Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis

Charles Reiss
University of California, Berkeley
charles@eecs.berkeley.edu

Alexey Tumanov
Carnegie Mellon University
atumanov@cmu.edu

Gregory R. Ganger
Carnegie Mellon University
ganger@ece.cmu.edu

Randy H. Katz
University of California, Berkeley
randy@eecs.berkeley.edu

Michael A. Kozuch
Intel Labs
michael.a.kozuch@intel.com

ABSTRACT

To better understand the challenges in developing effective cloud-based resource schedulers, we analyze the first publicly available trace data from a sizable multi-purpose cluster. The most notable workload characteristic is heterogeneity: in resource types (e.g., cores:RAM per machine) and their usage (e.g., duration and resources needed). Such heterogeneity reduces the effectiveness of traditional slot- and core-based scheduling. Furthermore, some tasks are constrained as to the kind of machine types they can use, increasing the complexity of resource assignment and complicating task migration. The workload is also highly dynamic, varying over time and most workload features, and is driven by many short jobs that demand quick scheduling decisions. While few simplifying assumptions apply, we find that many longer-running jobs have relatively stable resource utilizations, which can help adaptive resource schedulers.

Categories and Subject Descriptors

D.4.7 [**Operating systems**]: Organization and design—*Distributed systems*

General Terms

Measurement

1. INTRODUCTION

Consolidation of differing tenants' processing demands into a common shared infrastructure plays a central role in cloud computing. This has the advantages of statistical multiplexing of physical resource use and centralized asset management, as well as workload-specific benefits, such as sharing of common datasets and intermediate computational results. Nevertheless, consolidated environments present new resource management challenges. In particular, unlike traditional scientific and supercomputing environments, cloud computing represents much higher diversity in workload profiles, spanning the patterns of scientific computing, elastic long-running Internet services, data analytics of many scales, software development and test, engineering optimization, and others.

Although there have been many proposals for new resource management approaches for cloud infrastructures [1, 12, 23, 25], and effective resource management is a major challenge for the leading cloud infrastructure operators (e.g., Google, Microsoft, Amazon), little is understood about the details of the underlying workloads and their real-world operational demands. To facilitate this understanding, Google recently released a substantial cluster usage dataset to give researchers some visibility into real workloads [24].

SOCC'12, October 14-17, 2012, San Jose, CA USA

We analyze this trace to reveal several insights that we believe are important for designers of new resource scheduling systems intended for multi-purpose clouds. Our analysis exposes and characterizes a number of useful, and sometimes surprising, issues to be faced by such schedulers, including:

Machine and workload heterogeneity and variability. Unlike more traditional application and organization-specific clusters, consolidated cloud environments are likely to be constructed from a variety of machine classes, representing different points in the configuration space of processing to memory to storage ratios. Several generations of machines, with different specifications, are likely to be encountered, as the underlying machine types evolve over time with respect to economically attractive price-performance. A subset of machines with specialized accelerators, such as graphics processors, may also be available in limited numbers. Finally, as the workload spans multiple organizations, it is likely to be inherently more diverse in its resource demands than one from any single organization. Should such a high degree of heterogeneity and variability in workload demand be encountered, it will significantly complicate the method by which units of resources are allocated to achieve good utilization without excessive waiting for resources.

Highly dynamic resource demand and availability. The greater the number of organizational entities that share a common computing infrastructure, each executing its own particular mix of tasks, the more dynamic their aggregated demand and arrival patterns are likely to be. Such an environment may be characterized by a large dynamic range of resource demands with high variation over short time intervals. This would have several scheduling consequences, including rapid task scheduling decision making, revision of previous assignment decisions, and accommodation of dynamic resource requests from longer-running jobs that experience difficult to predict interference for resources over time.

Predictable, but poorly predicted, resource needs. Because of the high degree of heterogeneity and dynamism in the workloads, one would expect that it would be difficult to achieve accurate visibility into future resource demands, thus further complicating the resource allocation problem. Nevertheless, deeper patterns of resource usage may exist, for example, if long running tasks with resource consumption patterns that can be learned form a significant portion of the workload. This places a demand on resource schedulers to maintain more sophisticated time-based models of resource usage, at least for some tasks that can be characterized as long running.

Resource class preferences and constraints. Task specifications may include constraints or preferences for particular classes of machines on which they may run, to obtain machines with specific configurations or with specialized accelerators. Thus, the underlying machines are not fully fungible, and there may be utilization and task latency consequences when allocating a machine with specialized capabilities to a task that does not need it (and conversely, when allocating a task able to exploit specialized capabilities to a machine without them). Such preferences and constraints further complicate the scheduling decisions.

As we shall see the remainder of the paper, these hypotheses hold true for our analyzed Google trace. Section 2 introduces the terminology of cloud resource allocation and related work. The analysis of the trace along dimensions of heterogeneity and dynamicity are addressed in Sections 3 and 4, respectively. We show that the trace has a wider variety of scheduling requirements than are likely to be handled by common schedulers. Section 5 focuses on what the trace tells us about the predictability of resource usage. The implications of resource preferences and constraints are addressed in Section 6. Section 7 discusses our conclusions and future work.

2. BACKGROUND

Trace studies play an extensive role in understanding systems challenges, including those facing schedulers. Since most previous clusters have not been faced with the diverse workloads of multi-purpose clouds, most cluster trace analyses report on more homogeneous workloads [20]. This section overviews prior work and the Google trace analyzed in this paper.

Previous work. Most studied cluster workloads have fallen into one of three broad categories:

- long-running services (example analyses: [2, 3, 16]): servers (such as web servers) that require a certain amount of resources (usually CPU time) to achieve acceptable performance and run indefinitely;
- DAG-of-task systems (example analyses: [4, 11, 5]):
  MapReduce[7]- or Dryad[10]-like systems that run many independent short (seconds to minutes) tasks that are assumed to be CPU-bound or I/O-bound; and
- high-performance (or throughput) computing (example analyses: [14, 27, 9, 15]): batch queuing systems that typically run CPU-bound programs, can usually tolerate substantial wait times, and often require many machines simultaneously for a long period of time (hours to days).

This paper examines a workload that is a mix of these types; consequently, it has a mix of their quirks. Other analyses of the same workload include [13, 8, 26], which were conducted in parallel with this work.

Each of these categories brings different challenges to a cluster scheduler. For example, long-running interactive services have external client loads, which they distribute among their instances independently from the cluster scheduler, and are concerned with metrics like instantaneous availability and tail request response times. Though it may continuously monitor such applications, scheduler activity is rare:

demand is usually stable, and when it is not, the necessary configuration changes do not require rescheduling most instances of the service. The scheduler has plenty of time to measure the behavior of its long-running applications and few kinds of behavior to understand. DAG-of-task systems, on the other hand, may involve frequent cluster scheduler interactions, as interactive data analyses are now common, making scheduler latency important. Used primarily for data-parallel computation, such systems are often scheduled assuming that each task will have similar resource needs, enabling approaches based on fixed-sized slots for each machine and not bothering with on-line monitoring or migration. HPC-like environments often have infrequent cluster scheduling, where the scheduler only acts as long-lived jobs start, but with a focus on job runtimes and overall cluster utilization.

Existing cluster schedulers are not designed to cope with a large-scale mixture of these and other application needs, let alone with the additional challenges (e.g., extensive use of constraints [21]) not found in most current environments. Of course, Google must have a scheduler for the system from which the Google trace data was collected, but their ongoing effort to create a new scheduler [23] suggests that new approaches are needed.

Google trace. The Google cluster trace captures a range of behaviors that includes all of the above categories, among others. The "trace" consists of what could be considered several concurrent traces for a month of activity in a single ∼12K machine cluster. It includes a trace of all cluster scheduler requests and actions, a trace of per-task resource usage over time, and a trace of machine availability. The trace describes hundreds of thousands of *jobs*, submitted by *users*. Each *job* is composed of one to tens of thousands of *tasks*, which are programs to be executed on an available machine. These tasks are not gang-scheduled, but are usually executed simultaneously.

Each task is specified with various parameters, including priority, resource request (estimated maximum RAM and CPU needed), and, sometimes, constraints (e.g., do not run on a machine without an external IP address). These parameters span a wider range than seen in traditional cluster workloads; we examine them in more detail in later sections. For each task, the trace also indicates each time it is submitted, assigned to a machine, or descheduled; these records allow us to examine task and job durations and identify unusual task and scheduler behavior. The trace also includes per-assigned-task resource usage information every five minutes from every machine.

The trace lacks precise information about the purpose of jobs and configuration of machines. The trace does include identifiers for jobs, user names, machine platforms, and configurations; however, these identifiers have been obfuscated by the trace providers, so we are only able to identify the distributions of job and user names and machine characteristics across the trace. Thus, we cannot use application or user names to infer what's running. Instead, we will determine application types from tasks' scheduling parameters and resource usage. Similarly, resource (RAM and CPU) information is provided in normalized units, so we are able to accurately compare resource usage, request and capacity measurements, but we cannot report the exact number of cores or amount of memory available or used on any machine.

Analysis of the Google cluster trace has also been conducted independently by three other groups of researchers. Di et al. [8] focus their analysis on comparing the Google trace characteristics to those of Grid/HPC systems. Liu and Cho [13] study machine properties and their lifecycle management, workload behavior, and resource utilization. Zhang et al. [26] study the trace from the perspective of energy-aware provisioning and energy-cost minimization, using it to motivate dynamic capacity provisioning and the challenges associated with it.

3. HETEROGENEITY

The traced 'cloud computing' workload is much less homogeneous than researchers often assume. It appears to be a mix of latency-sensitive tasks, with characteristics similar to web site serving, and less latency-sensitive programs, with characteristics similar to high-performance computing and MapReduce workloads. This heterogeneity will break many scheduling strategies that might target more specific environments. Assumptions that machines or tasks can be treated equally are broken; for example, no scheduling strategy that uses fixed-sized 'slots' or uniform randomization among tasks or machines is likely to perform well.

3.1 Machine types and attributes

The cluster machines are not homogeneous; they consist of three different platforms (the trace providers distinguish them by indicating "the microarchitecture and chipset version" [18]) and a variety of memory/compute ratios. The configurations are shown in Table 1. Exact numbers of CPU cores and bytes of memory are unavailable; instead, CPU and memory size measurements are normalized to the configuration of the largest machines. We will use these units throughout this paper. Most of the machines have half of the memory and half the CPU of the largest machines.

This variety of configurations is unlike the fully homogeneous clusters usually assumed by prior work. It is also distinct from prior work that focuses on clusters where some machines have fundamentally different types of computing hardware, like GPUs, FPGAs, or very low-power CPUs. The machines here differ in ways that can be explained by the machines being acquired over time using whatever configuration was most cost-effective then, rather than any deliberate decision to use heterogeneous hardware.

In addition to the CPU and memory capacity and microarchitecture of the machines, a substantial fraction of machine heterogeneity, from the scheduler's perspective, comes from "machine attributes". They are obfuscated <key,value> pairs, with a total of 67 unique machine attribute keys in the cell. The majority of those attributes have fewer than 10 unique values ever used by any machine. That is consistent with [21], where the only machine attributes with possible values exceeding 10 were number of disks and clock speed. In this trace, exactly

Number of machines	Platform	CPUs	Memory
6732	B	0.50	0.50
3863	B	0.50	0.25
1001	B	0.50	0.75
795	C	1.00	1.00
126	A	0.25	0.25
52	B	0.50	0.12
5	B	0.50	0.03
5	B	0.50	0.97
3	C	1.00	0.50
1	B	0.50	0.06

Table 1: Configurations of machines in the cluster. CPU and memory units are linearly scaled so that the maximum machine is 1. Machines may change configuration during the trace; we show their first configuration.

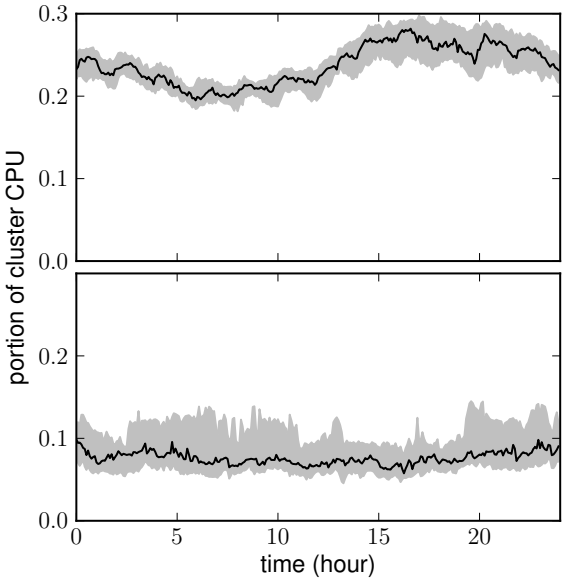


Figure 1: Normal production (top) and lower (bottom) priority CPU usage by hour of day. The dark line is the median and the grey band represents the quartiles.

10 keys are used with more than 10 possible values. 6 of these keys are used as constraints. One of them has 12569 unique values — an order of magnitude greater than all others combined, which roughly corresponds to the number of machines in the cluster. Based on further analysis in Section 6 and [21], these attributes likely reflect a combination of machine configuration and location information. Since these attributes are all candidates for task placement constraints (discussed later), their number and variety are a concern for a scheduler. Scheduler designs can no longer consider heterogeneity of hardware an aberration.

3.2 Workload types

One signal of differing job types is the priority associated with the tasks. The trace uses twelve task priorities (numbered 0 to 11), which we will group into three sets: production (9–11), middle (2–8), and gratis (0–1). The trace providers tell us that latency-sensitive tasks (as marked by another task attribute) in the production priorities should not be "evicted due to over-allocation of machine resources" [18] and that users of tasks of gratis priorities are charged substantially less for their resources.

The aggregate usage shows that the production priorities represent a different kind of workload than the others. As shown in Figure 1, production priorities account for more resource usage than all the other priorities and have the clearest daily patterns in usage (with a peak-to-mean ratio of 1.3). As can be seen from Figure 2, the production priorities also include more long-duration jobs, accounting for a majority of all jobs which run longer than a day even though only 7% of all jobs run at production priority. Usage at the lowest priority shows little

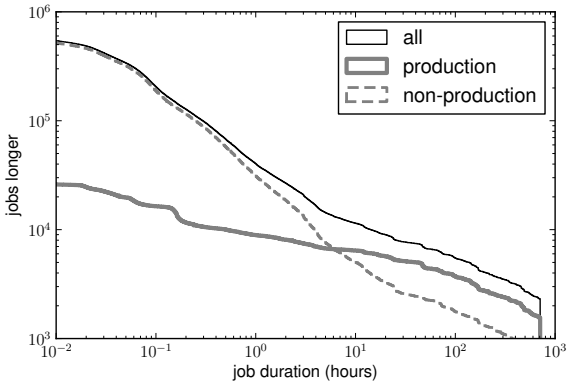


Figure 2: Log-log scale inverted CDF of job durations. Only the duration for which the job runs during the trace time period is known; thus, for example, we do not observe durations longer than around 700 hours. The thin, black line shows all jobs; the thick line shows production-priority jobs; and the dashed line shows non-production priority jobs.

such pattern, and this remains true even if short-running jobs are excluded.

These are clearly not perfect divisions of job purpose — each priority set appears to contain jobs that behave like user-facing services would and large numbers of short-lived batch-like jobs (based on their durations and utilization patterns). The trace contains no obvious job or task attribute that distinguishes between types of jobs besides their actual resource usage and duration: even 'scheduling class', which the trace providers say represents how latency-sensitive a job is, does not separate short-duration jobs from long-duration jobs. Nevertheless, the qualitative difference in the aggregate workloads at the higher and lower priorities shows that the trace is both unlike batch workload traces (which lack the combination of diurnal patterns and very long-running jobs) and unlike interactive service (which lack large sets of short jobs with little pattern in demand).

3.3 Job durations

Job durations range from tens of seconds to essentially the entire duration of the trace. Over 2000 jobs (from hundreds of distinct users) run for the entire trace period, while a majority of jobs last for only minutes. We infer durations from how long tasks are active during the one month time window of the trace; jobs which are cut off by the beginning or end of the trace are a small portion ($< 1\%$) of jobs and consist mostly of jobs which are active for at least several days and so are not responsible for us observing many shorter job durations. These come from a large portion of the users, so it is not likely that the workload is skewed by one particular individual or application. Consistent with our intuition about priorities correlating with job types, production priorities have a much higher proportion of long-running jobs and the 'other' priorities have a much lower proportion. But slicing the jobs by priority or 'scheduling class' (which the trace providers say should reflect how latency-sensitive a job is) reveals a similar heavy-tailed distribution shape with a large number of short jobs.

3.4 Task shapes

Each task has a resource request, which should indicate the amount of CPU and memory space the task will require. (The requests are intended to represent the submitter's predicted "maximum" usage for the task.) Both the amount of the resources requested and the amount actually used by tasks varies by several orders of magnitude; see Figures 3 and 4, respectively. These are not just outliers. Over 2000 jobs request less than 0.0001 normalized units of memory per task, and over 8000 jobs request more than 0.1 units of memory per task. Similarly, over 70000 jobs request less than 0.0001 units of CPU per task, and over 8000 request more than 0.1 units of CPU. Both tiny and large resource requesting jobs include hundreds of distinct users, so it is not likely that the particularly large or small requests are caused by the quirky demands of a single individual or service.

We believe that this variety in task "shapes" has not been seen in prior workloads, if only because most schedulers simply do not support this range of sizes. The smallest resource requests are likely so small that it would be difficult for any VM-based scheduler to run a VM using that little memory. (0.0001 units would be around 50MB if the largest machines in the cluster had 512GB of memory.) Also, any slot-based scheduler, which includes all HPC and Grid installations we are aware of, would be unlikely to have thousands of slots per commodity machine.

The ratio between CPU and memory requests also spans a large range. The memory and CPU request sizes are correlated, but weakly (linear regression $R^2 \approx 0.14$). A large number jobs request 0 units of CPU — presumably they require so little CPU they can depend on

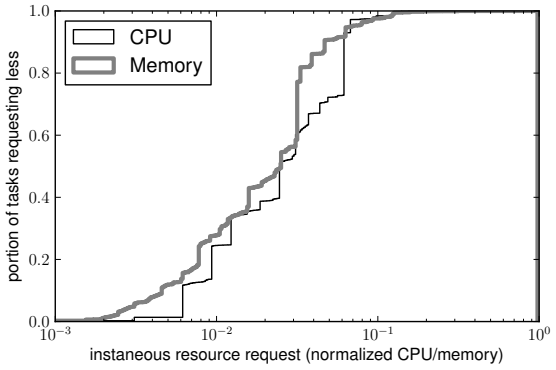


Figure 3: CDF of instantaneous task requested resources. (1 unit = max machine size.) These are the raw resource requests in the trace; they do not account for task duration.

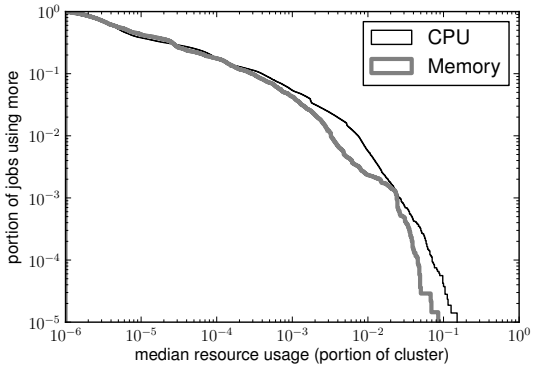


Figure 4: Log-log scale inverted CDF of instantaneous median job usage, accounting for both varying per-task usage and varying job task counts.

running in the 'left-over' CPU of a machine; it makes little sense to talk about the CPU:memory ratio without adjusting these. Rounding these requests to the next largest request size, the CPU:memory ratio spans about two orders of magnitude.

3.5 Distributions

The job durations and total requested resources of jobs appear to form a heavy-tailed distribution. We found that power law distributions are not good fits to the data (p-value $\ll 0.1$ for the Kolmogorov-Smirnov test for goodness of fit, from the Monte Carlo method recommended in [6]– clearly below the 0.10 threshold it recommends). The same is true for task durations and the total requested resources (CPU-days or memory-days) of tasks and jobs.

The median instantaneous usage of the largest 0.2% or so jobs does appear to follow a power law distribution by this test. The tail (top 20%) of the distribution of the usage (in terms of task-days used) of 'users' (which can represent individuals or services) is also a plausible power law fit. But generally, the overall distributions in this trace are do not appear to match a power law or other simple statistical distributions, like a lognormal, Weibull, or exponential distribution.

For durations, there is a bias from our inability to observe when jobs or tasks are running outside the month of the trace. Particularly, this means that the durations we can observe are limited to the time from when the job or task starts till the end of the trace time period. We corrected for this by minimizing the K-S distance between the power law distribution with fixed cut-off and identically cut-off observations. We selected only jobs or tasks that started near the beginning of the trace to ensure that the fixed cut-off could be large without introducing bias from tasks which start near the end of the trace time window. The p-value for the K-S test was still substantially below the threshold recommended in [6] for plausibility of fit.

Job and task parameters, like the number of tasks in jobs or the amount of CPU or memory requested for each task, are very discrete and

unsmooth, apparently due to human factors (see Section 5.4). This makes it unlikely that any common distribution will accurately model these parameters.

Given the lack of a clear statistical fit, we believe that anyone trying to evaluate a scheduler targeting workloads like this trace's would be best advised to sample directly. We believe, however, that using a power-law-like heavy-tailed distribution for task usage and task durations with similar distributions heavily-biased towards powers-of-two and ten for user-set job parameters like task counts and resource requests may approximate the observed behavior.

4. DYNAMICITY

Schedulers targeting workloads of long-running services may assume that the cluster state changes slowly and, consequently, may assume that considerable time or resources can be expended when making scheduling or placement decisions. The mixed workload of this trace violates those assumptions. The scheduler needs to make decisions about where to place tasks tens of times per second and even frequently needs to restart tasks. Even though there are long-running programs whose placement could be carefully optimized, these are not most of the requests to the scheduler; instead, most of the requests are shorter tasks which the scheduler must process quickly.

In this section, we take a closer look at the highly dynamic nature of the Google cluster analyzed. The dynamicity is both in hardware availability to the scheduler and the behavior of tasks.

4.1 Machine Churn

Researchers evaluating cluster schedulers which have dedicated clusters usually assume machine downtimes are either negligible or occur at a rate similar to hardware failure rates. In this cluster, machines become unavailable to the scheduler more frequently: about 40% of the machines are unavailable to the scheduler at least once. The rate of unavailability corresponds to about 9.7 losses of availability per machine per year. Since hardware failures do not occur so frequently, we suspect that most of the down times represent machine maintenance, but the trace lacks information to confirm this. Usually, these periods of unavailability last less than half an hour, which is more consistent with planned maintenance events than with hardware failures.

There are some indications of space-correlated downtime, including tens of events where tens to around 100 distinct machine losses occur within minutes of each other, but these do not account for a majority of machine downtimes in the trace. At no point in the trace does it appear that less than 98% of the machines are available, and over 95% of the time, more than 99% of the machines are available.

4.2 Task and Job Churn

Since many tasks behave like long-running services, one might expect the scheduler not to have much work from new tasks. This is especially true since the trace providers indicate that MapReduce programs execute separate jobs for the workers and the masters. The cluster jobs serve as execution containers for many map/reduce tasks. So, at least some common sources of what would be fine-grained tasks are not directly managed by this scheduler. However, Figure 5 shows that the scheduler must decide where (or whether) to place runnable tasks frequently. In peak hours, the scheduler needs to make hundreds of task placement decisions per second. Even during quieter times, the average scheduling throughput is several tasks per second.

There are two reasons for the frequency of scheduling events. One is that there are many short-duration tasks being scheduled. Another is that tasks terminate and need to be rescheduled; we will call this *resubmission*. The task terminations preceding these resubmissions are labeled by the trace providers: they are either *failures* (software crashes of a task), *evictions* (task ceased to fit on the machine, due to competing workload, over-commitment of the machine, or hardware failures) or *kills* (underlying reason for a task's death is not available).

Resubmissions account for nearly half of task submissions to the scheduler. However, a principal cause of resubmissions (14M of the 22M resubmission events) is tasks which repeatedly fail and are retried. Another major cause (4.5M events) of resubmissions is evictions; as discussed later, most of these evictions are attributable to machine configuration changes or other (higher priority) workload being started on the machine. For the remaining resubmissions (4.1M events), the task is marked as *killed*; these may logically represent software failures or evictions, or other reasons, such as tasks being restarted to change their configuration.

Crash-loops. Large spikes in the rate of task resubmissions seen in Figure 5 can be attributed to 'crash-loops'. These are cases where the tasks of a job fail deterministically shortly after starting, but they are restarted after these failures. Of the 14M task failures recorded in the trace, 10M are in three crash looping jobs, each of which has tens of thousands of distinct tasks and repeatedly failing tasks. The length of these three large jobs ranges from around 30 minutes to around 5 days.

Jobs with large numbers of tasks are not the only ones that appear to experience crash-loops. Approximately 2% of the memory-time requested comes from jobs that experience more than 10 failures per task over the course of the trace. Most of these failures occur in lower priority jobs (which are probably used for development), but there are some noticeable (but smaller in terms of task failure count) crash loops even at the production priorities.

Small jobs. Even though the scheduler runs large, long-lived parallel jobs, most of the jobs in the trace only request a small amount of a single machine's resources and only run for several minutes. 75% of jobs consist of only one task, half of the jobs run for less than 3 minutes,

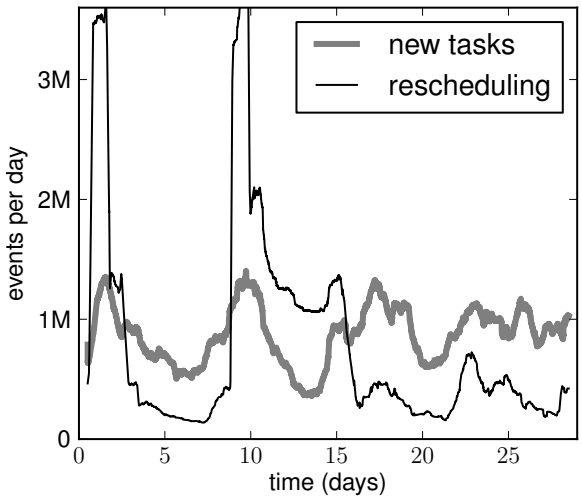


Figure 5: Moving average (over a day-long window) of task submission (a task becomes runnable) rates.

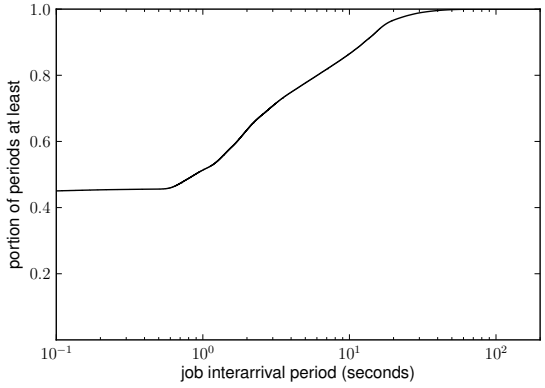


Figure 6: Linear-log plot of inverted CDF of interarrival periods between jobs.

and most jobs request less than 5% of the average machine's resources. These small job submissions are frequent, ensuring that the cluster scheduler has new work to schedule nearly every minute of every day.

Job submissions are clustered together (Figure 6 shows job interarrival times), with around 40% of submissions recorded less than 10 milliseconds after the previous submission even though the median interarrival period is 900 ms. The tail of the distribution of interarrival times is power-law-like, though the maximum job interarrival period is only 11 minutes.

The prevalence of many very small interarrival periods suggest that some sets of jobs are part of the same logical program and intended to run together. For example, the trace providers indicate that MapReduce programs run with a separate 'master' and 'worker' jobs, which will presumably each have a different shape. Another likely cause is embarrassingly parallel programs being split into many distinct single-task jobs. (Users might specify many small jobs rather than one job with many tasks to avoid implying any co-scheduling requirement between the parallel tasks.) A combination of the two of these might explain the very large number of single-task jobs.

Evictions. Evictions are also a common cause of task rescheduling. There are 4.5M evictions recorded in the trace, more than the number of recorded software failures after excluding the largest crash-looping jobs. As would be expected, eviction rates are related to task priorities. The rate of evictions for production priority tasks is comparable to the rate of machine churn: between one per one hundred task days and one per fifteen task days, depending on how many unknown task terminations are due to evictions. Most of these evictions are near in time to a machine configuration record for the machine the task was evicted from, so we suspect most of these evictions are due to machine availability changes.

The rate of evictions at lower priorities varies by orders of magnitude, with some weekly pattern in the eviction rate. Gratis priority tasks average about at least 4 evictions per task-day, though almost none of these evictions occur on what appear to be weekends. Given this

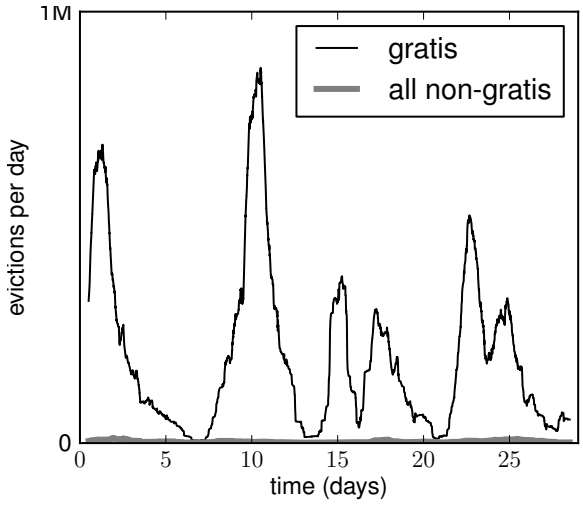


Figure 7: Moving average (over day-long window) of task eviction rates, broken by priority.

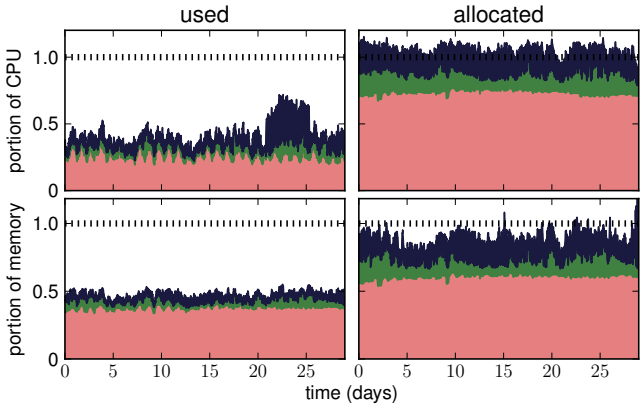


Figure 8: Moving hourly average of CPU (top) and memory (bottom) utilization (left) and resource requests (right). Stacked plot by priority range, highest priorities (production) on bottom (in red/lightest color), followed by the middle priorities (green), and gratis (blue/darkest color). The dashed line near the top of each plot shows the total capacity of the cluster.

eviction rate, an average 100-task job running at a gratis priority would expect about one task to be lost every 15 minutes. These programs must tolerate a very high "failure" rate by the standards of a typical cloud computing provider or Hadoop cluster.

Almost all of these evictions occur within half a second of another task of the same or higher priority starting on the same machine. This indicates that most of these evictions are probably intended to free resources for those tasks. Since the evictions occur so soon after the higher priority task is scheduled, it is unlikely that many of these evictions are driven by resource usage monitoring. If the scheduler were measuring resource usage to determine when lower-priority tasks should be evicted, then one would expect many higher-priority tasks to have a 'warm-up' period of at least some seconds. During this period, the resources used by the lower-priority task would not yet be required, so the task would not be immediately evicted.

Given that the scheduler is evicting before the resources actually come into conflict, some evicted tasks could probably run substantially longer — potentially till completion, without any resource conflicts. To estimate how often this might be occurring, we examined maximum machine usage after evictions events and compared these to the requested resources of evicted tasks. After around 30% of evictions, resources requested by the evicted tasks appear to remain free for an hour after the eviction, suggesting that these evictions were either unnecessary or were to make way for brief usage spikes we cannot detect in this monitoring data.

5. RESOURCE USAGE PREDICTABILITY

The trace includes two types of information about the resource usage of jobs running on the cluster: the resource *requests* that accompany

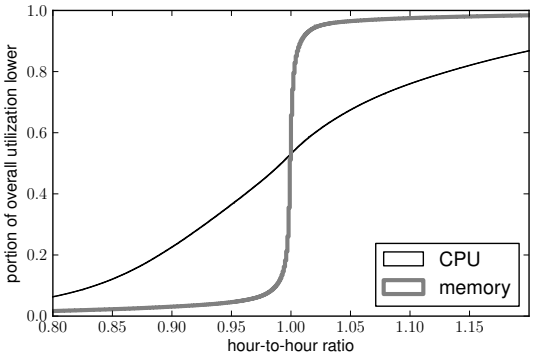


Figure 9: CDF of changes in average task utilization between two consecutive hours, weighted by task duration. Tasks which do not run in consecutive hours are excluded.

each task and the actual resource *usage* of running tasks. If the scheduler could predict actual task resource usage more accurately than that suggested by the requests, tasks could be packed more tightly without degrading performance. Thus, we are interested in the actual usage by jobs on the cluster. We find that, even though there is a lot of task churn, overall resource usage is stable. This stability provides better predictions of resource usage than the resource requests.

5.1 Usage overview

Figure 8 shows the utilization on the cluster over the 29 day trace period. We evaluated utilization both in terms of the measured resource consumption (left side of figure) and 'allocations' (requested resources of running tasks; right side of figure). Based on allocations, the cluster is very heavily booked. Total resource allocation at almost any time account for more than 80% of the cluster's memory capacity and **more** than 100% of the cluster's CPU capacity. Overall usage is much lower: averaging over one-hour windows, memory usage does not exceed about 50% of the capacity of the cluster and CPU usage does not exceed about 60%.

The trace providers include usage information for tasks in five-minute segments. At each five-minute boundary, when data is not missing, there is at least (and usually exactly) one usage record for each task which is running during that time period. Each record is marked with a start and end time. This usage record includes a number of types of utilization measurements gathered from Linux containers. Since they are obtained from the Linux kernel, memory usage measurements include some of the memory usage the kernel makes on behalf of the task (such as page cache); tasks are expected to request enough memory to include such kernel-managed memory they require. We will usually use utilization measurements that represent the average CPU and memory utilization over the measurement period.

To compute the actual utilization, we divided the trace into the five-minute sampling periods; within period, for each task usage record available, we took the sum of the average CPU and memory usage weighted by the length of the measurement. We did not attempt to compensate for missing usage records (which the trace producers estimate accounts for no more than 1% of the records). The trace providers state that missing records may result from "the monitoring system or cluster [getting] overloaded" and from filtering out records "mislabeled due to a bug in the monitoring system" [18].

5.2 Usage stability

When tasks run for several hours, their resource usage is generally stable, as can be seen in Figure 9. Task memory usage changes very little once most tasks are running. Memory usage data is based on physical memory usage, so this stability is not simply a consequence of measuring the available address space and not actual memory pressure.

Because there are many small tasks, a large relative change in CPU or memory usage of an individual task may not translate to large changes in the overall ability to fit new tasks on a machine. A simple strategy for determining if tasks fit on a machine is to examine what resources are currently free on each machine and predict that those resources will remain free. We examined how well this would perform by examining how much machine usage changes. On timescales of minutes, this usually predicts machine utilization well as can be seen in Figure 10. Since most tasks only run for minutes, this amount of prediction is all that is likely required to place most tasks effectively. Longer running tasks may require more planning, but their resource usage tends to mimic other longer-running tasks, so the scheduler can even plan well for these long-running tasks by monitoring running task usage.

5.3 Short jobs

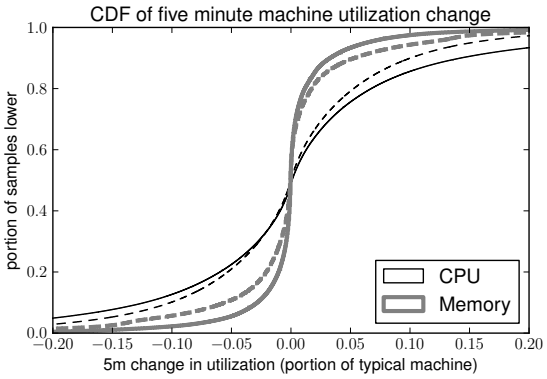


Figure 10: CDF of changes in average machine utilization between two consecutive five minute sampling periods. Solid lines exclude tasks which start or stop during one of the five minute sampling periods.

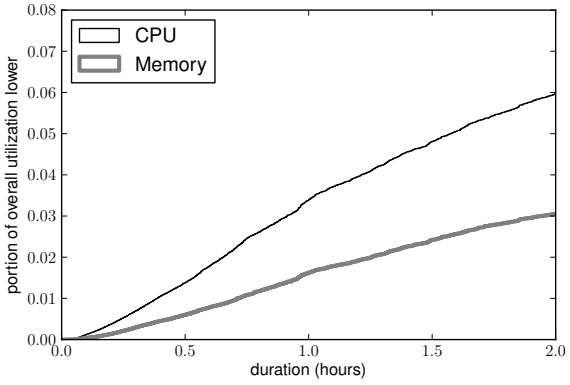


Figure 11: CDF of utilization by task duration. Note that tasks running for less than two hours account for less than 10% of utilization by any measure.

One apparent obstacle to forecasting resource availability from previous resource usage is the frequency with which tasks start and stop. Fortunately, though there are a large number of tasks starting and stopping, these short tasks do not contribute significantly to usage. This is why, as seen in Figure 10, ignoring the many tasks which start or stop within five minutes does not have a very large effect. Figure 11 indicates that jobs shorter than two hours account for less than 10% of the overall utilization (even though they represent more than 95% of the jobs). Hence, the scheduler may safely ignore short-running jobs when forecasting cluster utilization.

Even though most jobs are very short, it is not rare for users to run long jobs. 615 of the 925 users of the cluster submit at least one job which runs for more than a day, and 310 do so outside the gratis priorities.

5.4 Resource requests

Even though we've seen that resource requests in this trace are not accurate in aggregate, perhaps this behavior only arises because users lack tools to determine accurate requests or because users lack incentives to make accurate requests. Thus, we are interested in how closely requests could be made to reflect the monitored usage.

Non-automation. Resource requests appear to be specified manually, which may explain why they do not correspond to actual usage. One sign of manual request specification is the uneven distribution of resource request sizes, shown in Figure 12. When users specify parameters, they tend to choose round numbers like 16, 100, 500, 1000, and 1024. This pattern can clearly be seen in the number of tasks selected for jobs in this trace; it is not plausible that the multiples of powers of ten are the result of a technical choice. We cannot directly identify any similar round numbers in the CPU and memory requests because the raw values have been rescaled, but the distribution shows similar periodic peaks, which might represent, e.g., multiples of 100 megabytes of memory or some fraction of a core. For memory requests, it is unlikely that these round numbers accurately reflect requirements. For CPU requests, whole numbers of CPUs would accurately reflect

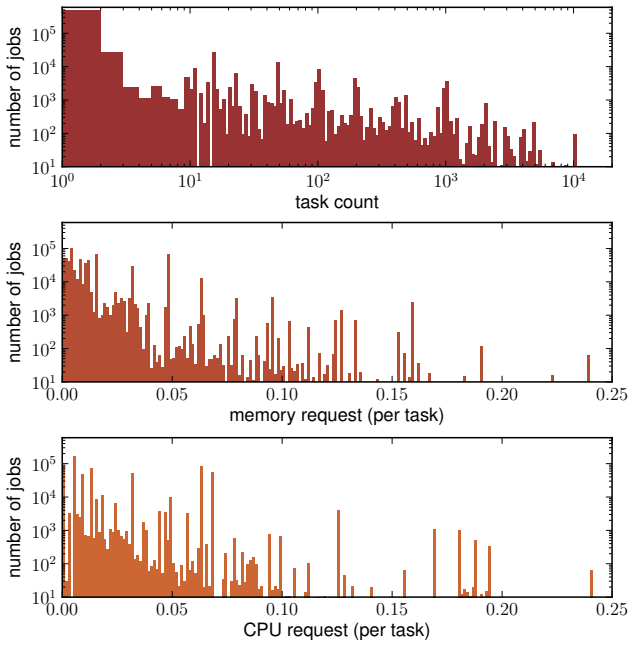


Figure 12: Histograms of job counts by task count (top), memory request size (middle) and CPU request size (bottom). Note the log-scale on each y-axis and the log-scale on the top plot's x-axis. We speculate that many of the periodic peaks in task counts and resource request sizes represent humans choosing round numbers. Memory and CPU units are the same as Table 1. Due to the choice of x-axis limits, not all jobs appear on these plots.

the CPU a disproportionate number of tasks would use [19], but it seems unlikely that the smallest 'bands' (at around 1/80th of a machine) represent a whole number of cores on a 2011 commodity machine.

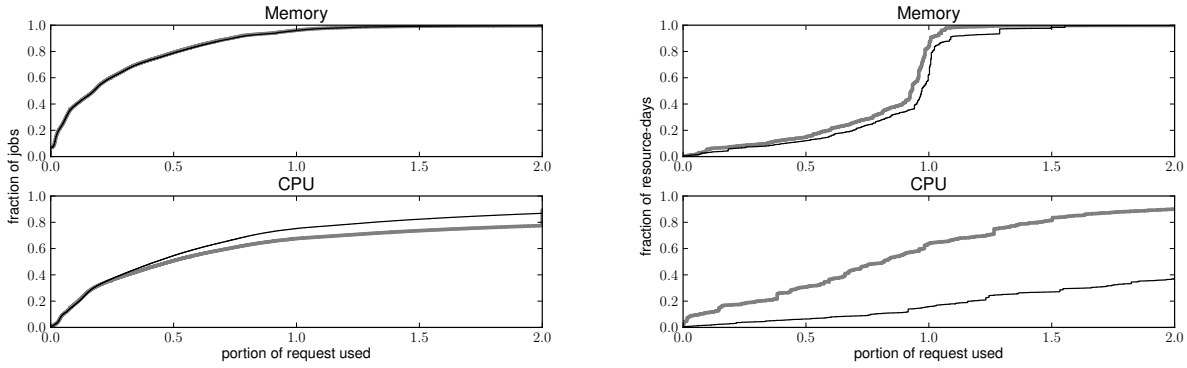
Request accuracy. Requests in this trace are supposed to indicate the "maximum amount . . . a task is permitted to use" [18]. Large gaps between aggregate usage and aggregate allocation, therefore, do not necessarily indicate that the requests are inaccurate. If a task ever required those CPU and memory resources for even a second of its execution, then the request would be accurate, regardless of its average consumption. Thus, resource requests could be thought of as reflecting the maximum *anticipated* utilization of CPU and memory for the requesting task.

Without the ability to rerun programs and understand their performance requirements, we cannot find the "correct" resource request for applications, i.e., a request that reflects the maximum resources that the tasks may need. We will make some guesses from the actual usage we observe: we believe the maximum or some high (e.g. 99th) percentile of the actual usage samples is a good estimate of an appropriate resource request. Since the tasks in a job are usually identical, we will assume that the resource request needs to be suitable for each task within a job. To avoid being sensitive to any outlier tasks, we will take the 99th percentile of the estimates for each task as the estimate for resource request for all tasks of the jobs.

The differences between the resource request we would estimate from usage and the actual request is not what one would infer from the aggregate usage and requests shown in Figure 8. One would assume that jobs generally requested twice as much as they used from the aggregate figures. But considering the high percentile usage as an estimate of the actual resource request, the typical request 'accuracy' is very different.

As shown in Figure 13(b), jobs accounting for about 60% of the memory allocated fall within 10% of our estimate of their appropriate request. The remaining jobs over-estimate their memory usage. Memory requests rarely under-estimate their jobs' memory utilization by a large factor, probably because tasks are terminated if their memory request is exceeded by too much. The memory overestimates amount to about a fifth of the total memory allocation while the total difference between the memory usage and allocation is about 50% of the memory allocation. The remaining 30% can be accounted for by the difference between high-percentile memory usage and average memory usage within each job, discussed in the next section.

CPU usage is not as constrained by the request as memory usage. Tasks both use much less or much more than CPU than they request. To determine whether the CPU requests are usually too high or too low, it is useful to weight each job by the size of its per-task resource request multiplied by the number of task-days for which its tasks run. The difference between the request with this weighting is shown in Figure



(a) unweighted (b) weighted

Figure 13: 13(a) shows the CDF of the maximum five-minute usage sample relative to the resource request for the corresponding job. The thin line represents the largest five-minute sample within each job. The thick line discards outlier tasks within a job; for each job, it shows the 99th percentile of the largest five-minute usage measurements for each task. The top graphs shows memory usage; the bottom graphs show CPU usage. 13(b) shows the same CDFs as 13(a), but with each job weighted by its per-task request times the number of task-days its tasks run.

13(b); the resulting CDF reflects how much each unit of allocation is an over- or under-estimate of its usage. As shown in Figure 13(b), with this weighting weighted, the 'middle' job accurately estimates its CPU usage: about half the weight is in jobs for which 99% of the tasks have no CPU usage sample greater than the CPU request, and half in jobs with samples strictly greater.

Outliers within tasks. Resource usage is not the same between tasks in a job or over time within a job. As long as requests or predictions need to represent maximum usage, this variation will hamper the efficiency of any scheduler no matter how accurate the requests or predictions are. In this trace, this variation seems to be responsible for more of the inability of resource requests to predict usage than the requests being poorly set by users.

Conceivably, if users understood their resource usage well enough and the scheduler interface were rich enough, one might imagine that users would identify these outliers explicitly. The system from which the trace was extracted allows users to change their resource requests over time and across tasks in a job, but this feature is rarely used: jobs accounting for around 4% of memory allocated adjusted their CPU request in this fashion, and jobs accounting for another 3% updated their memory request.

Based on the usage measurements in the trace, the maximum of the usage samples are much larger than average usage — both between tasks in a job and within single long-running tasks over time. Even though memory usage is very stable overtime, these outliers exists even for memory utilization: differences between the maximum usage of tasks within jobs account for about 20% of the total memory-hours requested (roughly the same as the apparent inaccuracies of each job's requests). Differences between the maximum usage for a task and its typical usage account for another 18%. (Because maximum memory usage can slightly exceed the memory allocation, the sum of these percentages can exceed the aggregate difference between requested and used memory-hours.)

These large gaps are rare; for example, most of the difference is the difference between the maximum and 99th percentile measurements (even for memory usage). We suspect that some of these outliers may be measurement error (e.g., page cache memory for common system files being accounted to only one program) or spurious (extra data being cached by the kernel that would be released at no performance cost under memory pressure). However, similar gaps exist for CPU usage where variability is less surprising. The effect does not appear to arise solely from startup effects, as even very long-running jobs experience these outliers.

Even though these gaps are rare, they have substantial implications for schedulers. For many applications, if any one task in a job can experience a usage spike then schedulers or users might (perhaps correctly) assume that all tasks will eventually. Schedulers that actually set aside resources for this spike will necessarily limit their system utilization they can achieve. Instead, to achieve high utilization in the face of usage spikes, schedulers should not set aside resources, but have resources that can, in the rare cases where necessary, be made available by, for example, stopping or migrating fine-grained or low-priority workload.

5.5 Repeated Submissions

The frequency of job submissions suggests that some programs are run repeatedly. Repeated jobs provide a natural opportunity for resource prediction. If a workload is composed of primarily of repeated jobs — such as periodic batch jobs or programs being tweaked while being developed — than over time the scheduler can allocate the job exactly the resources it requires by using statistics collected during previous runs of the job.

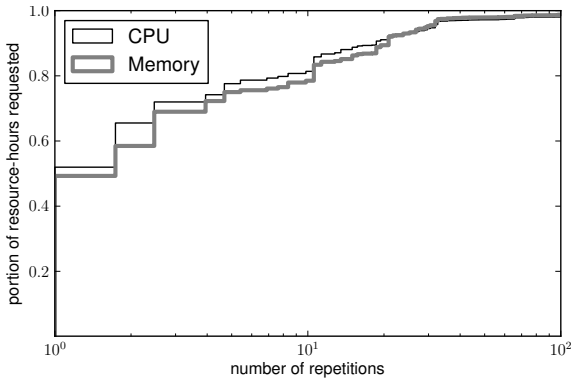


Figure 14: CDF of the portion of CPU-hour and memory-hours requested by jobs by the number of apparent repetitions.

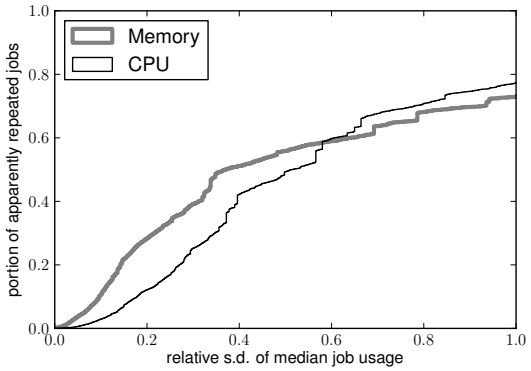


Figure 15: CDF of relative standard deviation of median repeated job utilization.

We can approximately identify repeated jobs in this trace through the "logical job name", which, according the trace providers, will usually remain the same across different executions of the same program. The trace providers do not guarantee, however, that different programs cannot have the same logical job name. Based on this information, we find that many jobs may represent repeated programs.

There are around 40k unique names in the trace. The number of jobs per name loosely follows a power-law-like distribution except that some numbers of job repetitions are much more popular than predicted. These peaks correspond to quantities such as the number of days in the trace, so they may indicate some periodic jobs.

Frequently repeated jobs do not, however, account for very much of the utilization of the cluster. As shown in Figure 14, jobs names which repeat more than 5 times account for only about 30% of the utilization, so the effect of any repeated-job-based prediction is limited. More concerningly, the usage of repeated jobs is often not very consistent; as shown in Figure 15. Without more information, predictions based on this notion of repeated jobs are only likely to be accurate within 25% for jobs accounting for less than half of usage (in memory-hours) of all the repeated jobs.

6. TASK CONSTRAINTS

In general, a constraint is any specification from the user that restricts the placement of a task, such as based on machine capabilities or the relative placement of related tasks. These constraints can be challenging for schedulers since they can leave the scheduler with tasks for which sufficient machine resources are free but unusable until less-constrained tasks are moved, terminated, or finish. Constraints can be *hard* or *soft*. Hard constraints divide the space of possible resource assignments into feasible and non-feasible. Soft constraints specify a preference gradient over members of the solution space [22]. The Google trace provides hard constraints for approximately 6% of all tasks submitted to the scheduler. These constraints capture restrictions based on machine attributes and prevent coscheduling of a job's tasks on a single machine (known as the anti-affinity constraint). While some soft constraints and other types of hard constraints are supported by the scheduler [18], only these two major categories of hard constraints were captured by the trace: resource attribute based and anti-affinity

restrictions.

6.1 General characterization

In the first category, task constraints are specified as (key, value, operator) tuples over machine attributes. There are 17 unique attribute keys used by task constraints. Of the 6% that specify constraints, most tasks do so over a single attribute; Table 2 shows how many unique attributes each of the 6% uses in its constraint specification.

task count	unique constraint count
24019159	0
1368376	1
33026	2
2782	3
1352	4
30	5
6	6
25424731	17

Table 2: Task count by the number of unique attribute keys specified in constraints.[1]

Operators used in the trace are $=$, $\neq$, $<$, and $>$. A machine can be used for a task only if the machine attribute value corresponding to the key satisfies the binary relation specified by the operator and the value provided in the constraint tuple. If the machine specifies no value for an attribute, then 0 or the empty string is assumed. This permits testing for the presence or absence of a particular attribute. Note that multiple constraints against the same key are permitted on a task. It is possible, for example, for a task's constraints to require that a machine attribute fall within a certain range.

According to [21], example attribute keys might be architecture, number of cores, number of spindles, number of CPUs, kernel version, CPU clock speed, Ethernet speed, and platform family. The trace has all attribute keys anonymized, and numeric values obfuscated such that only the relative ordering is preserved. In some cases, however, we were able to make reasonable guesses as to the nature of a particular constraint key.

All attribute keys used by constraints can be uniquely identified by their 2-letter prefix. We will thus refer to them as such for brevity. Table 3 shows the three most popular constraints. Two of them have an order of magnitude more tasks specifying them than any other constraint, and, counted by unique users, tasks, or jobs, they always make the top 3. The most popular constraint, 'o/', is at the top in all three categories. Interestingly, this constraint is only ever used with a single operator—equality—and only one attribute value—the empty string. Thus, it specifies all machines that do not have this attribute key. There are only 142 machines that have this attribute and are avoided by the specification of this constraint.

One aspect of constraint specification that we found surprising is the relatively small number of *unique* (key, value, operator) tuples specified, especially considering that values can be numeric and operators include $<$ and $>$. Excluding attribute key 'GK' (getting separate treatment in 6.3), there are only 44 unique tuples. The only constraint that has thousands of possible values specified in this trace is the 'GK' constraint, which behaves like a unique machine identifier. All other constraints have less than 10 unique values they compare against.

The second class of hard constraints captured by this trace is a type of *anti-affinity constraint*. Each task can specify that it cannot run on the same machine as any other task belonging to the same job. This constraint is specified as a boolean flag on each task, separate from the other constraints, and will be discussed in greater detail in Section 6.2.

6.2 Constraint-induced scheduling delay

[1]The uniqueness in this case was determined by attribute key alone. This ensures that "range" constraints (e.g., $X > 2$ and $X < 5$) and attribute value updates are not double-counted.

constraint key	task count	job count	user count
o/	958203	51870	173
5d	307918	2310	68
ma	20144	1415	73

Table 3: The three most popular constraint keys by unique task, job and user count.

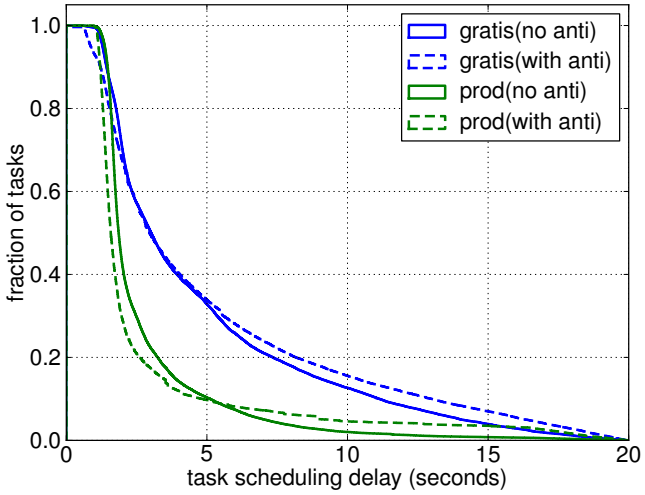


Figure 16: Task scheduling delay CDF broken down by both priority and anti-affinity requirement. The dashed line and "(with anti)" indicates those tasks with anti-affinity constraints. Scheduling delay is higher for tasks with anti-affinity constraints for two most important classes in the trace.

	no anti-affinity		with anti-affinity	
Priority	task count	delay (s)	task count	delay (s)
0-1	28002936	233.49	260809	892.19
2-8	14148983	244.19	2945288	11.40
9-10	1558255	5.06	80861	13.19

Table 4: Anti-affinity and priority as predictors of scheduling delay.

We found the number of constraints poorly correlated with scheduling latency. For example, mean scheduling delay across all tasks with one constraint specified is similar to that of tasks with six constraints (546 and 507 seconds, respectively), while tasks with five constraints see a 16 second mean scheduling delay. The reason for that is that constraints vary widely in how "constraining" they are. As an example, we've found at least 4 string-valued constraints that match the whole cluster. For example, all constraints on attribute key "rs", with the exception of attribute value "bf", match the whole cluster. The mere presence or absence of constraints, however, is a better predictor of delay. Unconstrained tasks see a mean scheduling delay of 211 seconds, while constrained ones spend 46% more time (308 seconds) in the queue, on average.

As shown in Table 4 and Figure 16, the absence of the special anti-affinity constraint correlates with a decrease in scheduling latency. The difference is a factor of 2 for *production* priority workload and a factor of 4 for *gratis*. The remaining priority groups in aggregate do not follow the same trend, though. In Figure 16, we've broken down the scheduling latency by priority group (color) as well as anti-affinity (linestyle). It can be seen that, for a given task scheduling delay greater than 2 seconds (for *gratis* workloads) and 5 seconds for *production*, more anti-affinity constrained tasks experience that delay than those without anti-affinity.

Thus, the presence or absence of either the anti-affinity constraint or constraints specified over the set of machine attributes correlates with higher or lower scheduling delay respectively. The difference is not as dramatic as reported in [21] for attribute constraints, but the anti-affinity requirement did have a factor of 2 to 4 effect.

6.3 Locality

One machine attribute was found to have the characteristics of a unique machine identifier. We will refer to this key as 'GK' (consistent with Section 6.1). First, there are 12569 unique values for this attribute, which is only 14 less than the number of machines. Second, in the trace, a machine's value for GK never changes. Each machine with the GK attribute has exactly one value throughout the entire trace. Conversely, each unique value is ever associated with exactly one machine. The 14 machines unaccounted for simply do not have this attribute specified, i.e. there's a one-to-one, but not onto, mapping from the set of GK values to the set of machines.

The number of unique values that the GK attribute was used with as a constraint is three orders of magnitude greater than the number of unique values for all other unique (attribute, operator) pairs combined. We counted 5977 unique values used with the GK constraint and

equality operator, but all other constraints had fewer than 10 unique values.

These three observations lead us to conclude that the GK attribute is used as a machine location identifier. We will refer to it as a locality constraint. It was used by 17% of 253 users that specified constraints, by 1.8% of constrained tasks, and 0.7% of constrained jobs. Although not wide-spread, it is not specific to one or a few jobs, users, or tasks; tens of users, hundreds of jobs, and tens of thousand of unique tasks used the GK locality constraint. In addition to equality tests, 5 unique users used the GK constraint with a $\neq$ test, explicitly avoiding 4 unique machines.

Even though the reasons for specifying locality are not available in this trace, the number of unique tasks (24854) that use it suggests that it was performance-related. Hypothetical examples might include cache, memory, loaded binary, or disk locality. This locality constraint, as all the others, was specified as a hard constraint. If our hypothesis is true, and this is indeed a performance-related locality constraint, its specification as a hard constraint could be sub-optimal. Specifically, it poses an interesting research question of how to trade off constraint-induced scheduling latency for runtime performance improvements [22]. The consequences of specifying locality *preferences* as hard constraints versus a placement preference is an interesting question for further exploration.

7. CONCLUSIONS

Analysis of Google trace data exposes a challenging set of consequences of consolidation, for which new resource management approaches will likely be necessary. The most notable characteristic, seen in almost every aspect, is heterogeneity: the resources (e.g., cores:RAM per machine) and, especially, the tasks executed (e.g., duration and resources needed) vary widely, making popular simplifications (e.g., slots) unsuitable. Some tasks also come with constraints on which machine types are suitable, increasing the impact of the heterogeneity and creating concerns with greedy schedulers that are unable to efficiently migrate executing tasks. The workload is also quite dynamic, varying over time on most dimensions, driven in part by many short jobs for which quick scheduling will be important. While few simplifying assumptions appear to hold, we do find that many longer-running jobs have relatively stable resource utilizations, which can help adaptive resource schedulers. Overall, we believe that the observations arising from our analysis confirm the need for new cloud resource schedulers and also provide useful guidance for their design.

8. ACKNOWLEDGMENTS

We thank Google for releasing this trace and John Wilkes for his valuable feedback on earlier versions of this paper.

This research is supported in part by Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by gifts from Google, SAP, Amazon Web Services, APC, Blue Goji, Cloudera, EMC, Emulex, Ericsson, Facebook, Fusion-IO, General Electric, Hewlett Packard, Hitachi, Huawei, IBM, Intel, MarkLogic, Microsoft, NEC Labs, NetApp, Oracle, Panasas, Quanta, Riverbed, Samsung, Seagate, Splunk, STEC, Symantec, VMware, by an NSERC Postgraduate Doctoral Fellowship, and by DARPA (contract number FA8650-11-C-7136). We thank the member companies of the PDL Consortium and the AMPLab sponsors for their interest, insights, feedback, and support.

9. REFERENCES

[1] ISTC-CC: Research Overview, 2012. `http://www.istc-cc.cmu.edu/research`.

[2] B. Abrahao and A. Zhang. Characterizing application workloads on CPU utilization for utility computing. Technical Report HPL-2004-157, HP Labs, 2004.

[3] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proc. of the 1st ACM Symposium on Cloud Computing*, 2010.

[4] Y. Chen, S. Alspaugh, and R. H. Katz. Design insights for mapreduce from diverse production workloads. Technical Report UCB/EECS-2012-17, EECS Dept, University of California, Berkeley, Jan 2012.

[5] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating MapReduce performance using workload suites. In *Proc. of 19th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 390–399, July 2011.

[6] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, Nov. 2009.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] S. Di, D. Kondo, and W. Cirne. Characterization and Comparison of Google Cloud Load versus Grids. `http://hal.archives-ouvertes.fr/hal-00705858`, 2012.

[9] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *SIGMETRICS Perform. Eval. Rev.*, 26:14–29, March 1999.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proc. of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.

[11] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proc. of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010.

[12] M. Kozuch, M. Ryan, R. Gass, S. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. Ganger. Tashi: location-aware cluster management. In *Proc. of the 1st Workshop on Automated Control for Datacenters and Clouds*, 2009.

[13] Z. Liu and S. Cho. Characterizing machines and workloads on a Google cluster. In *Eighth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems*, September 2012.

[14] U. Lublin and D. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[15] A. Oliner and J. Stearley. What supercomputers say: a study of five system logs. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*, pages 575–584, June 2007.

[16] N. Poggi, D. Carrera, R. Gavalda, J. Torres, and E. Ayguadé. Characterization of workload and resource consumption for an online travel and booking site. In *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)*, 2010.

[17] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical Report ISTC-CC-TR-12-101, Intel Science and Technology Center for Cloud Computing, Apr 2012.

[18] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema, 2011. `http://goo.gl/5uJri`.

[19] C. Reiss, J. Wilkes, and J. L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *Proc. of IEEE/IFIP International Workshop on Cloud Management (CloudMan'12)*, April 2012.

[20] M. Schwarzkopf, D. G. Murray, and S. Hand. The seven deadly sins of cloud computing research. In *Proc. of the 4th USENIX Workshop on Hot Topics in Cloud Computing*, 2012.

[21] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *Proc. of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, 2011.

[22] A. Tumanov, J. Cipar, M. A. Kozuch, and G. R. Ganger. alsched: algebraic scheduling of mixed workloads in heterogeneous clouds. In *Proc. of the 3nd ACM Symposium on Cloud Computing*, SOCC '12, 2012.

[23] J. Wilkes. Omega: Cluster management at google. Video from 2011 Google Faculty Summit; `https://youtu.be/0ZFMlO98Jkc`, July 2011.

[24] J. Wilkes and C. Reiss. Details of the ClusterData-2011-1 trace, 2011. `https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1`.

[25] M. Zaharia, B. Hindman, A. Konwinski, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. The datacenter needs an operating system. In *HotCloud*, June 2011.

[26] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing enviornments. In *Proceedings of the 9th ACM International Conference on Autonomic Computing*, 2012.

[27] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner. Co-analysis of RAS log and job log on Blue Gene/P. In *Proc. of IEEE International Parallel Distributed Processing Symposium (IPDPS)*, May 2011.