Bigger, Longer, Fewer: what do cluster jobs look like outside Google?

George Amvrosiadis[†], Jun Woo Park[†], Gregory R. Ganger[†], Garth A. Gibson[†],
Elisabeth Baseman[‡], Nathan DeBardeleben[‡]

[†]Carnegie Mellon University
[‡]Los Alamos National Laboratory

CMU-PDL-17-104
October 2017

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

In the last 5 years, a set of job scheduler logs released by Google has been used in more than 400 publications as the token cloud workload. While this is an invaluable trace, we think it is crucial that researchers evaluate their work under other workloads as well, to ensure the generality of their techniques. To aid them in this process, we analyze three new traces consisting of job scheduler logs from one private and two HPC clusters. We further release the two HPC traces, which we expect to be of interest to the community due to their unique characteristics. The new traces represent clusters 0.3-3 times the size of the Google cluster in terms of CPU cores, and cover a 3-60 times longer time span.
This paper presents an analysis of the differences and similarities between all aforementioned traces. We discuss a variety of aspects: job characteristics, workload heterogeneity, resource utilization, and failure rates. More importantly, we review assumptions from the literature that were originally derived from the Google trace, and verify whether they hold true when the new traces are considered. For those assumptions that are violated, we examine affected work from the literature. Finally, we demonstrate the importance of dataset plurality in job scheduling research by evaluating the performance of JVuPredict, the job runtime estimate module of the TetriSched scheduler, using all four traces.

Acknowledgements: 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-17-28045. We would like to thank John Wilkes for his invaluable feedback. We also thank the members and companies of the PDL Consortium (Broadcom, Citadel, Dell EMC, Google, HP Labs, Hitachi, Intel, Microsoft Research, MongoDB, NetApp, Oracle, Samsung, Seagate Technology, Tintri, Two Sigma, Toshiba, Uber, Veritas and Western Digital) for their interest, insights, feedback, and support.

Keywords: cloud, clusters, study, workload, scheduling, traces, JamaisVu, heterogeneity, utilization, failure rates

Section	Characteristic	Google	HedgeFund	Mustang	OpenTrinity
Job Characteristics (§3)	Majority of jobs are small	✔	✘	✘	✘
	Majority of jobs are short	✔	✘	✘	✘
Workload Heterogeneity (§4)	Existence of diurnal patterns	✔	✔	✔	✔
	High job submission rate	✔	✔	✘	✘
Resource Utilization (§5)	Resource over-commitment	✔	✘	✘	✘
	Sub-second job inter-arrival periods	✔	✔	✔	✔
	User request variability	✘	✔	✔	✔
Failure Analysis (§6)	High job failure rates	✔	✔	✘	✘
	Significant resource waste due to failures	✔	✔	✘	✔
	Longer/larger jobs fail more often	✔	✘	✘	✘

Table 1: Summary of the most important Google trace characteristics, and their universality across traces.

1 Introduction

Despite intense activity in the areas of cloud and job scheduling research, publicly available cluster workload datasets remain scarce. There are three major cluster dataset sources today: the Google cluster trace [57] collected in 2011, the Parallel Workload Archive with HPC traces collected between 1993 and 2015 [20], and the Grid Workload Archive [13]. Of these, the Google cluster trace is by far the most popular, having been used in more than 400 publications in the last 5 years.

As the Google trace's popularity rises, we set out to question whether it is fair to consider it as the representative cloud workload, and whether using it in isolation when evaluating a new technique is sufficient to prove the technique's generality. Furthermore, the trace itself is only 29 days long, so it is uncertain whether the long-term trends, such as job submission and failure rates, that are derived from it and used widely in the literature should even be considered representative of the overall workload of the cluster where it originated.

Our goal is to put assumptions made in the literature about cluster workloads to test, and provide researchers with additional datasets for evaluating their work. To achieve this, our first contribution is to introduce three new traces: one from a private cloud, and two from High Performance Computing (HPC) clusters located at Los Alamos National Laboratory (LANL). Of those, we publicly release both LANL traces. While several other HPC traces are already available [13, 20], the two we consider here are unique: one is a 5-year trace covering the entire lifetime of a general-purpose HPC cluster, and the other originates from Trinity, the flagship supercomputer at LANL in 2017. We introduce all three traces, and the environments where they were collected, in Section 2.

Our second contribution is a trace analysis examining the generality of cluster workload assumptions derived from the Google trace, when our three new traces are considered. Table 1 shows a summary of our findings. For those assumptions that are violated, we further examine their usage in the literature, and attempt to list techniques that may be affected. To achieve this, we surveyed 400 papers that cite the study of the Google trace by Reiss et al. [40] to identify popular cluster workload assumptions. We then compare and contrast these assumptions with trends in the private and LANL clusters to find the ones that are violated. We group the most important assumptions into the following categories: job characteristics (Section 3), workload heterogeneity (Section 4), resource utilization (Section 5), and failure analysis (Section 6).

Our third contribution is a case study of the importance of dataset plurality in the evaluation of new research. For our demonstration we use JVuPredict, a component of the JamaisVu scheduling system [51].

Platform	Nodes	CPUs	RAM	Length
LANL Trinity	9408	32	128GB	3 months
LANL Mustang	1600	24	64GB	5 years
HedgeFund A	872	24	256GB	9 months
HedgeFund 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 with traces in this paper. For the Google trace [40], (*) signifies a resource has been normalized to the largest node.

JVuPredict is the component responsible for providing job runtime estimates to the scheduler, which it achieves by training itself on historic data. The accuracy of the generated runtime estimates were previously evaluated using the Google trace. Evaluating JVuPredict with our traces resulted in better results for traces containing fields that make it easier to detect related and recurring jobs (which have more predictable behavior), helping us identify the importance of such fields for future data collection. We describe our findings in Section 7.

Finally, we discuss experiences with data anonymization and the effect of trace length on its ability to represent a cluster workload in Section 8. We list related work studying cluster traces in Section 9, before concluding.

2 Dataset information

We introduce three datasets consisting of job scheduler logs that were collected from clusters deployed in two organizations: one general-purpose and one cutting-edge supercomputer at LANL, and clusters of a private firm. The following subsections describe each dataset in more detail, and the hardware configurations of each cluster are summarized in Table 2.

In our analysis, a *job* is defined as a single user request to the cluster scheduler to execute a specific program. Multiple jobs are often started en masse or in sequence via user scripts, and we consider each one of these jobs individually. Each job can spawn a collection of processes, which we refer to as *tasks*, that are distributed across cluster nodes. In the context of HPC clusters, where resources are allocated at the granularity of physical nodes similar to Emulab [4, 17, 27, 56], tasks from different jobs are never scheduled on the same node. This is not necessarily true in private clusters.

2.1 LANL Mustang cluster

Mustang was a generic HPC cluster used for *capacity computing* at LANL from 2011 to 2016. Mustang was allocated to users in a fashion similar to Emulab [27, 56], at the granularity of physical nodes. Capacity clusters such as Mustang are architected as efficient, cost-effective compute clusters intended to act as a general-purpose resource used by a large number of users. It was largely used by staff at the LANL facilities, including scientists, engineers, and software developers.

Mustang consisted of 1600 identical compute nodes, with a total of 38400 AMD Opteron 6176 2.3GHz cores and 102.4TB 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. It consists of 2.1 million multi-node jobs collected by SLURM [45], an open-source cluster resource manager, and submitted by 565 users. Submission, start, and end times are available for every job, including the number of cores and nodes allocated, its duration, its time budget (a user-specified runtime estimate that if exceeded, causes the job to be killed), its owner, and job outcome information.

2.2 LANL Trinity supercomputer

In 2017, Trinity is the largest supercomputer at LANL, and it is used for *capability computing*. Capability supercomputers are a large-scale, high-demand resource introducing novel hardware technologies with the purpose of 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 1.16PB 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 when it is in production use, after receiving its final security classification. We note that OpenScience period workloads are expected to be representative of a capability supercomputer's workload, and OpenScience periods 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.

2.3 Private hedge fund firm cluster

We refer to the private company trace as the *HedgeFund* trace. It originates from two clusters operating in two datacenters of a private hedge fund firm. The workload consists of jobs analyzing financial data. A large fraction of those jobs are serviced by home-grown data analysis frameworks, while the rest are serviced by a Spark [49] installation. There are no long-running services, and datacenter resources are not over-committed. Between the two datacenters there are 1313 identical compute nodes, with a total of 31512 CPU cores and 328TB RAM.

The dataset covers 9 and 7 months of each datacenters' operation respectively, starting in January 2016. It consists of 78.5 million tasks, most of which were collected by an internally-developed job scheduler running on top of Mesos, as part of 3.2 million jobs scheduled between the two datacenters. Unlike LANL, users do not have to specify a time limit when submitting a job, so job runtime is not budgeted as strictly. Another difference is that users can allocate individual cores (depending on the data analysis framework configuration), as opposed to entire physical nodes. The fields available in the trace are similar to those in the Trinity and Mustang traces. Since the workload is similar across both datacenters, we present results from both datacenters combined for the remainder of the paper.

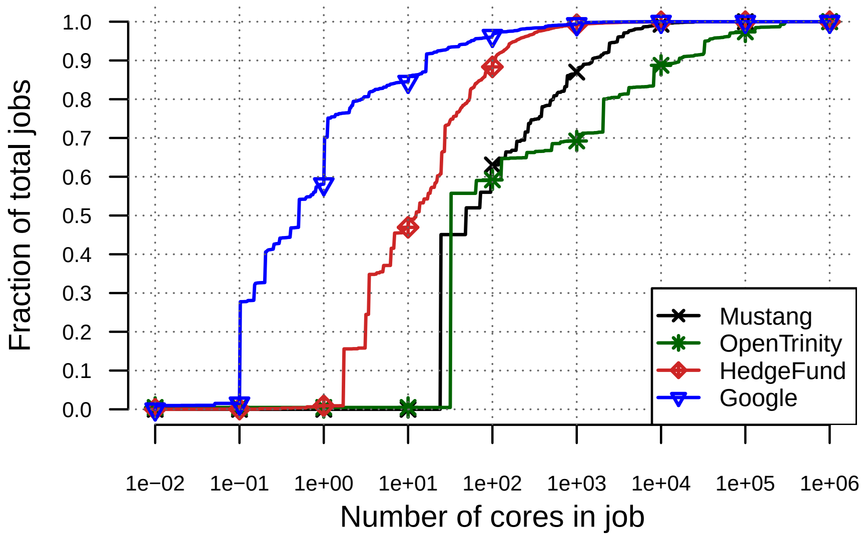

Figure 1: CDF of job sizes, in terms of CPU cores.

2.4 Google cluster

In 2012, Google released a trace of the jobs that ran in one of their compute clusters [40]. This trace, being the largest non-HPC cluster trace, has been used extensively in the literature. It consists of 48 million tasks issued as part of 672074 jobs, some of which were issued through the MapReduce framework. The trace was collected over 29 days from a heterogeneous cluster of 12583 compute nodes in May 2011.

Google has not released the exact hardware specifications of each node type. Instead, as shown in Table 2, nodes are presented through distinct, anonymized platform names representing machines with different combinations of microarchitectures and chipsets [57]. 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, which Google reportedly used in their datacenters in 2009-2010 [26]. Unlike previous workloads, this trace contains long-running services that often exceed the length of the trace. The unit of allocation at Google can be as small as a fraction 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, 14, 15, 39, 50]. In our analysis of the LANL and HedgeFund workloads these tails are often significantly lighter.

Observation 1: *On average, jobs in the HedgeFund 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. We find that the 90% of smallest jobs in the Google trace request 16 CPU cores or fewer. The same fraction of jobs request 108 cores in the HedgeFund trace and 1535-16383 cores in the LANL traces. Very large jobs are actually more common outside Google. While less than 0.3% of jobs request more than one

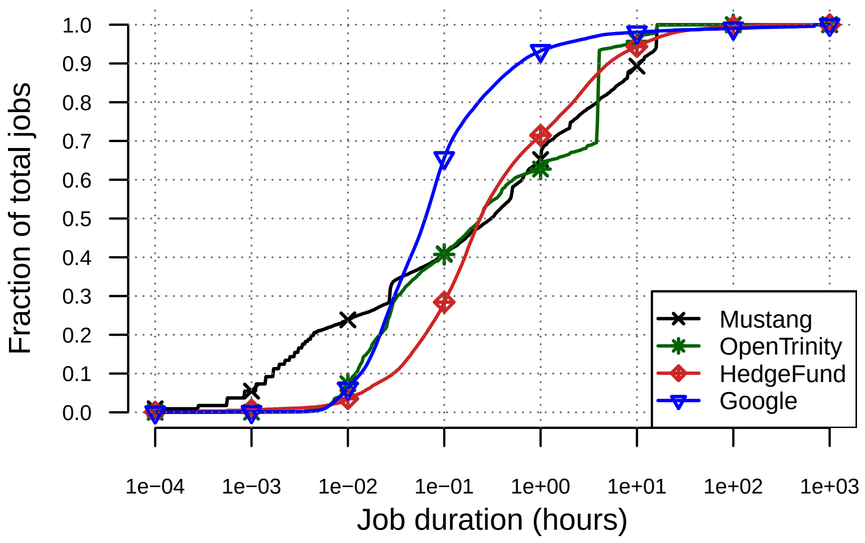

Figure 2: CDF of the durations of individual jobs.

percent of the cores in the Google cluster, such large resource requests are placed by 3% of jobs in HedgeFund, 20% of jobs in OpenTrinity, and 22% of jobs in Mustang. The latter two results are unsurprising, as LANL clusters are primarily used to run massively parallel scientific applications. Note that these requests are even larger for Trinity, which is 10 times larger than Mustang, but they are still expected for a cluster designed to push the scale boundary of scientific applications. The relatively high percentage is surprising for the HedgeFund cluster, however, even if we consider that the traced Google cluster is 3 times as big (so HedgeFund requests are expected to be smaller). An analysis of memory usage yields similar trends.

Observation 2: *The median job in the Google trace is 4-5 times shorter than in the LANL or HedgeFund 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 HedgeFund traces.*

Figure 2 shows the CDFs of job durations for all organizations. We find that in the Google trace, 80% of jobs last less than 12 minutes *each*. In the LANL and HedgeFund traces jobs are at least an order of magnitude longer. In HedgeFund, the same fraction of jobs last up to 2 hours and in LANL, they can last up to 3 hours for Mustang and 6 hours for OpenTrinity. Another way to view this data is that the percentage of jobs that last less than 12 minutes in LANL and HedgeFund ranges from 30-55%. Surprisingly, the tail end of the distribution is slightly shorter for the LANL clusters than for the Google and HedgeFund clusters. The longest job is 16 hours on Mustang, due to a hard time limit, 32 hours in OpenTrinity, 200 hours in HedgeFund, and at least 29 days in Google (the duration of the trace).

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 (Google 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 HedgeFund traces request *more* than 0.1% of the cluster's cores *each*, significantly decreasing the efficiency of the approach. This does not consider the fact that at LANL tasks are also tightly-coupled and the entire job has to be duplicated.

Another example is the work by Delgado et al. [15], 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 HedgeFund 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, tasks running in the HedgeFund 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 relevant approaches include task migration of short and small jobs [50], and hybrid scheduling aimed on improving head-of-line blocking for short jobs [14].

4 Workload heterogeneity

Another common assumption about cloud workloads that is derived from analyzing the Google trace, is that clouds are characterized by heterogeneity both in terms of the resources available to jobs, and in terms of job interarrival times [7, 24, 30, 46, 55]. The four new clusters we study, however, are homogeneous in terms of hardware (see Table 2), and activity follows well-defined diurnal patterns. These diurnal patterns, however, do not always manifest as expected, and the rate of scheduling requests varies significantly across clusters.

Observation 3: *While diurnal patterns are evident, they don't always emerge as expected. All traced clusters received more scheduling requests and smaller jobs during daytime, with the exception of the Google trace.*

In Figure 3, we show the average number of job scheduling requests for every hour of the day (time is adjusted to the timezone where each datacenter is located). First, we note that despite serving users from different locations, diurnal patterns are evident in every trace. The activity, however, is concentrated at daytime (7AM to 7PM), which is in agreement with findings in the literature [36]. An exception to this is the Google trace, which is most active from midnight to 4AM.

Apart from job submissions, the sizes of submitted jobs are also correlated with the time of day. By analyzing the distribution of jobs' duration and size, 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, and we have no information to conclude whether this is due to user habits, maintenance, or intentional to avoid impacting the latency of user-facing services running during the day. This trend also extends to the HedgeFund cluster, despite the diurnal pattern being similar to that in the LANL clusters (more jobs are scheduled during the day). This may be due to the HedgeFund cluster'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 between clusters. Sub-second scheduling decisions seem a necessity in order to keep up with the average workload.*

One more thing that is evident from Figure 3 is that the rate of requests to the scheduler can differ significantly across clusters. For the Google and HedgeFund 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 just 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 Trinity.

Implications: Previous work such as Omega [46] and ClusterFQ [55] propose distributed scheduling designs especially applicable to heterogeneous clusters. This does not seem to be an issue for environments such as LANL and HedgeFund, which intentionally architect homogeneous clusters to lower performance optimization and administration costs. Scheduling constraints for jobs at LANL are supported by the job scheduler, but we find that users rarely relinquish control of the most powerful settings. For example, more

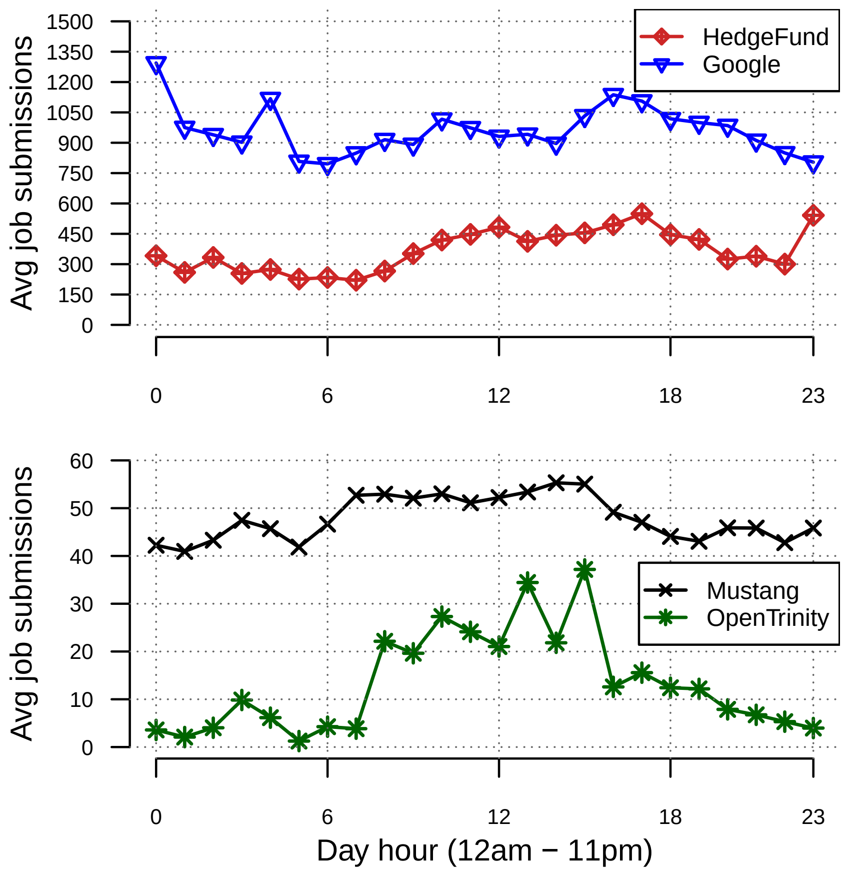

Figure 3: Average job submissions per hour of day.

than 91% of jobs in the OpenTrinity trace disable the option allowing them to be voluntarily preempted if needed, and fewer than 8% of jobs accept to be scheduled using backfilling, i.e., best-effort, low-priority scheduling.

Another category of relevant prior work is that of scalable schedulers. Quincy [30] 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. [24] find that Quincy can require 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 is will not be the case for long. Trinity is currently operating with 19000 nodes, and under the DoE's Exascale Computing Project [38], 25 times larger (exascale) machines are planned within the next 5 years. Note that when discussing scheduling so far we have used the term *job*, since all nodes requested by a LANL job need to be available for the same time period. Placement algorithms such as Quincy, however, are targeted for *task* placement. In Figure 4, we show the average rate of task scheduling requests in the Google and HedgeFund traces. In those organizations, sub-second placement appears paramount.

An improvement to Quincy is Firmament [24], a distributed scheduler employing a generalized approach based on a combination of MCMF optimization techniques to achieve sub-second task placement latency on average. This latency, however, increases to several seconds as cluster utilization increases. For

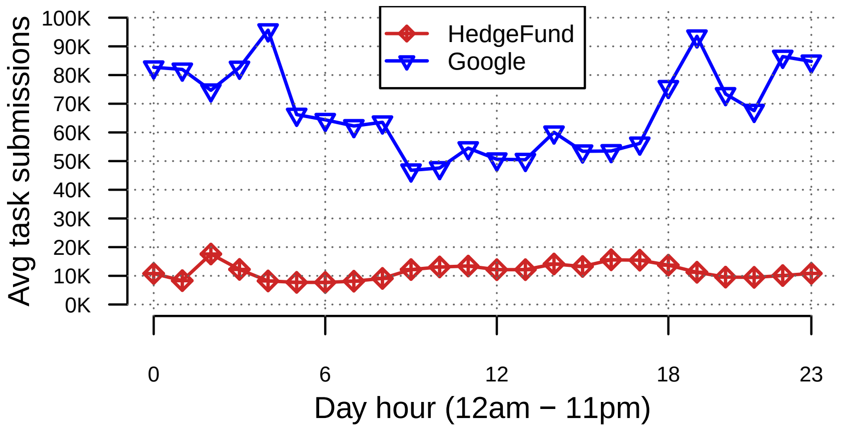

Figure 4: Average task scheduling requests per hour of day in the traced Google and the HedgeFund clusters.

the HedgeFund and Google traces, where tasks arrive every 50-250ms, this can be problematic. Cluster utilization is discussed in detail in the next section.

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, 16, 33, 34, 40]. 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 [16, 31, 32]. Our analysis finds that user resource requests in the LANL and HedgeFund traces are characterized by higher variability than in the Google trace. We also look into job inter-arrival times and how they are approximated in systems' evaluations, and whether starvation due to big jobs is a significant problem [55].

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

Unfortunately, the LANL and HedgeFund 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 [40], Facebook [10], and Twitter [16], 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 low latency during peak utilization. This is usually resolved by over-provisioning hardware and auctioning available resources, since overall utilization will be low. For LANL, clusters are architected to accommodate a predefined set of applications, so high utilization is expected. Further motivation for ensuring high utilization is that it helps justify the federal funding used to procure and operate the large LANL clusters. For the HedgeFund cluster, provisioning for peak utilization is preferred, even though this leads to low overall utilization and resources are not rented out to third parties. This model makes sense in the context of businesses whose revenue may be heavily tied to the responsiveness of their analytics jobs.

In Figure 5, we show the average CPU allocation for every hour of the day for each of our traces. Memory allocation numbers are somewhat lower, albeit follow a similar trend. LANL is exception to this; CPU and memory curves for LANL are identical because resources are allocated at the granularity of physical, identical nodes. First, we note that resources are only over-committed in the Google trace. For the LANL

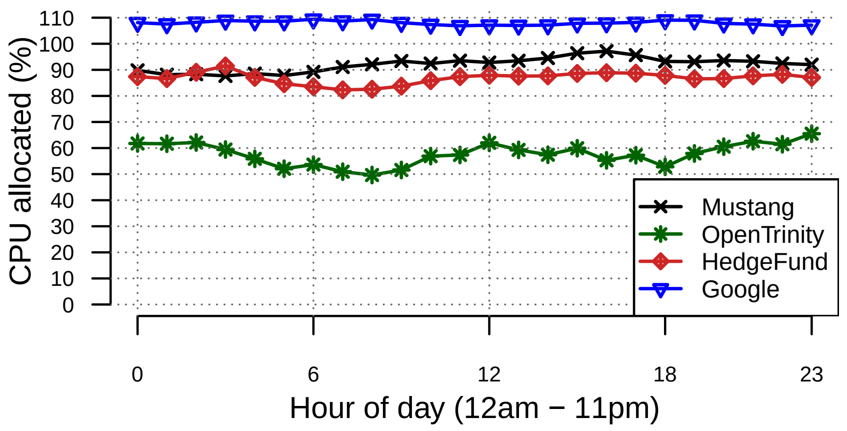

Figure 5: Allocated fraction of cluster CPU per hour of day. Memory utilization numbers are somewhat lower, albeit follow a similar trend.

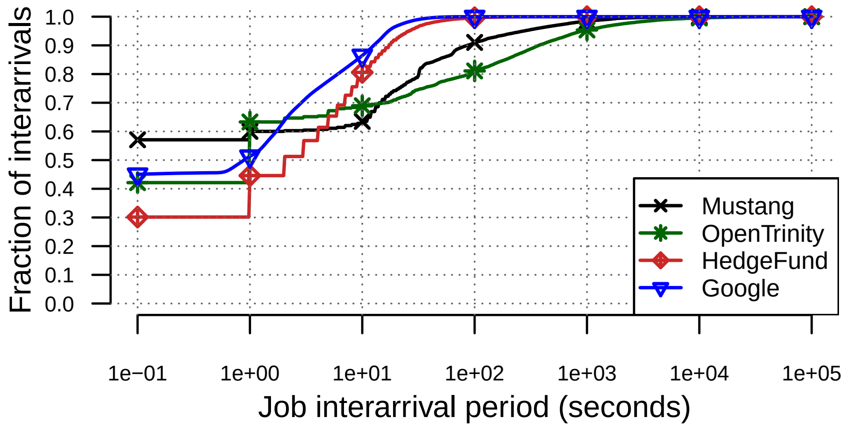

Figure 6: CDF of job interarrival times.

traces, the fact that over-commitment is avoided is likely a sign that jobs are expected to utilize all of the allocated resources, and preventing performance jitter is considered more important. In the case of Trinity, the overall percentage of allocated resources is lower than expected because data collection took place while the machine was operating in OpenScience (beta) mode, so reduced load is not surprising.

Observation 6: *The majority of job interarrivals are sub-second, and do not resemble a Poisson distribution.*

Interarrival periods are a crucial parameter of an experimental setup, as they dictate the load on the system under test. Two common configurations are Poisson-distributed [28] or second-granularity interarrivals [16], and we find that neither characterizes interarrivals accurately. In Figure 6, we show the CDFs for job interarrival period lengths. Our attempts to fit a Poisson distribution on this data have been unsuccessful, as Kolmogorov-Smirnov tests [35] reject the null hypothesis with p-values $< 2.2 \times 10^{-16}$, so caution should be used when a Poisson distribution is assumed. Furthermore, we note that 44-62% of interarrival periods are sub-second, which implies that jobs arrive at a faster rate than previously assumed.

Another common assumption is that big jobs can starve small jobs. Earlier work [28, 55] classifies as

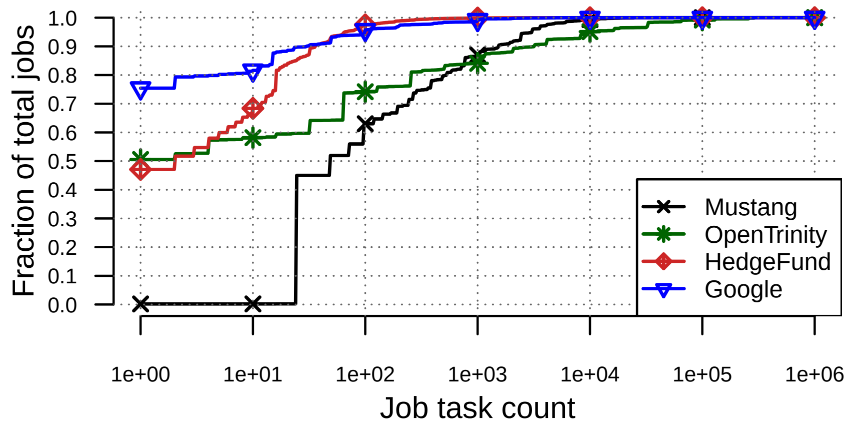

Figure 7: CDF of task counts per job.

small jobs those with 150 tasks or fewer; other jobs are classified as *big*. In Figure 7 we show the CDFs for the number of tasks per job across organizations. For the Google and HedgeFund trace, 96-98% of jobs are small by the definition above, a fraction that is in agreement with data published for Facebook [28]. For the LANL traces, a task is equivalent to a single CPU core and 150-core jobs make up 63-75% of jobs, with the top 5% of jobs exceeding 10^3 (Mustang) and 10^4 (Trinity) cores.

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

An obvious approach to combat cloud resource under-utilization is workload consolidation. To ensure minimal interference, applications are typically profiled and classified according to historical data [16, 31]. Our analysis shows that this approach is likely to be less successful outside the Internet services world. Specifically, we studied the variability of user resource requests for individual users by looking at the Coefficient of Variation[1] (CoV) across user requests. For the Google trace, we note that more than half of the users issue jobs that stay within 2x of the average request in CPU cores. For the LANL traces, on the other hand, 60% of users can deviate by 2-10x of their average request. For the HedgeFund trace, this is true for 80% of users.

Implications: A number of earlier studies of Google [40], Twitter [16], and Facebook [10] data have highlighted the imbalance between resource allocation and utilization. Google tackles this issue by overcommitting resources, however as shown in Figure 5, this is not the case for the LANL and HedgeFund trace. Another proposed solution is Quasar [16], a system that consolidates workloads while guaranteeing a predefined level of QoS. This is achieved by profiling the job at submission, and then classifying it 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, as we showed in Figure 5, which makes it hard to provision resources for profiling. For the HedgeFund and Google traces Quasar may be

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

a better fit, however, at the rate of 2.7 jobs per second (Figure 3), 15 seconds of profiling time [16] 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 to keep up with the average load (assuming the profiling time does not need to be increased for these jobs). Finally, Quasar [16] 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 6), and that it will not increase the profiling cost further.

Another related approach to workload consolidation is provided by TSF [55], 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 big jobs (with 100+ tasks). This would be prohibitive for LANL, where jobs must be scheduled as a whole, and 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, 28].

6 Failure analysis

Job scheduler logs are often analyzed to gain an understanding of job failure characteristics in different environments [9, 18, 22, 23, 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]. We identify similarities and differences in the job characteristics for each of our traces, and describe the implications of our findings on related work.

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 available across traces. 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 by users or administrators. For the HedgeFund trace, no event type other than failure is recorded. We group all these job outcomes as *aborted jobs*, and we collectively refer to failed and aborted jobs as *unsuccessful jobs*. Finally, there is another job outcome category. At LANL, users are required to specify a runtime estimate for each job. This estimate is also treated by the scheduler as a time limit, meaning that once it is exceeded, the job is killed. We refer to these as *timeout jobs*, and present them separately because the entire CPU time taken by the job is unlikely to be wasted for three reasons: (a) HPC jobs may use the timeout cut-off as a stopping criterion, (b) job state may be periodically checkpointed to disk, and (c) a job may complete its work before its time limit and fail to terminate cleanly.

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

In Figure 8, 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 are aborted. According to the trace documentation [57] these jobs could have been aborted by a user or the scheduler, or because jobs they depended on died.

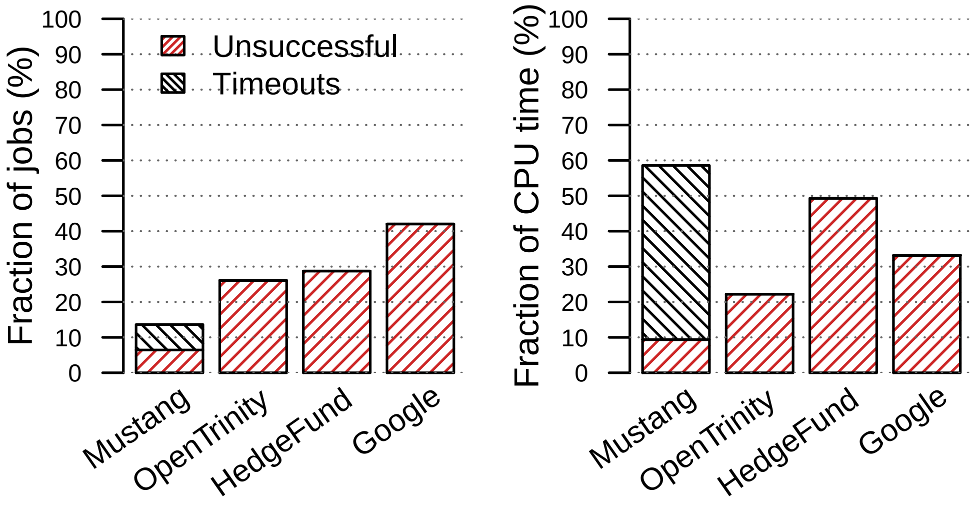

Figure 8: Breakdown of the total number of jobs, as well as CPU time, by job outcome.

As a result, we cannot rule out the possibility that these jobs were linked to a failure that was only visible to the user or scheduler. For this reason, prior work groups all unsuccessful jobs under the "*failed*" label [18]. Instead, we choose to group them under the "*unsuccessful*" label in Figure 8. Another fact that further highlights how blurred the line between failed and aborted jobs can be, is that all unsuccessful jobs in the HedgeFund trace are marked as failed, whether they were aborted or there was an actual failure.

Lastly, we should note that unsuccessful jobs are not equally detrimental to the overall efficiency of all clusters. While the rate of unsuccessful jobs for the HedgeFund trace is similar to the rate of unsuccessful jobs in the LANL traces, each unsuccessful job lasts longer. Specifically, unsuccessful jobs in the LANL traces waste 34-80% less CPU time than in the Google and HedgeFund traces. A final thing to note is that 49% of Mustang's CPU time is allocated to jobs that time out after exceeding their time limit, which warrants further investigation into user behavior.

Observation 9: *For the Google and OpenTrinity traces, most unsuccessful jobs request fewer resources than successful ones. This is untrue for Mustang and the HedgeFund trace.*

In Figure 9, 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. As was already noted in Figure 1, CPU requests for jobs in the Google trace are significantly smaller than requests in all other traces. By separating jobs based on their outcome in Figure 9, we observe that requests from successful jobs in the Google trace are smaller, overall, than those from unsuccessful jobs. This observation has also been made in earlier work [18, 22], but it does not hold for our other traces. CPU requests for successful jobs in the HedgeFund and LANL traces are similar to requests made by unsuccessful jobs[2].

Observation 10: *Success rates decrease for jobs consuming more CPU hours in the Google and HedgeFund traces. The opposite is true in the LANL traces.*

For all traces we analyze, the success of a job is not explicitly attributed to a software or hardware event. This distinction, if available, would be crucial when studying failure rates. For example, we expect that hardware failures are random events whose occurrence roughly approximates some frequency based on

[2]This trend is opposite to what is seen in older HPC job logs [58]. And because many of these traces were also collected through SLURM and MOAB, we do not expect this discrepancy to be due to semantic differences in the way failure is defined across HPC traces.

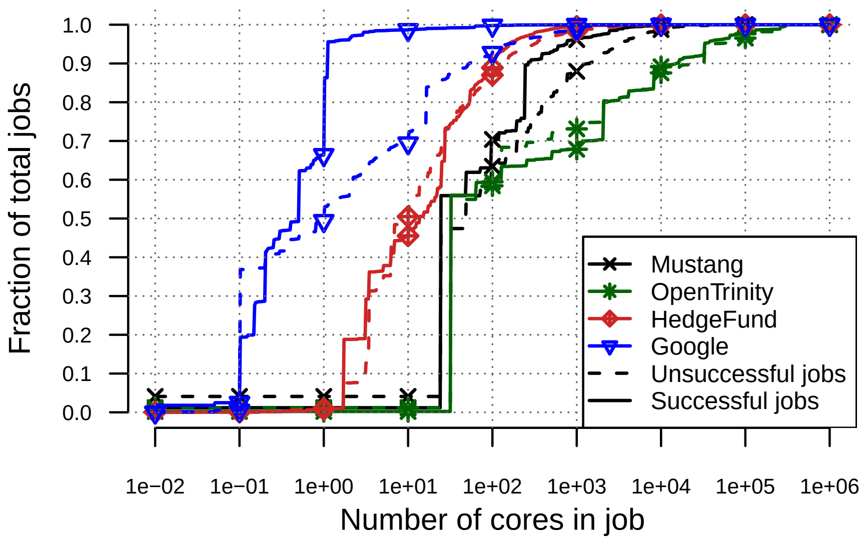

Figure 9: CDFs of job sizes (in CPU cores) for unsuccessful and successful jobs.

the components' MTBF (Mean Time Between Failure) ratings. As a result, jobs that are larger and/or longer, would be more likely to fail. In Figure 10, 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 taking more CPU hours in the Google and HedgeFund traces, but they are stable or decrease for both LANL clusters. One possible explanation for this is that larger, longer jobs at LANL are more carefully planned and tested. Another possible explanation is due to semantic differences (that we may be unaware of) in the way success and failure are defined.

Implications. The majority of papers analyzing the characteristics of job and task failures in the Google trace build failure prediction models that assume the existence of the trends we have shown on success rates, and the resource consumption of failed jobs. Chen et al. [9] highlight the difference in resource consumption between unsuccessful and successful jobs, and El-Sayed et al. [18] note that this is the second most influential predictor (next to early task failures) for their random forest-based failure prediction models. As we have shown in Figure 10, unsuccessful jobs may not be linked to resource consumption in other traces at all. Another predictor highlighted in both studies is job re-submissions. It is noted that successful jobs are re-submitted fewer times than unsuccessful ones. 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 "wasted" due to unsuccessful jobs increases with job runtime [18, 23].

7 A case study on dataset pluralism

Evaluating systems against multiple traces enables researchers to identify practical sensitivities of their design and implementation. It also allows claims on the wide applicability of novel techniques to be grounded on data. We demonstrate this through a case study on JVuPredict, the job runtime[3] predictor module of the JamaisVu [51] cluster scheduler. Our evaluation of JVuPredict with all four traces revealed the predictive power of logical job names and consistent user behavior in workload traces, and the difficulty of obtaining accurate runtime predictions in systems that cannot provide enough information to the scheduler to identify

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

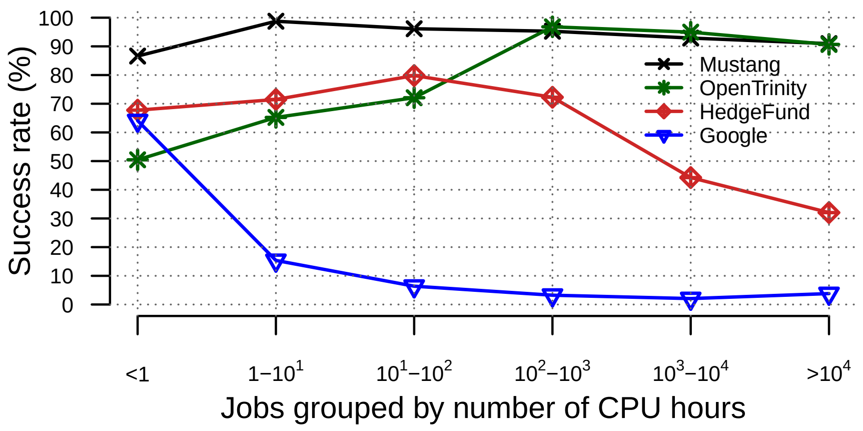

Figure 10: Success rates for jobs grouped by number of CPU hours.

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

JamaisVu [51] is an end-to-end cluster scheduling system that automatically decides how to use historic data from the cluster's operation to generate and exploit job runtime predictions. Accurate knowledge of job runtimes allows a scheduler to pack jobs more aggressively in a cluster [12, 19, 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].

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 runtime 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 [29], Jockey [21], and ARIA [53]. For periodic jobs, simple history-based predictions can also work well [12]. But these approaches are still inadequate for consolidated clusters without a known structure or history.

JVuPredict, the runtime prediction module for JamaisVu, 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. To do this, it uses the 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.

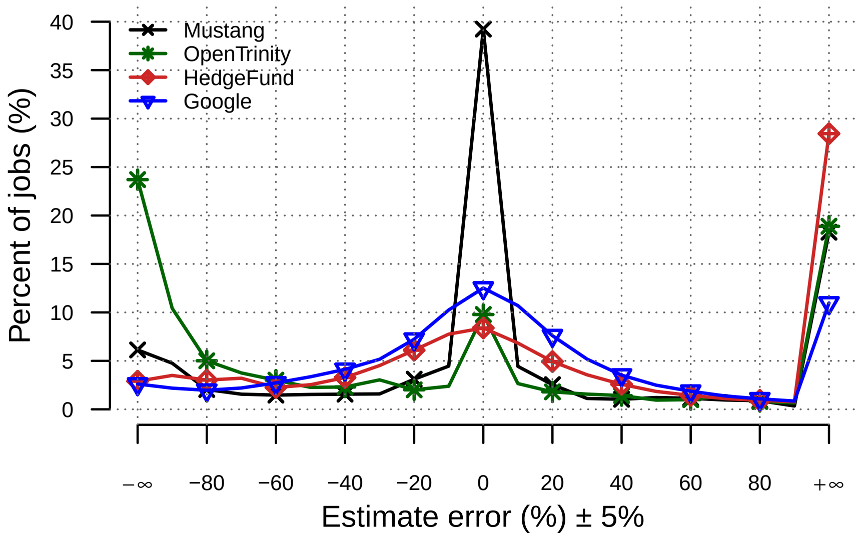

Figure 11: Accuracy of JVuPredict predictions of runtime estimates, for all four traces.

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 three new 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 11 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 values 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 (we elaborate more on this in Section 8.1). Instead, this field is either assigned a unique value for every job, or entirely omitted from the trace. All three 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 HedgeFund 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 HedgeFund workload achieves identical performance to Google: 31% of job runtimes are underestimated, and 14.8% 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 HedgeFund and Mustang results, with the ones achieved for the Google trace, suggests that JamaisVu would also perform well under these workloads.

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 in 2 of 3 traces. Specifically, the user ID and CPU core count for every job seem to perform best for both HedgeFund and the Mustang trace.

8 Discussion: Limiting factors of traces

Working with traces often forces researchers to make key assumptions as they interpret the data, in order to cope with missing information. An example of this, which we encountered in Section 6, are ad-hoc classifications of the severity of failures when information on root causes is missing. Missing information leaves aspects of the data open to interpretation, which is usually necessary. This necessity stems from restrictive anonymization of data features in order to protect the organization releasing the data. We discuss our experience with this process in Section 8.1. Another common assumption is that a trace is representative of the overall system workload. This is a reasonable assumption for longer traces, but the Google trace covers 29 days; we examine whether this is sufficient to characterize the workload in our other traces in Section 8.2 (hint: it's not).

8.1 Data anonymization

There is an inherent trade-off to anonymization. Strict anonymization makes is harder to glean confidential or proprietary information from the data [37, 41]. At the same time, it reduces the value of the data by altering important characteristics, such as the distribution of a feature's values. We find that anonymization techniques are primarily decided based on their implementation complexity, even if they reduce the overall value of the data. Two such techniques are: tokenizing the data by replacing all occurrences of a value with the same token, or removing features altogether.

Two fields are especially affected by these anonymization techniques. The first is the *job name*. In the Google trace, this field is a hash of the program's name and parameters, making it easy to track job re-runs. Anonymizing this field proved to be time-consuming for the people processing the HedgeFund and LANL traces so it was either removed, or replaced by a unique value per job. The difficulty in preserving this field stemmed from the complexity of user job scripts submitted to the scheduler, which often contain conditionals that make it hard to distinguish which program was used. The second field is the pair of the *user*

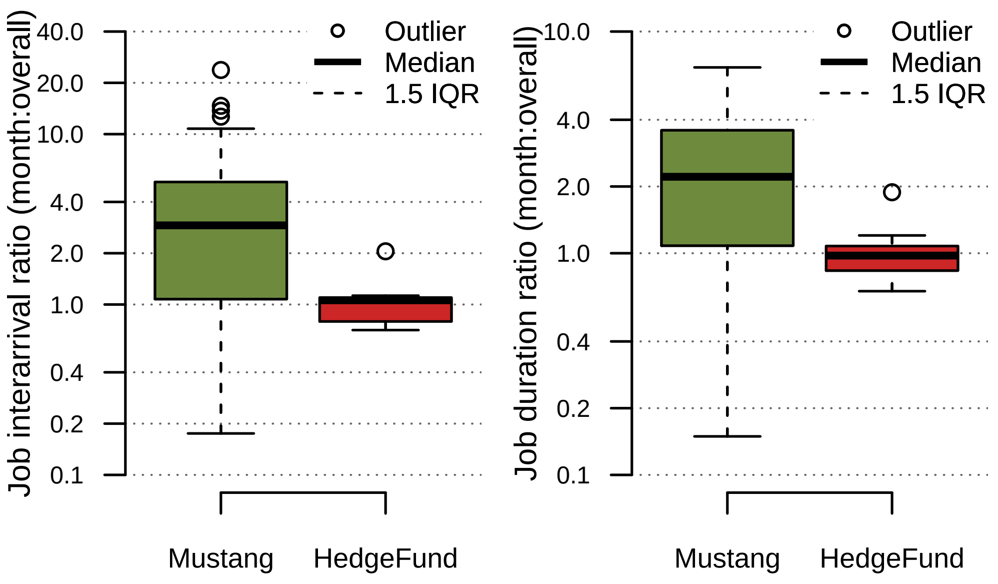

Figure 12: *Is a month representative of the overall workload?* The boxplots show distributions of the average job inter-arrival period (left) and duration (right) per month, normalized by the trace's overall average. The boxplot whiskers are defined to be 1.5 times the Inter-Quartile Range of the distribution (standard Tukey boxplots).

and group IDs, which got anonymized through tokenization. This process removed information that could be used to identify a user's role in the system such as administrators, who tend to display unique behavior that is typically uncharacteristic of typical users. For the LANL traces we have confirmed administrator IDs and marked them accordingly.

8.2 Trace Length

A common (unwritten) assumption when using or analyzing a trace, is that it sufficiently represents the workload of the environment wherein it was collected. However, the Google trace covers only a 29-day period, while the traces we introduce in this paper are 3-60 times longer. We were unsure whether the trace length makes a difference beyond the 29-day mark of Google, so next we examine how representative individual 29-day periods can be for our HedgeFund and Mustang traces, especially given that the Mustang trace covers the entire lifetime of the cluster.

Our experiment consisted of dividing our traces in 29-day "months". 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 the job sizes, durations, and interarrival periods. Figure 12 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 in the box represents the median, and the whiskers are defined as 1.5 times the Inter-Quartile Range (IQR) of the distribution, or the furthest data point if no outliers exist (shown in circles here). We see that individual months can vary significantly for the Mustang trace, and they can differ somewhat less across months in the HedgeFund trace as well. More specifically, the average job interarrival of a given month can be 0.7-2.0x the value of the overall average in the HedgeFund 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 HedgeFund trace, and 0.1-6.9x of the average in the Mustang trace. Overall, our results conclusively show that both interarrivals and job durations display significant differences from month to month.

9 Related Work

The Parallel Workloads Archive (PWA) [20] hosts the largest collection of public HPC traces. At the time of this writing, 38 HPC traces have been collected between 1993 and 2015, and contributed mainly by government organizations and universities. Our HPC traces complement this collection. The Mustang trace is almost two times longer in duration than the longest publicly available trace, covers the entire lifetime of the cluster, and contains four times as many jobs. We should note that only three traces in the archive originate in machines that are similar or larger in size to Mustang, and the trace we release is three times longer than the longest of those traces. More importantly, the Mustang trace covers the entire lifetime of the machine, enabling complete longitudinal analyses. The Trinity trace is also complementary to existing traces, as it is collected on a machine almost two times bigger than Argonne National Lab's Intrepid, which is the largest supercomputer with a publicly available trace 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. As expected the clusters studied handle significantly less traffic than the Google and HedgeFund clusters we examine. Interestingly, a sizable fraction of interarrival periods for individual jobs are longer than 100 seconds, which is more reminiscent of our HPC workloads. At the same time, however, 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. Despite that, we find 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 our HedgeFund cluster, while Facebook handles upwards of a thousand submissions per hour, which is more related to traffic in the Google cluster.

Other studies that look at private clusters focus on Virtual Machine workloads. Shen et al. [47] analyze two datasets consisting 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 a minute to a day, and includes storage and CPU measurements, as well as repair 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, in addition to low job size variability per tenant.

Conclusion

We have introduced and analyzed three cluster job scheduler traces: a private cluster trace, and traces from two HPC clusters. We publicly release both HPC cluster traces, which we expect to be of interest to researchers due to their unique characteristics: the Mustang trace covers the entire lifetime of the cluster it was collected in, and is the longest publicly available cluster trace to date; the Trinity trace represents the largest supercomputer (in terms of CPU core count), with a publicly available trace.

Our analysis showed that the workload in the private cluster approximates more closely the characteristics of the HPC traces we studied, rather than the workload in the popular Google cluster trace, which is surprising. This observation holds across many facets of the workload: job sizes and duration, user request variability, and the fact that resources are not over-committed to jobs. We also listed differences and similarities across other facets of the workload, and work from the literature that is affected by our findings.

Finally, we demonstrated the importance of using multiple datasets in the evaluation of new research. For the job runtime prediction module of the JamaisVu cluster scheduler, we show that using more than one trace allowed us to reliably rank data features by predictive power. We hope that by providing traces from contemporary systems, we will enable researchers to better understand the dependence and sensitivity of their techniques to different workload characteristics.

References

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

[2] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective Straggler Mitigation: Attack of the Clones. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 185–198, Lombard, IL, 2013. USENIX.

[3] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 285–300, Broomfield, CO, 2014. USENIX Association.

[4] Anton Burtsev, Prashanth Radhakrishnan, Mike Hibler, and Jay Lepreau. Transparent checkpoints of closed distributed systems in emulab. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 173–186, New York, NY, USA, 2009. ACM.

[5] Ignacio Cano, Srinivas Aiyar, and Arvind Krishnamurthy. Characterizing Private Clouds: A Large-Scale Empirical Analysis of Enterprise Clusters. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 29–41, New York, NY, USA, 2016. ACM.

[6] Marcus Carvalho, Walfredo Cirne, Franciso Brasileiro, and John Wilkes. Long-term SLOs for reclaimed cloud computing resources. In *ACM Symposium on Cloud Computing (SoCC)*, pages 20:1–20:13, Seattle, WA, USA, 2014.

[7] Chen Chen, Wei Wang, Shengkai Zhang, and Bo Li. Cluster Fair Queueing: Speeding up Data-Parallel Jobs with Delay Guarantees. In *Proceedings of the IEEE International Conference on Computer Communications*, IEEE INFOCOM 2017, May 2017.

[8] S. Chen, M. Ghorbani, Y. Wang, P. Bogdan, and M. Pedram. Trace-Based Analysis and Prediction of Cloud Computing User Behavior Using the Fractal Modeling Technique. In *2014 IEEE International Congress on Big Data*, pages 733–739, June 2014.

[9] X. Chen, C. D. Lu, and K. Pattabiraman. Failure Analysis of Jobs in Compute Clouds: A Google Cluster Case Study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 167–177, Nov 2014.

[10] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive Analytical Processing in Big Data Systems: A Cross-industry Study of MapReduce Workloads. *Proc. VLDB Endow.*, 5(12):1802–1813, August 2012.

[11] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *ACM SIGOPS operating systems review*. ACM, 2017.

[12] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. Reservation-based Scheduling: If You're Late Don't Blame Us! In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 2:1–2:14, New York, NY, USA, 2014. ACM.

[13] Delft University of Technology. The Grid Workloads Archive. Online repository, April 2016. Posted at http://gwa.ewi.tudelft.nl/.

[14] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. Job-aware Scheduling in Eagle: Divide and Stick to Your Probes. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 497–509, New York, NY, USA, 2016. ACM.

[15] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid Datacenter Scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 499–510, Santa Clara, CA, 2015. USENIX Association.

[16] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, New York, NY, USA, 2014. ACM.

[17] Eric Eide, Leigh Stoller, and Jay Lepreau. An Experimentation Workbench for Replayable Networking Research. In *Proceedings of the 4th USENIX Conference on Networked Systems Design &#38; Implementation*, NSDI'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[18] N. El-Sayed, H. Zhu, and B. Schroeder. 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)*, pages 1333–1344, June 2017.

[19] Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. *Parallel Job Scheduling — A Status Report*, pages 1–16. Springer Berlin Heidelberg, 2005.

[20] Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing*, 74(10):2967 – 2982, 2014.

[21] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 99–112, New York, NY, USA, 2012. ACM.

[22] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu. An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment. *IEEE Transactions on Emerging Topics in Computing*, 2(2):166–180, June 2014.

[23] Bogdan Ghit and Dick Epema. 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*, HPDC '17, pages 105–116, New York, NY, USA, 2017. ACM.

[24] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 99–115, GA, 2016. USENIX Association.

[25] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, 2017. USENIX Association.

[26] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[27] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[28] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling Jobs Across Geo-distributed Datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 111–124, New York, NY, USA, 2015. ACM.

[29] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[30] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: Fair Scheduling for Distributed Computing Clusters. In *Proceedings of the 22nd Symposium on Operating Systems Principles*, SOSP '15, pages 261–276. ACM, October 2009.

[31] Jacob Leverich and Christos Kozyrakis. Reconciling High Server Utilization and Sub-millisecond Quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 4:1–4:14, New York, NY, USA, 2014. ACM.

[32] J. Liu, H. Shen, and L. Chen. CORP: Cooperative Opportunistic Resource Provisioning for Short-Lived Jobs in Cloud Systems. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 90–99, Sept 2016.

[33] Jiuyue Ma, Xiufeng Sui, Ninghui Sun, Yupeng Li, Zihao Yu, Bowen Huang, Tianni Xu, Zhicheng Yao, Yun Chen, Haibin Wang, Lixin Zhang, and Yungang Bao. 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*, ASPLOS '15, pages 131–143, New York, NY, USA, 2015. ACM.

[34] Paul Marshall, Kate Keahey, and Tim Freeman. Improving Utilization of Infrastructure Clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 205–214, Washington, DC, USA, 2011. IEEE Computer Society.

[35] Frank J. Massey Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[36] Esteban Meneses, Xiang Ni, Terry Jones, and Don Maxwell. Analyzing the Interplay of Failures and Workload on a Leadership-Class Supercomputer. In *Cray User Group Conference (CUG)*, April 2015.

[37] Arvind Narayanan and Vitaly Shmatikov. Robust De-anonymization of Large Sparse Datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 111–125, Washington, DC, USA, 2008. IEEE Computer Society.

[38] Office of Science (DOE-SC) and the National Nuclear Security Administration (NNSA), U.S. Department of Energy. Exascale Computing Project. `https://exascaleproject.org/`.

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

[40] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[41] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *CloudMAN*, Maui, HI, USA, 2012.

[42] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's Adolescence: An Analysis of Hadoop Usage in Scientific Workloads. *Proc. VLDB Endow.*, 6(10):853–864, August 2013.

[43] A. Ros, L. Y. Chen, and W. Binder. Predicting and Mitigating Jobs Failures in Big Data Clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 221–230, May 2015.

[44] A. Ros, L. Y. Chen, and W. Binder. 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*, pages 207–218, June 2015.

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

[46] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.

[47] S. Shen, V. v. Beek, and A. Iosup. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 465–474, May 2015.

[48] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A Chien, Paul Coteus, Nathan A DeBardeleben, Pedro C Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.

[49] The Apache Software Foundation. Apache Spark. `https://spark.apache.org/`.

[50] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das. Phoenix: A Constraint-Aware Scheduler for Heterogeneous Datacenters. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 977–987, June 2017.

[51] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A. Kozuch, and Gregory R. Ganger. JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes. Technical Report CMU-PDL-16-104, Carnegie Mellon University, September 2016.

[52] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. TetriSched: Global Rescheduling with Adaptive Plan-ahead in Dynamic Heterogeneous Clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 35:1–35:16, New York, NY, USA, 2016. ACM.

[53] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, New York, NY, USA, 2011. ACM.

[54] Abhishek Verma, Madhukar Korupolu, and John Wilkes. Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014*, pages 48–56, 11 2014.

[55] W. Wang, B. Li, B. Liang, and J. Li. Multi-resource Fair Sharing for Datacenter Jobs with Placement Constraints. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1003–1014, Nov 2016.

[56] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[57] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[58] Yulai Yuan, Yongwei Wu, Qiuping Wang, Guangwen Yang, and Weimin Zheng. Job failures in high performance computing systems: A large-scale empirical study. *Computers & Mathematics with Applications*, 63(2):365 – 377, 2012.