Towards understanding heterogeneous clouds at scale: Google trace analysis

Charles Reiss (UC Berkeley), Alexey Tumanov (CMU),
Gregory R. Ganger (CMU), Randy H. Katz (UC Berkeley),
Michael A. Kozuch (Intel Labs)

ISTC-CC-TR-12-101
April 2012

INTEL SCIENCE & TECHNOLOGY CENTER FOR CLOUD COMPUTING
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Abstract

With the emergence of large, heterogeneous, shared computing clusters, their efficient use by mixed distributed workloads and tenants remains an important challenge. Unfortunately, little data has been available about such workloads and clusters. This paper analyzes a recent Google release of scheduler request and utilization data across a large (12500+) general-purpose compute cluster over 29 days. We characterize cluster resource requests, their distribution, and the actual resource utilization. Unlike previous scheduler traces we are aware of, this one includes diverse workloads – from large web services to large CPU-intensive batch programs – and permits comparison of actual resource utilization with the user-supplied resource estimates available to the cluster resource scheduler. We observe some under-utilization despite over-commitment of resources, difficulty of scheduling high-priority tasks that specify constraints, and lack of dynamic adjustments to user allocation requests despite the apparent availability of this feature in the scheduler.

Acknowledgements: The authors wish to gratefully acknowledge Google for their public release of this cluster trace dataset. 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 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.

Keywords: cloud computing, cluster scheduling, trace characterization

1 Introduction

Consolidation of heterogeneous workloads into a common shared infrastructure is becoming increasingly common. It has the advantages of statistical multiplexing of physical resources, centralized asset management, and workload-specific benefits, such as the ability to share datasets and results of intermediate computations. Cluster resource consolidation has been popular with the scientific and supercomputing communities for decades. Only recently, however, has the shared infrastructure started being utilized for a variety of workload profiles, ranging from long-running services, to shorter map-reduce tasks, and even transient opportunistic background computation. Infrastructure operators, such as Google, Microsoft, and Amazon, have now fully embraced this approach.

Little is known, however, about the specifics of how these extensive heterogeneous, physical infrastructures are managed and their resources allocated. Publicly available datasets with traces from such clusters are rare. Google released one such dataset [20] in 2011, augmenting a prior release described in [7]. We are not aware of any other publicly available datasets that describe a larger cluster of such type. The level of detail and mixture of workload types in this trace is unprecedented.

This paper aims to characterize the composition of this example workload. We identify the "boulders" and "sand" of this workload. Like many large workloads, this trace is constructed overwhelmingly of small requests (sand), but its usage is dominated by a modest number of large requests (boulders). These large requests appear to represent long-running services, while the small requests are tiny on many dimensions, including resource requirements, duration, and number of replicas. All these requests do not appear to be particularly well-tuned: many parameters seem to be chosen arbitrarily by humans and do not reflect the actual performance of the jobs.

Comparing overall usage of the cluster to its capacity, we find that the average utilization of the cluster only accounts for half the cluster capacity[1], despite cluster resource over-commitment and demonstrably regular use of preemption to achieve its utilization. Interestingly, restarting tasks such as after preemption events is responsible for a comparable amount of the scheduler's activity to task scheduling. Additionally, we examine and quantify the difficulty of scheduling tasks. Since the scheduler does not gang-schedule jobs, we hypothesize that difficulties in fitting tasks into a running cluster are due to resource constraints.

The remainder of the paper is organized as follows. Section 2 describes the trace, including the cluster, workload mix, and key features and terminology. Section 3 examines basic characteristics, including overall utilization and request frequencies, durations, sizes, and shapes. Section 4 examines in more detail the accuracy of resource allocation requests in order to better understand the large gap between actual aggregate cluster usage and allocations. Section 5 examines task scheduling delay in more detail in order to understand which request characteristics are most challenging to accommodate. Section 6 discusses related work. Section 7 summarizes our contributions.

2 The Trace

The cluster trace released by Google consists of data collected from approximately 12,500 machines and is summarized in Table 1. The cluster machines are not homogeneous; they consist of three different platforms (microarchitecture and memory technology combinations) and a variety of memory/compute ratios. The configurations are shown in Table 2. Because the data has been obfuscated to hide exact machine configurations, 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

[1]Note that the 40-60% utilization seen in the Google trace is much better than is observed in typical data centers. For example, surveys of federal, corporate, and university data centers generally report average utilization of 7–25% [8, 11, 14]. Here, we focus on understanding the remaining room for improvement.

Trace Characteristic	Value
Time span of trace	29 days
Jobs run	650k
Number of users (with usage)	925
Tasks submitted	25M
Scheduler events	143M
Resource usage records	1233M
Compressed size	39 GB

Table 1: Summary characteristics of the trace.

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 2: 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.

throughout this paper. Most of the machines have half of the memory and half the CPU of the largest machines.

Data was collected from two sources: from the cluster scheduler, for which user requests and scheduling actions are available, and from the individual machines, for which measurements of running programs are available.

The cluster scheduler manages a *cell*, which is a collection of physical machines that are operated as a single management unit. The cluster scheduler places *tasks* on these machines. Each task is a request to schedule a unit of execution encapsulated in a Linux *container* – "a lightweight virtual system mechanism sometimes described as 'chroot on steroids' " [3]. A single task can be placed on machines multiple times, but it is only run on exactly one machine at any given time (except in cases of network partitions). For example, a task can be scheduled on a machine, run on that machine for some time, then be terminated and started again on a different machine. Tasks migrated in this fashion do not rely on any scheduler state for migration and/or restart.

Tasks are organized into *jobs*. A job is a collection of one or more usually identical tasks. Typically, all tasks in a job are meant to be run simultaneously, although the scheduler does not ensure this. Although it is possible for a job to contain a heterogeneous mix of tasks, this functionality is rarely used. Instead, when a user wishes to run a mix of different types of processes to act as a unit, she would typically launch it as multiple jobs. For example, a MapReduce program might be launched as two jobs: one for the MapReduce masters and one for the workers. The trace identifies a *user* (a developer or internal service identified for

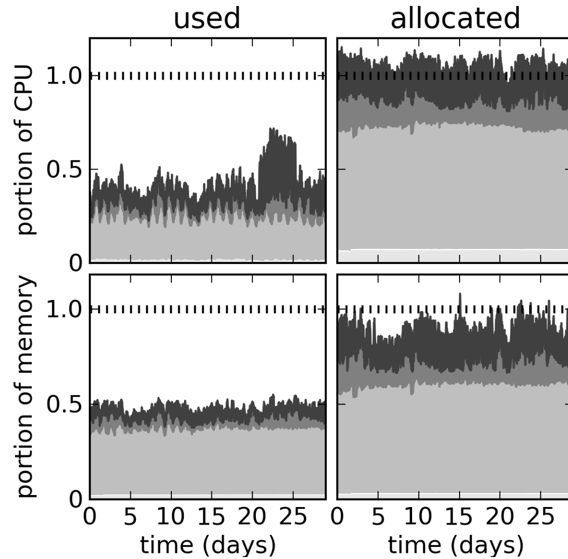

Figure 1: Moving hourly average of CPU (top) and memory (bottom) utilization (left) and resource requests (right). Stacked plot by priority range, highest priority (infrastructure) on bottom, followed by monitoring (barely visible white line), normal production (light grey), other (grey), and gratis (dark grey). The dashed line near the top of each plot shows the total capacity of the cluster.

accounting purposes) associated with each job. The traces do not identify the nature of the computation represented by the jobs/tasks.

Each task description also has scheduling attributes associated with it. Most notable are its *priority*, *resource request*, and *constraints*. *Priority* refers to the entitlement to evict other tasks for resource reclamation. *Resource request* indicates machine resources the task asks for, and task *constraints* indirectly specify which machines are suitable for execution. We define the resource requests successfully granted as *allocations*, while referring to all requests submitted (but not necessarily granted) as *resource requests*.

In addition to these scheduling attributes, the trace contains usage data for most tasks which run. These include measurements of CPU time, memory space usage, and some other measurements (such as local disk, CPI, etc.). These data have much coarser grain than the scheduler data: most measurements are available in only 5 minute increments, while the scheduler data has sub-second precision.

3 Workload characterization

This section provides a broad characterization of the workload by describing (1) the aggregate volume of work expressed as cluster utilization (Section 3.1) and (2) the properties of the jobs and tasks that make up the workload (Section 3.2).

3.1 Cluster utilization

Figure 1 shows the utilization on the cluster over the 29 day trace period. We evaluated utilization both in terms of the allocations (right side of figure) and the measured resource consumption (left 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%.

To compute the utilization, we divided the trace into five-minute time periods; within each time period, for each task usage record available, we took the sum of the resource requests and 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" [16].

3.1.1 Scheduler operation

Unlike many other cluster schedulers, the CPU and memory requests have a loose relationship to the actual allocation on the cluster. Some tasks are permitted to use free CPU cycles beyond their allocation and to use some memory in excess of the requested amount . Memory usage measurements account for some kernel-managed memory like "hot" page cache, so it is possible for tasks to use memory in excess of their request that the kernel can reclaim in the event of memory pressure. Otherwise, tasks that use more resources than they request risk being throttled (for CPU) or killed (for memory).

The scheduler can place tasks on a machine whose total requests exceed the available resources of the machine. This over-commitment is how the resource allocations can total more than 100% of the cluster's CPU and memory capacity. The aggregate amount by which the allocations exceed 100% of the capacity does not reflect the distribution of over-commitment across machines: tasks need to be packed onto machines, and many machines may not be fully allocated even when the total allocation exceeds capacity.

3.1.2 By priority

We have broken down the utilization by the priority in Figure 1. We group the twelve given task priorities, numbered from 0 to 11, into five sets, for which we have chosen the following names based on information given by the trace providers:

- infrastructure (11) — this is the highest (most entitled to get resources) priority in the trace and accounts for most of the recorded disk I/O, so we speculate it includes some storage services;
- monitoring (10)
- normal production (9) — this is the lowest (and most occupied) of the priorities labeled 'production'. The trace providers indicate that jobs at this priority and higher which are latency-sensitive should not be "evicted due to over-allocation of machine resources" [16].
- other (2-8) — we speculate that these priorities are dominated by batch jobs; and
- gratis (free) (0-1) — the trace providers indicate that resources used by tasks at these priorities are generally not charged.

We separate usage by priority in preference to other indicators because the trace providers indicate that the priority affects billing, so we believe that it will reflect the importance of the tasks accurately.

Most of the usage and allocation in the cluster is accounted for by tasks running at the 'normal production' priority. This priority accounts for 56/73% of the CPU/memory usage and 62/64% of the CPU/memory allocations over the period of the trace. It also has the most noticeable diurnal and weekly patterns. Figure 2 (top) shows the production CPU usage normalized over a 24 hr period; the daily peak-to-mean ratio is approximately 1.3.

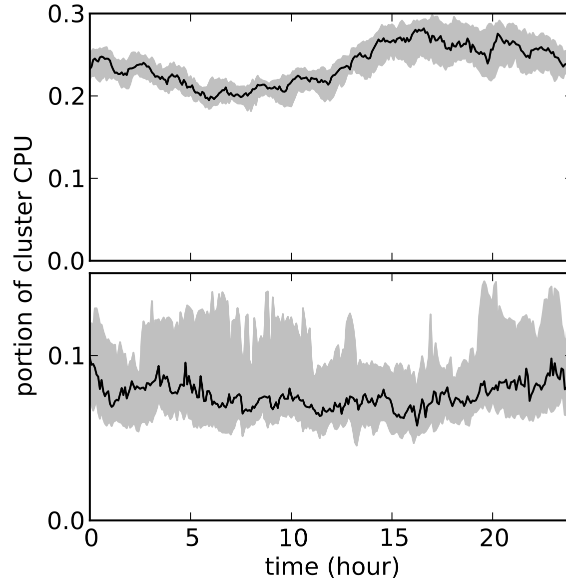

Figure 2: Normal production (top) and lower (bottom) priority CPU usage by hour of day. The dark line is the median and the grey band represents the middle quantile.

Usage at priorities higher than the normal production priority is small and stable. Usage at the monitoring priority is too small to be visible on Figure 1; it accounts for only 0.6% of the memory allocation (and similarly small amounts by other measurements). Utilization at the 'infrastructure' priority is larger (3% of the memory or 7% of the CPU), and shows little diurnal or weekly variation in its usage.

Usage at priorities below the production priorities is much more irregular. It is not dominated by any daily or weekly pattern but has a much larger peak-to-mean ratio than the production priorities. There is a significant event spike on days 21 through 25 of the trace, where CPU usage at non-production priorities increases by about a factor of two. This spike reflects the activity of only one user. The CPU usage spike is substantially larger than the amount of CPU requested by the tasks. Since the cluster's CPU is usually fully allocated, one can consider much of this CPU capacity effectively "borrowed" from the CPU allocation of higher priority tasks.

The variability in the non-production priorities reflects that, even though this is a large cluster, the activity of small numbers of users or jobs will have a large effect on overall performance. There are only 925 users (with usage recorded) over the trace. Although none of them dominate usage of the cluster over the month of the trace, a small number of users represent most of the usage overall. Though no user accounts for more than 13% of the CPU or memory usage, the top 10 users account for more than half the CPU and memory usage; the top 100 account for around 95%.

top 10 users:	$>$ 50% usage
top 100 users:	$>$ 95% usage
925 active users:	100% usage

3.2 Properties of jobs and tasks

To understand impediments that may prevent the scheduler from achieving higher cluster utilization, we further characterized the task load presented to the scheduler.

3.2.1 Job runtime durations

2% jobs: 80% usage

Most jobs are short, but short jobs contribute little to utilization. Even though less than 2% of jobs run for longer than one day, such jobs account for over 80% of the recorded usage by task-days, and requested CPU- or memory-days. About two-thirds of these long-running jobs are at production priorities and one-quarter at gratis priorities. Even though long-running jobs are a small subset of jobs, they represent many users: excluding gratis-priority jobs, 310 of the 925 users whose jobs have usage records are included, and including gratis priority jobs, 615.

Most (92%) of these long jobs come from the production priorities or the gratis priorities. Also, at production priorities, about half of the jobs are these long jobs; long jobs are a small percentage of the gratis priority jobs. This matches our expectation that the production priorities includes most important long-running services and that the gratis priorities include testing and development for all types of services. Other markers of job purpose are less conclusive. Each of the four "scheduling classes," which represent latency sensitivity, account for at least 14% of the day or longer jobs. Typical memory or CPU requests are larger for long jobs than for jobs overall, but still have a similar dynamic range.

About two-thirds of the jobs run for less than five minutes and one-fifth for less than one minute. These short jobs dominate the 'other' priority set. Like the long jobs, these short jobs are not dominated by a small subset of users.

3.2.2 Request shapes

Most jobs in the trace make relatively small resource requests, though utilization is dominated by somewhat larger jobs. Very little of the utilization is taken up by tasks that require nearly a whole machine (although such tasks exist), but most of the utilization is not "sand". In terms of resource requests, the workload is diverse; the utilization is not dominated by jobs with any particular request profile.

The trace contains approximately 650k jobs (which run and have usage records). About 75% of jobs only run one task, and the median length of a job is around 3 minutes. Others of these single-task jobs may support larger jobs; for example, the trace providers indicate that a MapReduce program will run with a separate job for the MapReduce master. Some of these single-task jobs likely arise from users running non-distributed programs or splitting an embarrassingly parallel computation across many jobs.

Similarly, most jobs request less than 2.5% of a typical machine's memory per task. These small, short jobs do not account for most of the utilization of the cluster, however. By just about any measure of how much of the cluster is used (task-days, memory- or CPU-days requested), jobs with 1 task, running for less than ten minutes, or requesting less than 5% of a typical machine per tasks account for well below 5% of the machine time on the cluster. Indeed, only jobs with at least hundreds of tasks contribute significantly to cluster resource allocation.

Even weighting by the portion of the total utilization they account for, requests for most of a typical machine are rare as can be seen in Figure 3. A majority is accounted for by tasks that each request (in both CPU and memory) less than a fifth of a typical machine. There is a wide variety of task CPU and memory request 'shapes': the middle quantile (for jobs) of ratios between memory and CPU requests varies by a factor of 7, and the middle quantile of per-task CPU and memory request sizes vary by a factor of 4 (memory) to 7 (CPU). There is a weak positive correlation between the size of per-task CPU requests and per-task memory requests ($R^2 = 0.14$ for a linear regression over jobs).

Though memory and CPU requests follow the rule of lots of small requests and a few large ones, neither appears to be well-approximated by a power law distribution. Memory requests are better modeled by a log-normal distribution, but a continuous distribution will not correctly model the special popularity of certain,

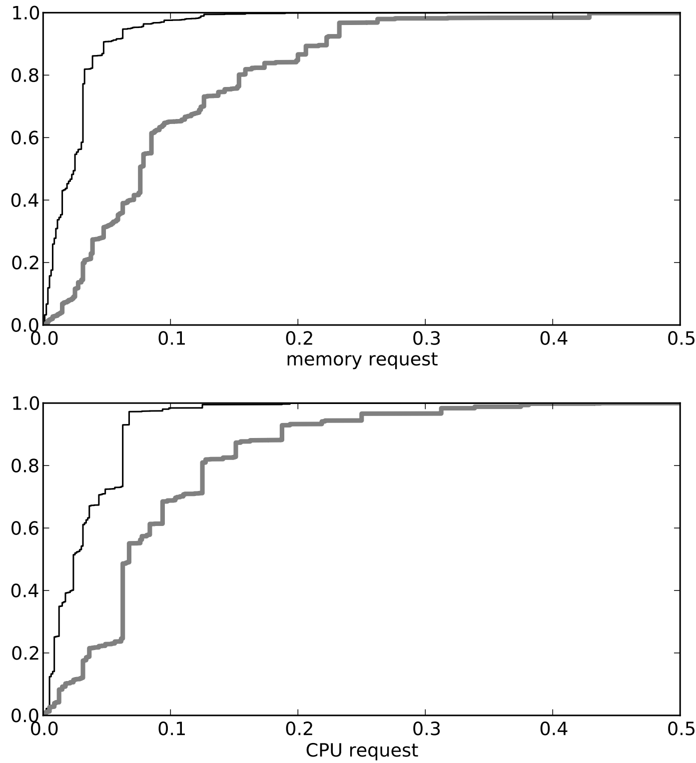

Figure 3: Weighted CDFs of memory (top) and CPU (bottom) requests in terms of the number of tasks with those requests (thin line) and the portion of the memory allocation used by tasks with those requests (thick line).

likely round number values. No such simple continuous distribution seems to be a good approximation to the distribution of CPU requests, based on our analyses.

The frequency of job submissions might suggest that many instances of the same program are being run. We can approximately identify repeated jobs 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, however, indicate how often different programs are likely to have the same 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 Zipf-like distribution; most names appear exactly once and the most common job name appears 22k times. Some numbers of job repetitions, however, are more popular than predicted by a Zipf-like distribution. Some of these peaks correspond to the number of days in the trace, so they may indicate some daily jobs. We suspect other peaks arise from sets of related jobs that run repeatedly.

There are approximately 14k unique request shapes (in terms of the CPU and memory request); no single task resource request shape accounts for more than 7% of jobs. We examined the ten most common request sizes. Six of the ten most common request sizes each represent less than 200 unique names even though they each account for tens of thousands of jobs. But, since all but one of these very common request shapes represent at least tens of jobs, we believe they do not simply arise from one application's requirements.

Job submissions are dominated by the lower priorities. Even though they account for most usage, the production priorities only account for 7% of jobs and 1% of tasks. This reflects that jobs running at these priorities run for longer than most of the lower-priority jobs. Over 50% of job submissions are in the other priorities, where job durations are shortest. For this reason, we speculate that the other priorities include

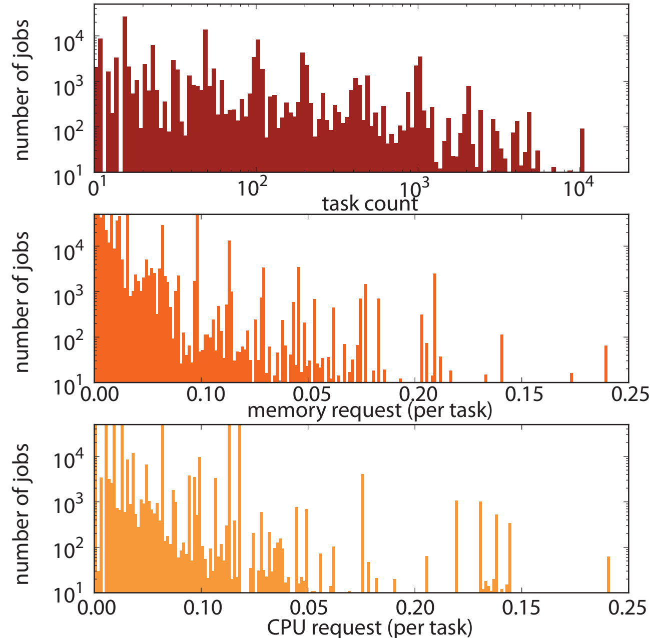

Figure 4: Histograms of job counts by task count (top), memory request size (middle) and CPU request size (bottom). Note the log-scale on the y-axis and the log-scale on the top plot's x-axis. We speculate that many of the periodic peaks in task counts and resource request sizes represent humans choosing round numbers. Memory and CPU units are the same as Table 2. Due to the choice of x-axis limits, not all jobs appear on these plots.

batch processing jobs. The remaining approximately 40% come from jobs at the gratis priorities.

3.2.3 Bias due to human influence

Job parameters show many indications of being selected by humans. Most notable are the sizes of jobs. Round numbers of tasks — such as 16, 100, 500, 1000, and 1024 — are disproportionately popular numbers of tasks. We can thus infer that the number of tasks being run is usually not selected through an automated process. Like resource requests sizes, there are a variety of other task counts in use.

For the resource requests, it is more difficult to identify whether the requests are for round numbers since the units are rescaled. Nevertheless, we can identify several evenly spaced bands of popular requests as seen in Figure 4. For memory, we believe that these likely represent round numbers (for example, multiples of 100 megabytes). It seems unlikely that a scheme based on measuring the memory requirement would choose such round numbers, so we believe that these requests are made by humans.

For CPU, there are similar patterns. But, some of the "bands" likely represent whole numbers of CPUs, which would be reasonable to choose even if the requests were chosen based on measurements. [17] We believe it is unlikely that this accounts for all of the especially popular CPU request sizes, because it would suggest that typical cluster machines have 40 to 80 cores, which seems unrealistically high for commodity hardware that would be in active use in May 2011. It does not, however, appear to be the case that jobs that

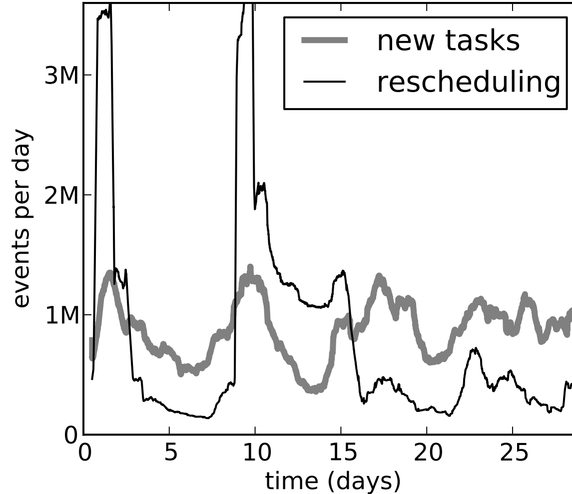

Figure 5: Moving average (over day-long window) of task submission (a task became runnable) rates.

have a common number for their CPU requests are much more likely to have a round number for its memory request.

3.2.4 Job arrival rates

Jobs arrive at the cluster approximately continuously. The median inter-arrival period is less than 4 seconds, and almost all job inter-arrival times are less than 5 minutes. Job inter-arrival times roughly follow an exponential distribution except that very small inter-arrival periods (seconds) are more common than a simple exponential distribution would predict. We speculate this skew comes from related sets of jobs that communicate with each other. Using hourly buckets, the peak-to-mean ratio of job arrivals is 2.5. There is a regular weekly peak in overall job arrival rates which accounts for the entire peak-to-mean ratio.

3.2.5 Task resubmissions and outcomes

Much of the scheduler's workload comes from *resubmitted* tasks. We consider a task resubmitted when it becomes runnable after terminating for any reason. Although it is possible for users to resubmit tasks manually, most resubmissions are likely automated. The terminations preceding most resubmissions are marked as either *failures* (software crashes of a task) or as *evictions* (task ceased to fit on the machine, due to competing workload, over-commitment of the machine, or hardware failures). Other possible reasons for terminations include *kills*, where the underlying reason for a task's death is not available, and *finishing*, where the program executed in a task exits normally. Some kill events may logically represent failures or evictions, and so we may undercount the true failure or eviction rate.

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 configured to be restarted shortly after failing. Of the 14M task failures recorded in the trace, 10M are in three crash looping jobs, each of which has tens of thousands of tasks. The length of these three large crash-loop examples 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 allocated is accounted for by jobs that experience more than 10 failures per task over the course of the trace. Most of the failures occur in lower priority jobs — especially the gratis priorities

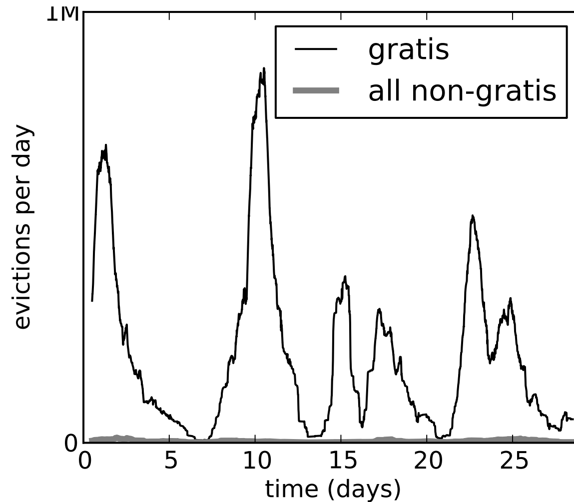

Figure 6: Moving average (over day-long window) of task eviction rates, broken by priority.

that are probably used for development, but there are some noticeable (but smaller in terms of task failure count) crash loops even at the normal production priority.

After crash loops, evictions are the next most common cause of task rescheduling. As would be expected, the eviction rate is heavily related to a task's priority. The rate of evictions for production tasks is very low: around one per one hundred task days. (If we assume that all non-final kill events for production-priority tasks are evictions, then we obtain a maximum eviction rate of one per 14 task-days.) We suspect that most of these evictions are due to machines becoming unavailable for maintenance; about 3% of the machines are briefly removed from the cluster each day, most likely for software upgrades.

The rate of evictions at lower priorities varies by orders of magnitude, with a weekly decrease in the eviction rate. Because of the variation in this eviction rate, we believe that almost all these evictions are to make space for higher priority tasks. Gratis priority tasks average about 4 evictions per task-day, though almost none of these evictions occur on what appear to be weekends. Given this eviction rate, an average 100-task job running at a gratis priority can expect about one task to be lost every 15 minutes; these programs must tolerate a very high "failure" rate by the standards of typical cloud computing provider or Hadoop cluster.

A majority of tasks that terminate without being restarted, terminate normally. 73% of tasks are marked as terminating successfully, and most other tasks are marked as killed (so we cannot determine if they terminated 'normally'). Even though rescheduling due to failures is common, failing tasks and jobs are relatively rare: about 2% of jobs and tasks. The median failing job is about twice as long (five and half minute) as the median job overall, while the median length of jobs that are successful is slightly shorter (about two minutes). The longer length of failed jobs can be partly explained by the retry on failure policy that is responsible for crash loops: about half of the failing jobs reschedule their tasks on average at least once; these jobs have a median duration of around 15 minutes.

But, because the workload is dominated by long-running jobs, this does not characterize most of the utilization. 40% of the memory allocation is contained in tasks that do not complete in the trace. Of long-running (more than half a day) tasks that do terminate, most of them are marked as killed. We speculate that these most likely represent long-running services that are killed by some user or external script. 2% of the memory usage within these long-running jobs is accounted for by jobs that experience about as many task failures and task starting events. These tasks generally finally terminate as killed, probably because some human or program recognizes that the job is stuck failing and kills it rather than the tasks reaching a threshold

for their maximum number of failures.

4 Resource requests versus usage

Resource requests clearly do not estimate the aggregate usage on this cluster. The overall memory usage is only 53% of the corresponding allocation, and the CPU usage only 40%. A scheduler that trusted these requests would have difficulty achieving high utilization. The requests are not consistently wrong, so simply assuming a fixed “fudge factor” would not work. For example, aggregate CPU usage at the gratis priorities ranges from around twice to half of the CPU requests, and individual jobs vary much more.

Requests to the scheduler in this cluster are supposed to indicate the “maximum amount . . . a task is permitted to use” [16]. A large gap between aggregate usage and aggregate allocation, therefore, does not necessarily indicate that the requests are inaccurate. If a task ever required those CPU and memory resources for even as little as a second of its execution, then the request would be accurate, regardless of its aggregate or subsequent consumption. Thus, resource requests could be thought of as reflecting the maximum *anticipated* utilization of CPU and memory for the requesting task.

It is difficult to infer from the type of data available in the trace whether a task *required* some amount of CPU or memory resources. One problem is that observed usage can either over- or under-indicate actual requirements. A task that is given more resources than it ‘needs’ may use them: for example, a web server might be able to cache more in its excess memory or burst to use more CPU than is required to meet its SLO. Conversely, given too few resources, a task might still run, albeit less efficiently. For example, given insufficient memory, it may thrash or experience garbage collection delays. With insufficient CPU, it may fail to meet its latency requirements. Additionally, interactive jobs may need some headroom to handle a rare worst case (e.g., denial of service) scenario.

For some types of programs, it does not make sense to talk about how much resources it requires or should be permitted to use. For example, a batch program may be able to make progress in proportion to the amount of CPU resources it is given. A program that makes use of a cache or lookup table may be able to similarly trade off memory resources.

Without the ability to rerun programs and understand their performance objectives, we cannot find the optimal resource request for applications. We will make some approximate best guesses from the actual usage we observe: we will assume the maximum or some high percentile of the (usually five minute long) 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 account for this, we will take a high percentile of the estimates for each task as the estimate for resource request within each task.

4.1 Request accuracy

The differences between the resource request we would estimate from usage and the actual request is not what one would infer from Figure 1. 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, accounting for about a fifth of the total memory allocation. (Memory requests rarely under-estimate their jobs’ memory utilization by a large factor, probably because tasks are killed when the memory request is exceeded by too much.) But the total difference between the memory allocation and its utilization is around 50% of the memory requested.

CPU usage is clearly not as constrained by the request as memory usage. Many jobs have tasks that briefly use much more CPU than the job requests. These spikes are visible looking at usage samples which usually account for five minutes of usage. (Some samples are shorter; for example, a task may not remain

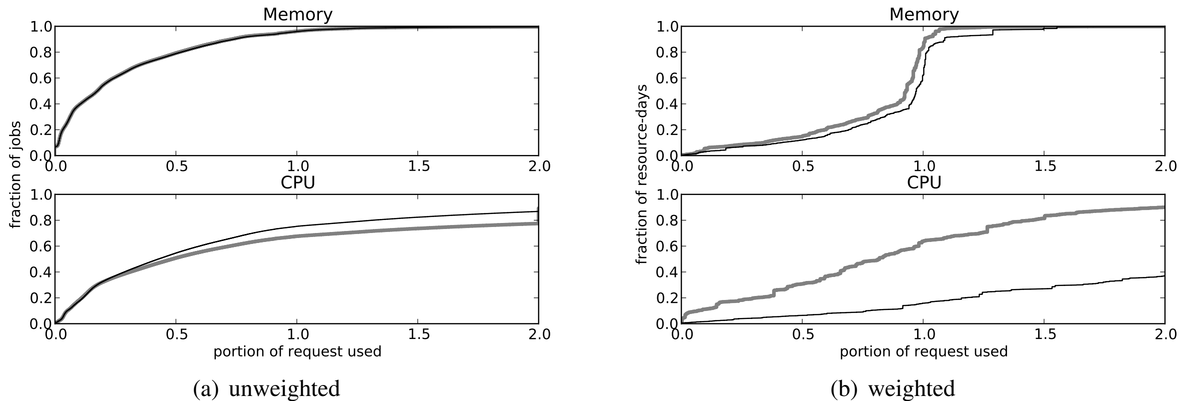

Figure 7: 7(a) shows the CDF of the maximum five-minute usage sample relative to the resource request for the corresponding job. The thin line represents the largest five-minute sample within each job. The thick line discards outlier tasks within a job; for each job, it shows the 99th percentile of the largest five-minute usage measurements for each task. The top graphs shows memory usage; the bottom graphs show CPU usage. 7(b) shows the same CDFs as 7(a), but with each job weighted by its per-task request times the number of task-days its tasks run.

running for all of the five-minute sampling period.) To avoid being biased by rare CPU spikes or short samples, we will primarily look at 99th percentiles of these samples. Even looking at the 99th percentile shows that many jobs have some task use more CPU than the job requests. Since there are many small jobs which make insignificant contributions to cluster usage, we will usually weight each job by the size of its per-task resource request multiplied by the number of task-days for which its tasks run. As shown in figure 7(b), thus 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. But, otherwise, there appears to be little relationship between the actual sustained CPU usage and the CPU request.

This estimate of CPU usage undercounts the requirements of jobs that require periodic CPU bursts rather than sustained CPU capacity. For each of the usually five minute usage samples, this trace includes data about the maximum one-second sample of the CPU usage in that sample. As this represents 99.7th (top 1 second of 300 seconds) percentile CPU usage within each five-minute window, it is very noisy. These maximum samples are often five or more times a job's CPU requests (we trip the x-axis limit at 2.0). This is true even after filtering out jobs with small CPU requests, low scheduling classes or priorities and selecting the median of the samples.

We assume that most CPU requests allow jobs to at least meet their performance requirements. Thus, it seems unlikely that latency-sensitive jobs actually require CPU bursts of four to five times their CPU request. We are doubtful, therefore, that the maximum CPU usage measurements can be used to infer the accuracy of the CPU requests.

4.2 Variation within tasks and jobs

Even if users could set the optimal resource request for their jobs, the gap between the overall allocation and usage would remain large. Comparing the maximum usage for each job with the actual usage of the tasks within the job, we find that any scheduler which actually sets aside resources equal to each running tasks' request cannot come close to fully utilizing these resources. Even for memory usage (which, in aggregate, is relatively constant), choosing the "correct" memory requests for each job would leave about 40% of the

memory allocation (in terms of byte-seconds) unused.

One source of this problem is that resource usage is not the same between tasks in a job or over time within a job. Particularly, even if most tasks in a job use the same amount of resources most of the time, there may be rarer outliers in resource usage that would define the corresponding resource request. If users understood their resource usage well enough, one might imagine that users could provide adjustments for their resource allocation over time or between tasks in the job. The system from which the trace was extracted supports this, but it 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 high percentile of the five-minute usage samples are much larger than average usage — both between tasks in a job and within tasks over time. Surprisingly, this difference exists even for memory utilization: differences between the maximum usage of tasks within jobs account for about 20% of the total memory allocation (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 exceed the memory allocation, the sum of these percentages can exceed the allocation/usage gap.)

Much of this difference is due to extreme outliers in resource usage. There is a large difference between the 99th percentile measurements, often about as large as the difference between the 99th percentile and the median. This difference exists even among long-running jobs, so it is unlikely to be from start-up effects where memory usage needs to ramp up. Outside of high percentile measurements, memory usage is much more stable, and, in particular, the memory usage of long-running tasks in one hour is usually within a small percentage of its memory usage in the next hour.

5 Challenges in Task Scheduling

In this section we analyze task scheduling delays and identify potential causes. Scheduling latency is characterized by priority as well as the presence or absence of constraints.

5.1 Scheduling Delay

Scheduling a wide diversity of workloads on a relatively full cluster is challenging. In addition to needing to find a machine where the requested task shape fits, the scheduler must satisfy priorities and any specified constraints. This trace is unique in providing anonymized yet expressive information on task placement constraints. It is likely that some of these constraints will make tasks difficult to schedule. We assessed this difficulty by examining how scheduling delay relates to the constraints placed upon a request.

Before examining task constraints, though, it is useful to have a general picture of how the task scheduling delay behaves across the trace. Figure 8 shows a cumulative distribution of scheduling delays over all scheduled task instances. Unlike jobs, tasks exhibit longer scheduling delays that range from 0 to more than 20 days. As a point of reference, we note that more than 10% of tasks have delays longer than 10 seconds, suggesting that there is room to optimize schedulers with this metric in mind.

The long tail of this CDF stretches to the maximum observed task scheduling delay of 20.3 days. Long scheduling delays are particularly interesting to us when they are symptomatic of task-specific constraints that cannot be easily met rather than just over-utilization of physical resources or delaying of tasks with low priorities. The most-delayed task (>20 days) had the highest possible priority and modest resource requests, thereby discounting those as potential causes. More generally, the 100 worst scheduling delays are a roughly evenly split mix of highest and lowest priorities—a counter-intuitive result, inviting a closer look at task priorities.

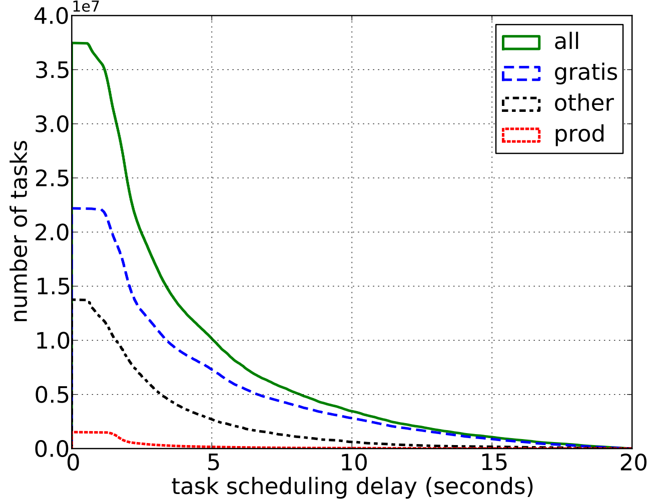

Figure 8: Task scheduling delay: reverse cumulative distribution. For each scheduling delay X, we plot the number of tasks that experienced a scheduling latency of X seconds or more for all (green) tasks and 3 subsets by priority: gratis (blue), other (black), and production (red).

5.2 Scheduling Delay by Priority

A look at worst-case examples suggests that having a high priority does not ensure short scheduling latencies. Figure 9 plots a cumulative distribution function of scheduling delay broken down by the major priority groups, namely *gratis*, *other*, and *production*. As the priority increases, the CDF curve becomes more convex—for a given task scheduling latency (on the x-axis), higher priority tasks have a lower probability of reaching such a delay. The highest probabilities of delay intuitively correspond to the gratis priority class and vice versa.

This was also confirmed by tabulating mean delays for each of these priority groups (see Table 3).

priority group	mean scheduling delay (s)
gratis (0-1)	242.9
other (2-8)	202.3
production (9-11)	52.2
9-10 only	5.4

Table 3: task scheduling delay by priority group

5.3 Constraints and Anti-Affinity

Even though priority classes correlate well with scheduling delay, a higher resolution look and a breakdown by individual priorities reveals a noisier picture. In particular, the scheduling latency for the highest priority group is the highest of all groups: 7.43 hrs on average. That is 2 orders of magnitude higher than lowest priority tasks, and exceeds the *production* priority group by 3 orders. Given the ability of the scheduler to revoke lower priority tasks as well as a generally non-imposing shape of these tasks, there must be additional factors. One strong possibility is the presence of the "anti-affinity" constraint found in all task instances with

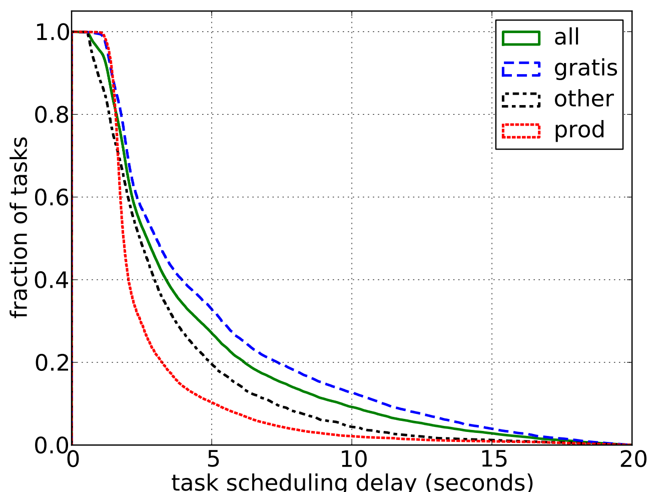

Figure 9: Task scheduling delay: reverse CDF. For each scheduling delay X, we plot the *fraction* of tasks that experienced a scheduling latency of X seconds or more for all (green) tasks and 3 subsets by priority: gratis (blue), other (black), and production (red). Note that with increase in priority, the probability of a larger scheduling delay decreases.

priority 11, which means that all tasks of the given job must run on different machines. There are also various other constraints that could be responsible for the scheduling delays.

It is interesting to note that there are about 25M unique tasks in the trace, but less than 6% of them have any constraints. For those that do, we've established that the number of unique constraints does not exceed 6, with fewer constraints being substantially more common, as can be seen in Table 4. Constraints in the trace

task count	unique constraint count
24 019 159	0
1 368 376	1
33 026	2
2 782	3
1 352	4
30	5
6	6
25 424 731	17

Table 4: Task count by the number of unique constraints specified.

are specified as an attribute key, value, and comparison operator tuple on a per task basis. The uniqueness of the constraint 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.

From our earlier experience with priorities, we took the presence/absence of constraints as the first-order predictor and discovered that unconstrained tasks are scheduled 1.5 times faster (211 seconds) than tasks with constraints ($>$ 300 seconds), on average.

Finally, the anti-affinity requirement has already been noted as capable of substantially affecting the schedulability of a task. As can be seen in table 5 and Figure 10, the absence of anti-affinity constraint

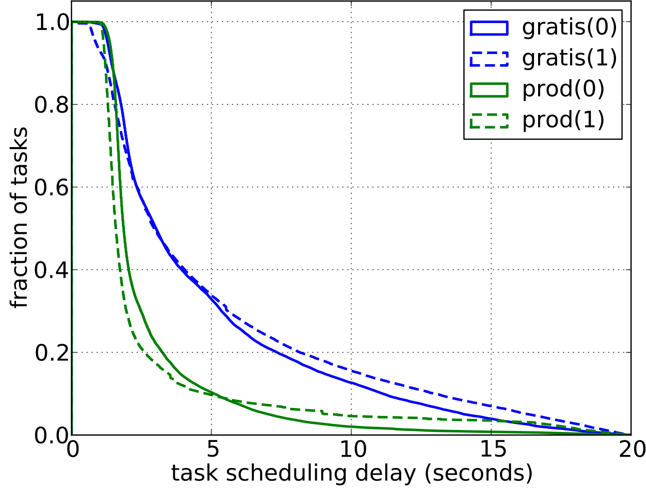

Figure 10: Task scheduling delay CDF broken down by both priority and anti-affinity requirement. The dashed line and "(1)" indicates those tasks with anti-affinity constraints.

	anti-affinity off		anti-affinity on	
Priority	task count	delay (s)	task count	delay (s)
0-1	28 002 936	233.49	260 809	892.19
2-8	14 148 983	244.19	2 945 288	11.40
9-10	1 558 255	5.06	80 861	13.19

Table 5: anti-affinity and priority as predictors of scheduling latency

correlates very well with a decrease in scheduling latency. The difference ranges from a factor of 2 (production priority) to a factor of 4 (*gratis* priority). In Fig. 10, we've broken it down by priority group (color) as well as anti-affinity (linestyle). It can be seen that starting from as early as a 2s delay (for *gratis* workloads) and 5s delay for *production*, a greater number of constrained tasks have a given task scheduling delay than tasks without anti-affinity.

The presence or absence of the anti-affinity constraint has thus also been shown to correlate with higher or lower scheduling latency respectively.

5.4 Pending tasks

Figure 11 is a high-level visualization of the cluster queuing delay by calculating the number of pending tasks on a minute-by-minute granularity. No smoothing was performed, preserving short-lived spikes in the data. Despite the noise, a weekly pattern clearly emerges, as the queuing delay starts rising in the beginning of each 7-day period and is falling off towards the end of the weekend. This pattern is strikingly consistent, including on the first day of the fifth week on day 29.

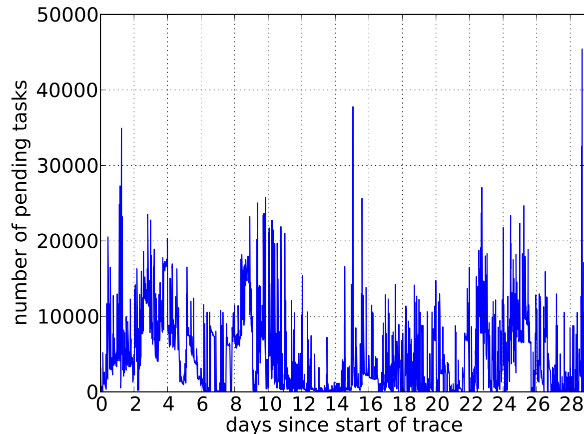

Figure 11: Task queue length over time.

6 Related Work

The corpus of computer science literature includes many examples of trace-based analysis. To our knowledge, this paper presents the most complete study of a commercial, multipurpose cluster of this size and over this duration. Perhaps the closest related work are studies of previous Google cluster traces. Chen *et al.* [7] analyzed a trace which lacked task request information and only spanned a 6.25 hour time period. Zhang *et al.* [21] analyzed a limited 5 day trace which included several clusters to determine an appropriate model for run-time task resource usage that would reproduce aggregate resource usage and scheduling delays. They found a model using only the mean and coefficient of variation within each of their task classes was sufficient for that purpose. Sharma *et al.* [19] used trace samples from three Google clusters that span 5 days to form a baseline against which synthetic workloads were compared.

Most other trace-based studies of a comparable size come from the high-performance computing or grid domains (see [2] and [1] for trace repositories). Some of the phenomena observed in the present study have analogues in these parallel, batch environments. For example, Downey *et al.* [9] found discrete distributions of job parameters and a similar high count of small jobs together with resource usage dominance by large jobs. More recent analyses [12, 6] of MapReduce logs has found a similar prevalence of many small jobs. Disagreement between user-specified estimates and reality have been observed in scientific computing clusters, but usually in terms of runtime estimates [4]. Although they could not have resource requests, examinations of actual CPU and memory usage in Hadoop clusters have found similar diversity in resource requirements [10].

The high-performance computing community has conducted extensive analysis of failure data from large batch systems [18, 15, 13]. Some non-scheduler traces of heterogeneous datacenters have been analyzed. For example, Benson *et al.* [5] analyzes network traffic traces from several multipurpose datacenters, identifying the mix of applications from the network activity.

7 Summary and Conclusions

The workloads served by large, multi-purpose clusters vary across most dimensions, creating new challenges for schedulers. This paper analyzes the Google cluster trace, which captures the demand on and activity of over 12,500 machines over 29 days, to better understand these challenges. Our analysis confirms popular anecdotes about ‘boulders’ and ‘sand’ among resource requests, with some tasks lasting for weeks but most

lasting no more than a few minutes; a few tasks require entire machines, but most – only a small fraction of a machine; some jobs have many component tasks, but most jobs have only one.

Our analysis also identifies and examines two specific characteristics that may induce new scheduler designs. First, we observe significant mismatches between resource requests and actual resource usage, which make it difficult for a scheduler to effectively pack tasks onto machines to achieve maximum utilization. Clearly aware of conservative resource allocation over-estimates, the traced scheduler over-commits machines and relies on eviction to increase throughput. It thus achieves a commendable 40–60% average utilization, leaving some room for further improvement in future work. Parts of this mismatch appear to result from inaccurate human resource request specifications, but parts appear to be inherent variation within tasks and jobs. Second, we observe substantial delays in the scheduling of some requests (e.g., up to 20 days), which we attribute to difficult-to-satisfy task constraints.

The analyzed trace, including the workload mix and the scheduler characteristics, represent a single noteworthy data point for those studying large, many-purpose heterogeneous commercial clusters. We believe that many of the insights garnered from our analyses will apply broadly to many such clusters. As such, we believe that there is substantial opportunity for new approaches to scheduling that explicitly address the increasing challenges of intra-cluster workload heterogeneity and platform heterogeneity.

References

[1] The Grid Workloads Archive. http://gwa.ewi.tudelft.nl/.

[2] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

[3] lxc Linux containers, 2012. http://lxc.sourceforge.net/.

[4] Cynthia Bailey Lee, Yael Schwartzman, Jennifer Hardy, and Allan Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'04, pages 253–263. Springer-Verlag, 2005.

[5] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th Annual Conference on Internet Measurement*, IMC '10, pages 267–280. ACM, 2010.

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

[7] Yanpei Chen, Archana Sulochana Ganapathi, Rean Griffith, and Randy H. Katz. Analysis and lessons from a publicly available Google cluster trace. Technical Report UCB/EECS-2010-95, EECS Department, University of California, Berkeley, Jun 2010.

[8] Committee on Oversight and Government Reform. Cloud computing: benefits and risks of moving federal IT into the cloud, July 2010. http://oversight.house.gov/index.php?option=com_content&task=view&id=5012&Itemid=2.

[9] Allen B. Downey and Dror G. Feitelson. The elusive goal of workload characterization. *SIGMETRICS Performance Evaluation Review*, 26:14–29, March 1999.

[10] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *Proceedings of the 8th*

USENIX Conference on Networked Systems Design and Implementation, NSDI'11, pages 24–24. USENIX Association, 2011.

[11] Michael Goodrich. GSA presentation on the federal cloud computing initiative. Software & Information Industry Association (SIAA) panel, June 2010. http://www.siia.net/blog/index.php/2010/06/gsa-presentation-on-the-federal-cloud-computing-initiative/.

[12] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 94 –103, May 2010.

[13] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The failure trace archive: enabling comparative analysis of failures in diverse distributed systems. In *Proc. of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 398 –407, May 2010.

[14] Huan Liu. A measurement study of server utilization in public clouds. In *Proceedings of International Conference on Cloud and Green Computing (CGC)*, Dec 2011. https://sites.google.com/site/huanliu/.

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

[16] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema, 2011. http://goo.gl/5uJri.

[17] Charles Reiss, John Wilkes, and Joseph 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.

[18] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proc. of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2006)*, pages 249–258, 2006.

[19] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 3:1–3:14. ACM, 2011.

[20] John Wilkes and Charles Reiss. Details of the ClusterData-2011-1 trace, 2011. https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1.

[21] Qi Zhang, Joseph L. Hellerstein, and Raouf Boutaba. Characterizing task usage shapes in Google's compute clusters. In *Proc. of Large-Scale Distributed Systems and Middleware (LADIS 2011)*, September 2011.