wasted resources. We also note that almost all unsuccessful jobs in the Google trace were aborted. According to the trace

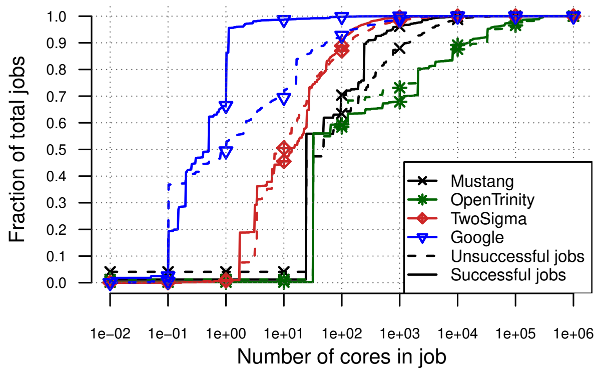

Figure 8: CDFs of job sizes (in CPU cores) for unsuccessful and successful jobs.

documentation [58] these jobs could have been aborted by a user or the scheduler, or by dependent jobs that failed. As a result, we cannot rule out the possibility that these jobs were linked to a failure. For this reason, prior work groups all unsuccessful jobs under the "*failed*" label [17], which we choose to avoid for clarity. Another fact that further highlights how blurred the line between failed and aborted jobs can be, is that all unsuccessful jobs in the TwoSigma trace are assigned a failure status. In short, our classification of jobs as "unsuccesful" may seem broad, but it is consistent with the liberal use of the term "failure" in the literature.

We also find that unsuccessful jobs are not equally detrimental to the overall efficiency of all clusters. While the rate of unsuccessful jobs for the TwoSigma trace is similar to the rate of unsuccessful jobs in the OpenTrinity trace, each unsuccessful job lasts longer. Specifically, unsuccessful jobs in the LANL traces waste 34-80% less CPU time than in the Google and TwoSigma traces. It is worth noting that 49-55% of CPU time at LANL is allocated to jobs that time out, which suggests that at least a small fraction of that time may become available through the use of better checkpoint strategies.

Observation 9: *For the Google trace, unsuccessful jobs tend to request more resources than successful ones. This is untrue for all other traces.*

In Figure 8, we show the CDFs of job sizes (in CPU cores) of individual jobs. For each trace, we show separate CDFs for unsuccessful and successful jobs. By separating jobs based on their outcome we observe that successful jobs in the Google trace request fewer resources, overall, than unsuccessful jobs. This observation has also been made in earlier work [17, 21], but it does not hold for our other traces. CPU requests for successful jobs in the TwoSigma and LANL traces are similar to requests made by unsuccessful jobs. This trend is opposite to what is seen in older HPC job logs [59], and since these traces were also collected through SLURM and MOAB

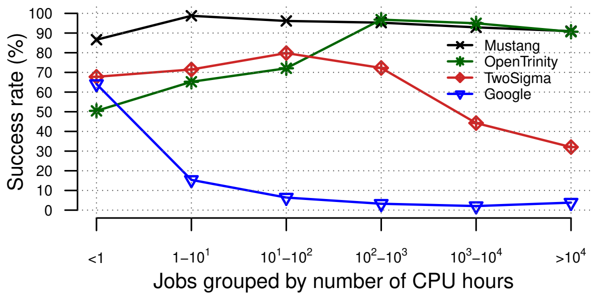

Figure 9: Success rates for jobs grouped by CPU hours.

we do not expect this discrepancy to be due to semantic differences in the way failure is defined across traces.

Observation 10: *For the Google and TwoSigma traces, success rates drop for jobs consuming more CPU hours. The opposite is true for LANL traces.*

For the traces we analyze, the root cause behind unsuccessful outcomes is not reliably recorded. Without this information, it is difficult to interpret and validate the results. For example, we expect that hardware failures are random events whose occurrence roughly approximates some frequency based on the components' Mean Time Between Failure ratings. As a result, jobs that are larger and/or longer, would be more likely to fail. In Figure 9 we have grouped jobs based on the CPU hours they consume (a measure of both size and length), and we show the success rate for each group. The trend that stands out is that success rates decrease for jobs consuming more CPU hours in the Google and TwoSigma traces, but they are increase and remain high for both LANL clusters. This could be attributed to larger, longer jobs at LANL being more carefully planned and tested, but it could also be due to semantic differences in the way success and failure are defined across traces.

Implications. The majority of papers analyzing the characteristics of job failures in the Google trace build failure prediction models that assume the existence of the trends we have shown on success rates and resource consumption of unsuccessful jobs. Chen et al. [9] highlight the difference in resource consumption between unsuccessful and successful jobs, and El-Sayed et al. [17] note that this is the second most influential predictor (next to early task failures) for their failure prediction models. As we have shown in Figure 9, unsuccessful jobs are not linked to resource consumption in other traces. Another predictor highlighted in both studies is job re-submissions, with successful jobs being resubmitted fewer times. We confirm that this trend is consistent across all traces, even though the majority of jobs (83-93%) are submitted exactly once. A final observation that does not hold true for LANL is that CPU time of unsuccessful jobs increases with job runtime [17, 22].

7 A case study on plurality and diversity

Evaluating systems against multiple traces enables researchers to identify practical sensitivities of new research and prove its generality. We demonstrate this through a case study on JVuPredict, the job runtime[2] predictor module of the JamaisVu [51] cluster scheduler. Our evaluation of JVuPredict with all the traces we have introduced revealed the predictive power of logical job names and consistent user behavior in workload traces. Conversely, we found it difficult to obtain accurate runtime predictions in systems that provide insufficient information to identify job re-runs. This section briefly describes the architecture of JVuPredict (Section 7.1) and our evaluation results (Section 7.2).

7.1 JVuPredict background

Recent schedulers [12, 24, 32, 51, 52] use information on job runtimes to make better scheduling decisions. Accurate knowledge of job runtimes allows a scheduler to pack jobs more aggressively in a cluster [12, 18, 54], or to delay a high-priority batch job to schedule a latency-sensitive job without exceeding the deadline of the batch job. In heterogeneous clusters, knowledge of a job's runtime can also be used to decide whether it is better to immediately start a job on hardware that is sub-optimal for it, let it wait until preferred hardware is available, or simply preempt other jobs to let it run [3, 52]. Such schedulers assume most of the provided runtime information is accurate. The accuracy of the provided runtime is important as these schedulers are only robust to a reasonable degree of error [52].

Traditional approaches for obtaining runtime knowledge are often as trivial as expecting the user to provide an estimate, an approach used in HPC environments such as LANL. As we have seen in Section 6, however, users often use these estimates as a stopping criterion (jobs get killed when they exceed them), specify a value that is too high, or simply fix them to a default value. Another option is to detect jobs with a known structure that are easy to profile as a means of ensuring accurate predictions, an approach followed by systems such as Dryad [30], Jockey [20], and ARIA [53]. For periodic jobs, simple history-based predictions can also work well [12, 32]. But these approaches are still inadequate for consolidated clusters without a known structure or history.

JVuPredict, the runtime prediction module of JamaisVu [51], aims to predict a job's runtime when it is submitted, using historical data on past job characteristics and runtimes. It differs from traditional approaches by attempting to detect jobs that repeat, even when successive runs are not declared as repeats. It is more effective, as only part of the history relevant to the newly submitted job is used to generate the estimate. To do this, it uses features of submitted jobs, such as user IDs and job names, to build multiple independent predictors. These predictors are then evaluated based on the accuracy achieved on historic data, and the most accurate one is selected for future predictions. Once a prediction is made, the new job is added to the history and the accuracy scores of each model are recalculated. Based on the updated scores a new predictor is selected and the process is repeated.

[2]The terms *runtime* and *duration* are used interchangeably here.

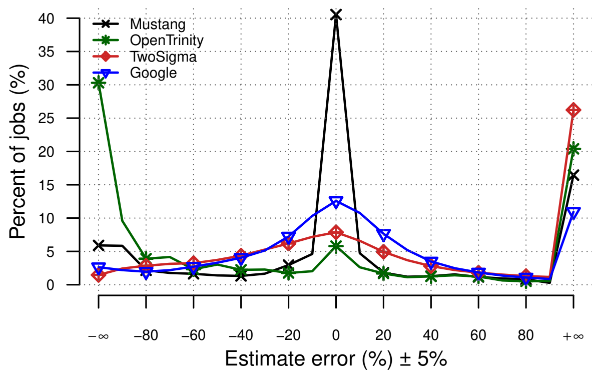

Figure 10: Accuracy of JVuPredict predictions of runtime estimates, for all four traces.

7.2 Evaluation results

JVuPredict had originally been evaluated using only the Google trace. Although predictions are not expected to be perfect, performance under the Google trace was reasonably good, with 86% of predictions falling within a factor of two of the actual runtime. This level of accuracy is sufficient for the JamaisVu scheduler, which further applies techniques to mitigate the effects of such mispredictions. In the end, the performance of JamaisVu with the Google trace is sufficient to closely match that of a hypothetical scheduler with perfect job runtime information and to outperform runtime-unaware scheduling [51]. This section repeats the evaluation of JVuPredict using our new TwoSigma and LANL traces. Our criterion for success is meeting or surpassing the prediction accuracy achieved with the Google trace.

A feature expected to effectively predict job repeats is the job's name. This field is typically anonymized by hashing the program's name and arguments, or simply by hashing the user-defined human-readable job name provided to the scheduler. For the Google trace, predictors using the logical job name field are selected most frequently by JVuPredict due to their high accuracy.

Figure 10 shows our evaluation results. On the x-axis we plot the prediction error for JVuPredict's runtime estimates, as a percentage of the actual runtime of the job. Each data point in the plot is a bucket representing val-

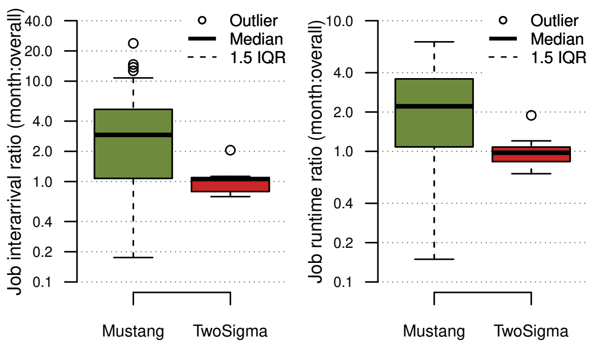

Figure 11: *Is a month representative of the overall workload?* The boxplots show distributions of the average job inter-arrival period (left) and duration (right) per month, normalized by the trace's overall average. Boxplot whiskers are defined at 1.5 times the distribution's Inter-Quartile Range (standard Tukey boxplots).

ues within 5% of the nearest decile. The y-axis shows the percentage of jobs whose predictions fall within each bucket. Overestimations of a job's runtime are easier to tolerate than underestimations, because they cause the scheduler to be more conservative when scheduling the job. Thus, the uptick at the right end of the graph is not alarming. For the Google trace, the total percentage of jobs whose runtimes are under-estimated is 32%, with 11.7% of underestimations being lower than half the actual runtime. We mark these numbers as acceptable, since performance of JVuPredict in the Google trace has been proven exceptional in simulation.

Although the logical job name is a feature that performs well for the Google trace, we find it is either unavailable, or unusable in our other traces. This is because of the difficulty inherent in producing an anonymized version of it, while maintaining enough information to distinguish job repeats. Instead, this field is either assigned a unique value for every job, or entirely omitted from the trace. All traces we introduce in this paper suffer from this limitation. The absence of the field, however, seems to not affect the performance of JVuPredict significantly. The fields selected by JVuPredict as the most effective predictors of job runtime for the Mustang and TwoSigma traces are: the ID of the user who submitted the job, the number of CPU cores requested by the job, or a combination of the two. We find that the TwoSigma workload achieves identical performance to Google: 31% of job runtimes are underestimated and 15% are predicted to be less than 50% of the actual runtime. The Mustang workload is much more predictable, though, with 38% of predictions falling within 5% of the actual runtime. Still, 16% of job runtimes were underestimated by more than half of the actual runtime. The similarity between the TwoSigma and Mustang results suggests that JamaisVu would also perform well under these workloads. Note that these results extend to the Google trace when the job name is omitted.

OpenTrinity performs worse than every other trace. Even though the preferred predictors are, again, the user ID and the number of CPU cores in the job, 55% of predictions have been underestimations. Even worse, 24% of predictions are underestimated by more than 95% of the actual runtime. A likely cause for this result is the variability present in the trace. We are unsure whether this variability is due to the short duration of the trace, or due to the workload being more inconsistent during the OpenScience configuration period.

In conclusion, two insights were obtained by evaluating JVuPredict with multiple traces. First, we find that although logical job names work well for the Google trace, they are hard to produce in anonymized form for other traces, so they may often be unavailable. Second, we find that in the absence of job names, there are other fields that can substitute for them and provide comparable accuracy all but the OpenTrinity trace. Specifically, the user ID and CPU core count for every job seem to perform best for both TwoSigma and the Mustang trace.

8 On the importance of trace length

Working with traces often forces researchers to make key assumptions as they interpret the data, in order to cope with missing information. A common (unwritten) assumption when using or analyzing a trace, is that it sufficiently represents the workload of the environment wherein it was collected. At the same time the Google trace spans only 29 days, while other traces we study in this paper are 3-60 times longer, even covering the entire lifetime of the cluster in the case of Mustang. Being unsure whether 29 days are sufficient to accurately describe a cluster's workload, we decided to examine how representative individual 29-day periods are of the overall workload in our TwoSigma and Mustang traces.

Our experiment consisted of dividing our traces in 29-day periods. For each such month we then compared the distributions of individual metrics against the overall distribution for the full trace. The metrics we considered were: job sizes, durations, and interarrival periods. Overall we found consecutive months' distributions to vary wildly for all these metrics. One distinguishable trend, however, is that during the third year the Mustang cluster is dominated by short jobs arriving in bursts.

Figure 11 summarizes our results by comparing the averages of different metrics for each month against the overall average across the entire trace. The boxplots show the distributions of average job interarrivals (left) and durations (right) per month, when normalized by the overall average for the trace. The boxplots are standard

Tukey boxplots, where the box is framed by the 25^{th} and 75^{th} percentiles, the dark line represents the median, and the whiskers are defined at 1.5 times the distribution's Inter-Quartile Range (IQR), or the furthest data point if no outliers exist (shown in circles here). We see that individual months vary significantly for the Mustang trace, and they differ somewhat less across months in the TwoSigma trace. More specifically, the average job interarrival of a given month can be 0.7-2.0x the value of the overall average in the TwoSigma trace, or 0.2-24x the value of the overall average in the Mustang trace. Average job durations can fluctuate between 0.7-1.9x of the average job duration in the TwoSigma trace, and 0.1-6.9x of the average in the Mustang trace. Overall, our results conclusively show that our cluster workloads display significant differences from month to month.

9 Related Work

The Parallel Workloads Archive (PWA) [19] hosts the largest collection of public HPC traces. At the time of this writing, 38 HPC traces have been collected between 1993 and 2015. Our HPC traces complement this collection. The Mustang trace is unique in a number of ways: it is almost two times longer in duration than the longest publicly available trace, contains four times as many jobs, and covers the entire lifetime of the cluster enabling longitudinal analyses. It is also similar in size to the largest clusters in PWA and its distribution of job duration distribution is shorter than all other HPC traces. The OpenTrinity trace is also complementary to existing traces, as it is collected on a machine almost two times bigger than the largest supercomputer with a publicly available trace (Argonne National Lab's Intrepid) as far as CPU core count is concerned.

Prior studies have looked at private cluster traces, specifically with the aim of characterizing MapReduce workloads. Ren et al. [42] examine three traces from academic Hadoop clusters in an attempt to identify popular application styles and characterize the input/output file sizes, the duration, and the frequency of individual MapReduce stages. These clusters handle significantly less traffic than the Google and TwoSigma clusters we examine. Interestingly, a sizable fraction of interarrival periods for individual jobs are longer than 100 seconds, which resembles our HPC workloads. At the same time, the majority of jobs last less than 8 minutes, which approximates the behavior in the Google trace. Chen et al. [10] look at both private clusters from Cloudera customers and Internet services clusters from Facebook. On the one hand, their private traces cover less than two months, while on the other hand their Facebook traces are much longer than the Google trace. Still, there are similarities in traffic, as measured in job submissions per hour. Specifically, Cloudera customers' private clusters deal with hundreds of job submissions per hour, a traffic pattern similar to the Two Sigma clusters, while Facebook handles upwards of a thousand submissions per hour, which is more related to traffic in the Google cluster. The diversity across these workloads further emphasizes the need for researchers to focus on evaluating new research using a diverse set of traces.

Other studies that look at private clusters focus on Virtual Machine workloads. Shen et al. [47] analyze datasets of monitoring data from individual VMs in two private clusters. They report high variability in resource consumption across VMs, but low overall cluster utilization. Cano et al. [5] examine telemetry data from 2000 clusters of Nutanix customers. The frequency of telemetry collection varies from minutes to days and includes storage, CPU measurements, and maintenance events. The authors report fewer hardware failures in these systems than previously reported in the literature. Cortez et al. [11] characterize the VM workload on Azure, Microsoft's cloud computing platform. They also report low cluster utilization and low variability in tenant job sizes.

Conclusion

We have introduced and analyzed job scheduler traces from two private and two HPC clusters. We publicly release all four traces, which we expect to be of interest to researchers due to their unique characteristics, including: the longest public trace to date spanning the entire 5-year lifetime of a cluster, one representing the largest cluster with a public trace to date, and the two longest private non-academic cluster traces made public to date.

Our analysis showed that the private clusters resemble the HPC workloads studied, rather than the popular Google trace workload, which is surprising. This observation holds across many aspects of the workload: job sizes and duration, resource allocation, user behavior variability, and unsuccessful job characteristics. We also listed prior work that relies too heavily on the Google trace's characteristics and may be affected.

Finally, we demonstrated the importance of dataset plurality and diversity in the evaluation of new research. For job runtime predictions, we show that using multiple traces allowed us to reliably rank data features by predictive power. We hope that by publishing our traces we will enable researchers to better understand the sensitivity of new research to different workload characteristics.

Dataset availability

The LANL Mustang, LANL OpenTrinity, and two Two Sigma scheduler logs can be downloaded from the ATLAS repository, which is publicly accessible through:

`www.pdl.cmu.edu/ATLAS`

Acknowledgments

We would like to thank John Wilkes, Steve Senator, our shepherd Konstantinos Karanasos, and the anonymous reviewers for their invaluable feedback. We also thank the members and companies of the PDL Consortium for their insights, feedback, and support: Alibaba Group, Broadcom, Dell EMC, Facebook, Google, HP Enterprise, Hitachi, IBM Research, Intel, Micron, Microsoft Research, MongoDB, NetApp, Oracle, Salesforce, Samsung, Seagate Technology, Two Sigma, Toshiba, Veritas and Western Digital. A portion of this work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-18-24488.

References

[1] ADAPTIVE COMPUTING. MOAB HPC Suite. `http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/`.

[2] ANANTHANARAYANAN, G., GHODSI, A., SHENKER, S., AND STOICA, I. Effective Straggler Mitigation: Attack of the Clones. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (Lombard, IL, 2013), USENIX, pp. 185–198.

[3] BOUTIN, E., EKANAYAKE, J., LIN, W., SHI, B., ZHOU, J., QIAN, Z., WU, M., AND ZHOU, L. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, 2014), USENIX Association, pp. 285–300.

[4] BURTSEV, A., RADHAKRISHNAN, P., HIBLER, M., AND LEPREAU, J. Transparent checkpoints of closed distributed systems in emulab. In *Proceedings of the 4th ACM European Conference on Computer Systems* (New York, NY, USA, 2009), EuroSys '09, ACM, pp. 173–186.

[5] CANO, I., AIYAR, S., AND KRISHNAMURTHY, A. Characterizing Private Clouds: A Large-Scale Empirical Analysis of Enterprise Clusters. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (New York, NY, USA, 2016), SoCC '16, ACM, pp. 29–41.

[6] CARVALHO, M., CIRNE, W., BRASILEIRO, F., AND WILKES, J. Long-term SLOs for reclaimed cloud computing resources. In *ACM Symposium on Cloud Computing (SoCC)* (Seattle, WA, USA, 2014), pp. 20:1–20:13.

[7] CHEN, C., WANG, W., ZHANG, S., AND LI, B. Cluster Fair Queueing: Speeding up Data-Parallel Jobs with Delay Guarantees. In *Proceedings of the IEEE International Conference on Computer Communications* (May 2017), IEEE INFOCOM 2017.

[8] CHEN, S., GHORBANI, M., WANG, Y., BOGDAN, P., AND PEDRAM, M. Trace-Based Analysis and Prediction of Cloud Computing User Behavior Using the Fractal Modeling Technique. In *2014 IEEE International Congress on Big Data* (June 2014), pp. 733–739.

[9] CHEN, X., LU, C. D., AND PATTABIRAMAN, K. Failure Analysis of Jobs in Compute Clouds: A Google Cluster Case Study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering* (Nov 2014), pp. 167–177.

[10] CHEN, Y., ALSPAUGH, S., AND KATZ, R. Interactive Analytical Processing in Big Data Systems: A Cross-industry Study of MapReduce Workloads. *Proc. VLDB Endow. 5*, 12 (Aug. 2012), 1802–1813.

[11] CORTEZ, E., BONDE, A., MUZIO, A., RUSSINOVICH, M., FONTOURA, M., AND BIANCHINI, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *ACM SIGOPS operating systems review* (2017), ACM.

[12] CURINO, C., DIFALLAH, D. E., DOUGLAS, C., KRISHNAN, S., RAMAKRISHNAN, R., AND RAO, S. Reservation-based Scheduling: If You're Late Don't Blame Us! In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2014), SOCC '14, ACM, pp. 2:1–2:14.

[13] DELGADO, P., DIDONA, D., DINU, F., AND ZWAENEPOEL, W. Job-aware Scheduling in Eagle: Divide and Stick to Your Probes. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (New York, NY, USA, 2016), SoCC '16, ACM, pp. 497–509.

[14] DELGADO, P., DINU, F., KERMARREC, A.-M., AND ZWAENEPOEL, W. Hawk: Hybrid Datacenter Scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 499–510.

[15] DELIMITROU, C., AND KOZYRAKIS, C. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 127–144.

[16] EIDE, E., STOLLER, L., AND LEPREAU, J. An Experimentation Workbench for Replayable Networking Research. In *Proceedings of the 4th USENIX Conference on Networked Systems Design &#38; Implementation* (Berkeley, CA, USA, 2007), NSDI'07, USENIX Association, pp. 16–16.

[17] EL-SAYED, N., ZHU, H., AND SCHROEDER, B. Learning from Failure Across Multiple Clusters: A Trace-Driven Approach to Understanding, Predicting, and Mitigating Job Terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (June 2017), pp. 1333–1344.

[18] FEITELSON, D. G., RUDOLPH, L., AND SCHWIEGELSHOHN, U. *Parallel Job Scheduling — A Status Report*. Springer Berlin Heidelberg, 2005, pp. 1–16.

[19] FEITELSON, D. G., TSAFRIR, D., AND KRAKOV, D. Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing 74*, 10 (2014), 2967 – 2982.

[20] FERGUSON, A. D., BODIK, P., KANDULA, S., BOUTIN, E., AND FONSECA, R. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems* (New York, NY, USA, 2012), EuroSys '12, ACM, pp. 99–112.

[21] GARRAGHAN, P., MORENO, I. S., TOWNEND, P., AND XU, J. An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment. *IEEE Transactions on Emerging Topics in Computing 2*, 2 (June 2014), 166–180.

[22] GHIT, B., AND EPEMA, D. Better Safe Than Sorry: Grappling with Failures of In-Memory Data Analytics Frameworks. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2017), HPDC '17, ACM, pp. 105–116.

[23] GOG, I., SCHWARZKOPF, M., GLEAVE, A., WATSON, R. N. M., AND HAND, S. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 99–115.

[24] GRANDL, R., CHOWDHURY, M., AKELLA, A., AND ANANTHANARAYANAN, G. Altruistic scheduling in multi-resource clusters. In *OSDI* (2016), pp. 65–80.

[25] GU, J., LEE, Y., ZHANG, Y., CHOWDHURY, M., AND SHIN, K. G. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 649–667.

[26] HENNESSY, J. L., AND PATTERSON, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[27] HIBLER, M., RICCI, R., STOLLER, L., DUERIG, J., GURUPRASAD, S., STACK, T., WEBB, K., AND LEPREAU, J. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference* (Berkeley, CA, USA, 2008), ATC'08, USENIX Association, pp. 113–128.

[28] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R., SHENKER, S., AND STOICA, I. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2011), NSDI'11, USENIX Association, pp. 295–308.

[29] HUNG, C.-C., GOLUBCHIK, L., AND YU, M. Scheduling Jobs Across Geo-distributed Datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (New York, NY, USA, 2015), SoCC '15, ACM, pp. 111–124.

[30] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), EuroSys '07, ACM, pp. 59–72.

[31] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy: Fair Scheduling for Distributed Computing Clusters. In *Proceedings of the 22nd Symposium on Operating Systems Principles* (October 2009), SOSP '15, ACM, pp. 261–276.

[32] JYOTHI, S. A., CURINO, C., MENACHE, I., NARAYANAMURTHY, S. M., TUMANOV, A., YANIV, J., MAVLYUTOV, R., GOIRI, Í., KRISHNAN, S., KULKARNI, J., ET AL. Morpheus: towards automated slos for enterprise clusters. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation* (2016), USENIX Association, pp. 117–134.

[33] LEVERICH, J., AND KOZYRAKIS, C. Reconciling High Server Utilization and Sub-millisecond Quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 4:1–4:14.

[34] LIU, J., SHEN, H., AND CHEN, L. CORP: Cooperative Opportunistic Resource Provisioning for Short-Lived Jobs in Cloud Systems. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)* (Sept 2016), pp. 90–99.

[35] MA, J., SUI, X., SUN, N., LI, Y., YU, Z., HUANG, B., XU, T., YAO, Z., CHEN, Y., WANG, H., ZHANG, L., AND BAO, Y. Supporting Differentiated Services in Computers via Programmable Architecture for Resourcing-on-Demand (PARD). In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 131–143.

[36] MARSHALL, P., KEAHEY, K., AND FREEMAN, T. Improving Utilization of Infrastructure Clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Washington, DC, USA, 2011), CCGRID '11, IEEE Computer Society, pp. 205–214.

[37] MASSEY JR., F. J. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association 46*, 253 (1951), 68–78.

[38] MENESES, E., NI, X., JONES, T., AND MAXWELL, D. Analyzing the Interplay of Failures and Workload on a Leadership-Class Supercomputer. In *Cray User Group Conference (CUG)* (April 2015).

[39] OFFICE OF SCIENCE (DOE-SC) AND THE NATIONAL NUCLEAR SECURITY ADMINISTRATION (NNSA), U.S. DEPARTMENT OF ENERGY. Exascale Computing Project. `https://exascaleproject.org/`.

[40] RAMPERSAUD, S., AND GROSU, D. Sharing-Aware Online Virtual Machine Packing in Heterogeneous Resource Clouds. *IEEE Transactions on Parallel and Distributed Systems 28*, 7 (July 2017), 2046–2059.

[41] REISS, C., TUMANOV, A., GANGER, G. R., KATZ, R. H., AND KOZUCH, M. A. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (New York, NY, USA, 2012), SoCC '12, ACM, pp. 7:1–7:13.

[42] REN, K., KWON, Y., BALAZINSKA, M., AND HOWE, B. Hadoop's Adolescence: An Analysis of Hadoop Usage in Scientific Workloads. *Proc. VLDB Endow. 6*, 10 (Aug. 2013), 853–864.

[43] ROS, A., CHEN, L. Y., AND BINDER, W. Predicting and Mitigating Jobs Failures in Big Data Clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (May 2015), pp. 221–230.

[44] ROS, A., CHEN, L. Y., AND BINDER, W. Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (June 2015), pp. 207–218.

[45] SCHEDMD. SLURM Workload Manager. `https://slurm.schedmd.com/`.

[46] SCHWARZKOPF, M., KONWINSKI, A., ABD-EL-MALEK, M., AND WILKES, J. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)* (Prague, Czech Republic, 2013), pp. 351–364.

[47] SHEN, S., V. BEEK, V., AND IOSUP, A. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (May 2015), pp. 465–474.

[48] SNIR, M., WISNIEWSKI, R. W., ABRAHAM, J. A., ADVE, S. V., BAGCHI, S., BALAJI, P., BELAK, J., BOSE, P., CAPPELLO, F., CARLSON, B., CHIEN, A. A., COTEUS, P., DEBARDELEBEN, N. A., DINIZ, P. C., ENGELMANN, C., EREZ, M., FAZZARI, S., GEIST, A., GUPTA, R., JOHNSON, F., KRISHNAMOORTHY, S., LEYFFER, S., LIBERTY, D., MITRA, S., MUNSON, T., SCHREIBER, R., STEARLEY, J., AND HENSBERGEN, E. V. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications 28*, 2 (2014), 129–173.

[49] THE APACHE SOFTWARE FOUNDATION. Apache Spark. `https://spark.apache.org/`.

[50] THINAKARAN, P., GUNASEKARAN, J. R., SHARMA, B., KANDEMIR, M. T., AND DAS, C. R. Phoenix: A Constraint-Aware Scheduler for Heterogeneous Datacenters. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (June 2017), pp. 977–987.

[51] TUMANOV, A., JIANG, A., PARK, J. W., KOZUCH, M. A., AND GANGER, G. R. JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes. Tech. Rep. CMU-PDL-16-104, Carnegie Mellon University, September 2016.

[52] TUMANOV, A., ZHU, T., PARK, J. W., KOZUCH, M. A., HARCHOL-BALTER, M., AND GANGER, G. R. TetriSched: Global Rescheduling with Adaptive Plan-ahead in Dynamic Heterogeneous Clusters. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 35:1–35:16.

[53] VERMA, A., CHERKASOVA, L., AND CAMPBELL, R. H. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing* (New York, NY, USA, 2011), ICAC '11, ACM, pp. 235–244.

[54] VERMA, A., KORUPOLU, M., AND WILKES, J. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014* (11 2014), pp. 48–56.

[55] VERMA, A., PEDROSA, L., KORUPOLU, M. R., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)* (Bordeaux, France, 2015).

[56] WANG, W., LI, B., LIANG, B., AND LI, J. Multi-resource Fair Sharing for Datacenter Jobs with Placement Constraints. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2016), pp. 1003–1014.

[57] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.

[58] WILKES, J. More Google cluster data. Google research blog, Nov. 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[59] YUAN, Y., WU, Y., WANG, Q., YANG, G., AND ZHENG, W. Job failures in high performance computing systems: A large-scale empirical study. *Computers & Mathematics with Applications 63*, 2 (2012), 365 – 377.