# Characterizing Machines and Workloads on a Google Cluster

Zitao Liu and Sangyeun Cho
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260, USA
{ztliu,cho}@cs.pitt.edu

***Abstract*—Cloud computing offers high scalability, flexibility and cost-effectiveness to meet emerging computing requirements. Understanding the characteristics of real workloads on a large production cloud cluster benefits not only cloud service providers but also researchers and daily users. This paper studies a large-scale Google cluster usage trace dataset and characterizes how the machines in the cluster are managed and the workloads submitted during a 29-day period behave. We focus on the frequency and pattern of machine maintenance events, job- and task-level workload behavior, and how the overall cluster resources are utilized.**

***Keywords*-Cloud computing, datacenter computing, system management, job scheduling.**

## I. Introduction

The last decade has seen a surge of interest and commercial developments in cloud computing (large-scale distributed data processing). Companies like Google, Facebook and Amazon routinely process petabytes of web and user data using distributed computing frameworks such as MapReduce and Hadoop [1], [2]. They expose ample coarse-grain parallelism and harness large clusters of machines. Cloud computing services are also available to enterprise users and individuals, like Amazon's EC2. The low cost, elastic scalability and robust performance makes cloud computing fast become a backbone of the society and necessity for everyday Internet uses.

Despite the popularity of cloud computing, designing and managing a large cloud computing system cost-effectively remains hard problems. For example, Barroso and Hölzle [3] list four challenges: rapidly changing workloads, building balanced systems with imbalanced commodity components, curbing energy usage and maintaining high parallel efficiency in the presence of mismatches in performance and cost trends of hardware components. Addressing each challenge requires detailed understanding about *how a cloud computing infrastructure is utilized under real workloads* first.

Recently, in November 2011, Google released a "cluster usage trace dataset" that records extensive machine and workload events on a large production cluster [4]. Compared with the trace released in late 2009 by the same company [4], the new dataset has vastly richer information that is collected during a much longer period (29 days vs. 7 hrs). This work analyzes the trace dataset with three focuses: We first study *how machines in the cluster are managed*. According to the dataset, individual machines may become unavailable and then available, to get upgraded, for instance, or because of hardware failures. We then look at *how jobs are scheduled and processed in the cluster*. The dataset allows us to obtain job mixes and reason about the job scheduling properties, e.g., when and how often jobs are "killed" and rescheduled. Lastly, we investigate *how the cluster resources are utilized*. Especially, we estimate the amount of useful, wasted and idle resources.

## II. Google Cluster Usage Trace

This section briefly introduces the Google cluster usage trace dataset (or simply "trace dataset") itself. Readers are referred to Reiss et al. [5] for full details.

### A. Google cluster

In Google datacenters, a *cluster* is comprised of machines that are connected by a high-bandwidth cluster network. A *cell* is a set of machines, typically all in a single cluster, sharing a common cluster management system that allocates work to machines. Work arrives at a cell in the form of *jobs*. For example, "Map" and "Reduce" could each become a job. A job is comprised of one or more *tasks*, each of which is accompanied by a set of resource requirements used toward scheduling and may run on one or more cores. Each task represents a Linux program, possibly involving multiple processes, to be run on a single machine. Tasks that belong to a common "parent" job are typically an identical binary and have the same resource usage requirements. *Users* of the clusters are typically Google employees and services.

### B. Scheduling jobs and tasks

Jobs and tasks are a unit of scheduling. Each job or task has a life cycle of four different states, as visualized in Fig. 1. State transitions occur on scheduler or machine events like: *Submit*, *Schedule*, *Evict*, *Fail*, *Finish*, *Kill*, *Lost*, *Update_Pending* and *Update_Running*. Among these events, Schedule (following Submit), Finish, Kill and Fail are the most frequent. A job or task may experience one or more terminating events like Fail and Kill before it is normally finished. We will closely examine job/task scheduling events in Section IV.

TABLE I
BASIC STATISTICS OF THE SIX DATA TABLES.

| Data Table Name | Machine Event | Machine Attr. | Job Event | Task Event | Task Constraint | Task Usage |
|---|---|---|---|---|---|---|
| # files | 1 | 1 | 500 | 500 | 500 | 500 |
| # data entries (total) | 37,780 | 10,748,566 | 2,012,242 | 144,648,288 | 28,485,619 | 1,232,792,102 |
| Avg. entries per file | 37,780 | 10,748,566 | 4,024.5 | 289,296.6 | 56,971.2 | 2,465,584.2 |
| # data attributes | 6 | 5 | 8 | 13 | 6 | 19 |
| Zipped Size | 339 KB | 136 MB | 83 MB | 1.5 GB | 147 MB | 36.6 GB |
| Unzipped Size | 3 MB | 1.1 GB | 315 MB | 15.4 GB | 2.8 GB | 158 GB |

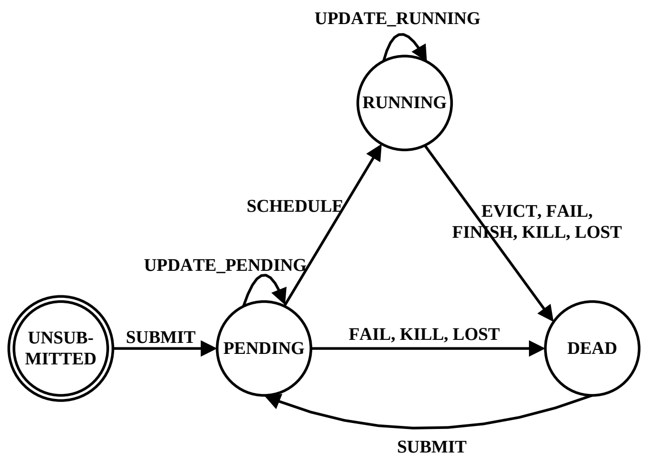

Fig. 1. State transitions for a job or task.

## C. Dataset organization and limitations

The Google trace dataset is comprised of six separate data tables: *Machine Event*, *Machine Attribute*, *Job Event*, *Task Event*, *Task Constraint* and *Task Usage*. The tables record various machine events, machine attributes (such as kernel version), scheduling events, scheduling constraints and resource usages, collected during a 29-day period in the month of May 2011 in one of Google's production cluster cell. The cell size is significant, with more than 12k distinct machines. Table I summarizes the basic statistics of these tables.

There are limitations in the trace dataset because certain information was obfuscated for confidentiality reasons [5]. Despite the sanitization done on the dataset, we felt that the trace dataset gives extremely useful information about the cluster usage. For instance, while we do not know actual user names, we can derive how many jobs were submitted by a particular user because the user name stays the same (while being a random string). Likewise, resource capacity scaling does not limit our ability to gain insight about the machine population mix and the temporal fluctuations in the cluster resource usage.

## III. MACHINE USAGE

This section considers *how machines in the studied cluster are managed* based on the data in the Machine Event table. It lists 12,583 distinct machines in total (12,477 machines initially). The table has 37,780 machine events of three different types:

**1. ADD**: a machine became available to the cluster; there are in total 8,966 ADD events (excl. "initial ADDs").

**2. REMOVE**: a machine was removed from the cluster and is unavailable for service; there are 8,957 instances.

**3. UPDATE**: a machine had its available resources changed; there are in total 7,380 UPDATE events.

### A. Machine population

Each machine in the dataset is characterized by its *capacity*. A machine's capacity is represented with a tuple, <CPU capacity, memory capacity>. Each variable in the tuple has a "normalized" value that is larger than 0 and at most 1 [4]. In the studied dataset, we found three distinct CPU capacity values: '0.25', '0.5' and '1'. In the case of memory, we saw ten different values: '0.03085', '0.06158', '0.1241', '0.2493', '0.2498', '0.4995', '0.5', '0.749', '0.9678' and '1'. Notice that we can naturally group the memory capacity values around five levels, '0.125', '0.25', '0.5', '0.75' and '1'. Table II counts initially available machines in 15 groups having the same CPU and memory capacity.

It is shown that a majority (93%) of the machines in the cluster have a CPU capacity value of '0.5'. We do not know how many CPU cores this relative value corresponds to; however, bulk compute capabilities of the cluster appear to come not from the highest capacity machines but from the medium capacity machines. A small number of the high capacity machines might have replaced some older machines with smaller capacities. This cluster's machines are fairly homogeneous, which we believe relaxes the complexities in scheduling jobs and tasks. Moreover, it would ease machine maintenance (e.g., fewer types of spare components need be in stock.)

TABLE II
NUMBER OF MACHINES IN DIFFERENT CAPACITY GROUPS.

| Mem.\CPU | = 0.25 | = 0.5 | = 1.0 | **Total** |
|---|---|---|---|---|
| $\leq$ 0.125 | 0 | 60 | 0 | 60 |
| ∼0.25 | 123 (1%) | 3,835 (31%) | 0 | 3,958 (32%) |
| ∼0.5 | 0 | 6,672 (54%) | 3 | 6,675 (54%) |
| ∼0.75 | 0 | 992 (8%) | 0 | 992 (8%) |
| ∼1.0 | 0 | 4 | 788 (6%) | 792 (6%) |
| **Total** | 123 (1%) | 11,563 (93%) | 791 (6%) | 12,477 (100%) |

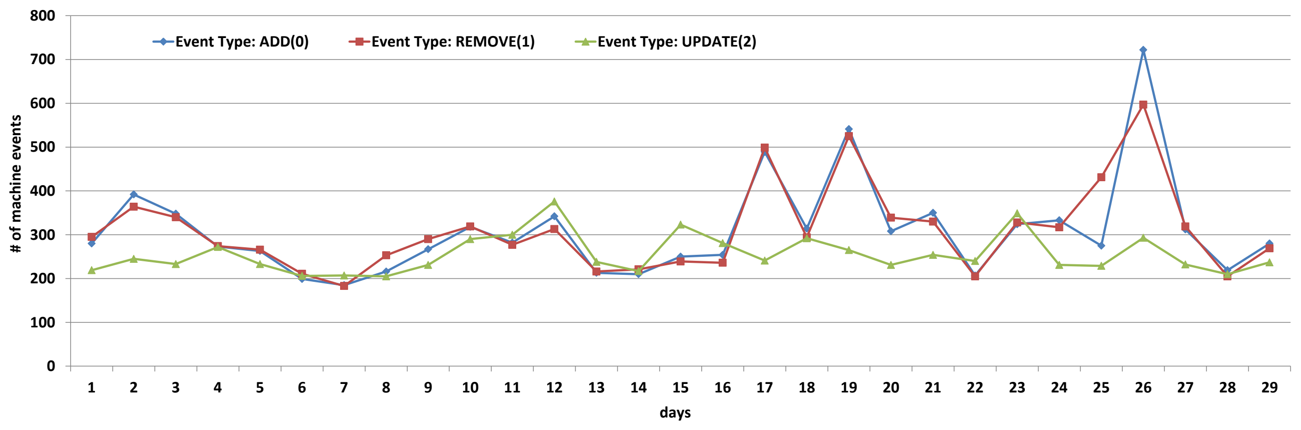

Fig. 2. Number of machine events of different types on each day of the 29-day tracing period.

## B. Daily machine events

Fig. 2 plots the frequency of daily machine events. There are over 870 machines events on average each day. We make two observations. First, machines are constantly updated; on average, 2% of all machines report an update everyday. Second, many machines are removed and added; on average, 2.5% of all machines are removed (and added back soon) each day. A machine could be removed due to system upgrades (e.g., automated kernel patching) and failures (e.g., network down) [4]. The same machine would be added back to the cluster and made available to services once it is upgraded (and rebooted) or the cause for failure has been resolved.

Fig. 3 shows the distribution of the *machine downtime* (time to machine addition after removal; we identified 8,860 such instances). While most downtimes are short, the tail is fairly long (the longest captured in the trace is 17,406 minutes). The plot combines all samples that are longer than 145 minutes into a single sample point in the graph. We suspect that many machine downtimes, more than 60% that are shorter than 25 minutes, might have been caused by automated system upgrades. Other longer downtimes might have involved human intervention, e.g., replacing a disk drive. In certain cases, staffing situations (e.g., failures during late hours) and availability of parts, may have resulted in even longer downtimes.

It is interesting to find that some machines experience more events than others. Fig. 4 shows that quite a few machines saw two or more REMOVE-ADD events. Some machines went through more than ten removals. Also, a few machines, while only a very small fraction of the machine population, reported a resource status update more than 30 times. The result suggests that there is some temporal and/or spatial *locality* of machine events.

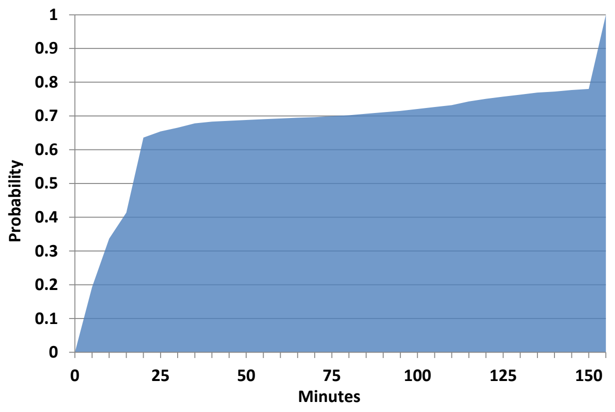

Fig. 3. Machines' downtime as CDF.

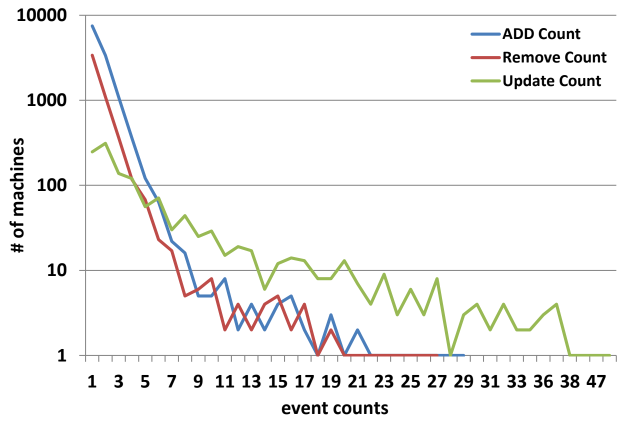

Fig. 4. Machines receiving the same number of events.

## IV. Workload Behavior

### A. Basic job level statistics

As the first step, we pick the four most frequently appearing states—Schedule, Fail, Finish and Kill—and count the number of job events that are associated with those states on each day. Fig. 5 plots the result. We find from the result that: (1) the number of scheduled jobs show a strong periodicity and the period is seven days—just one week. We suspect that fewer jobs are submitted on weekends; (2) the trend in the number of killed jobs and finished jobs follows that of scheduled jobs. That is, the relative probability of jobs being killed or completed (out of all scheduled jobs) is fairly stable; and (3) jobs rarely fail. However, *jobs are frequently killed*—as many

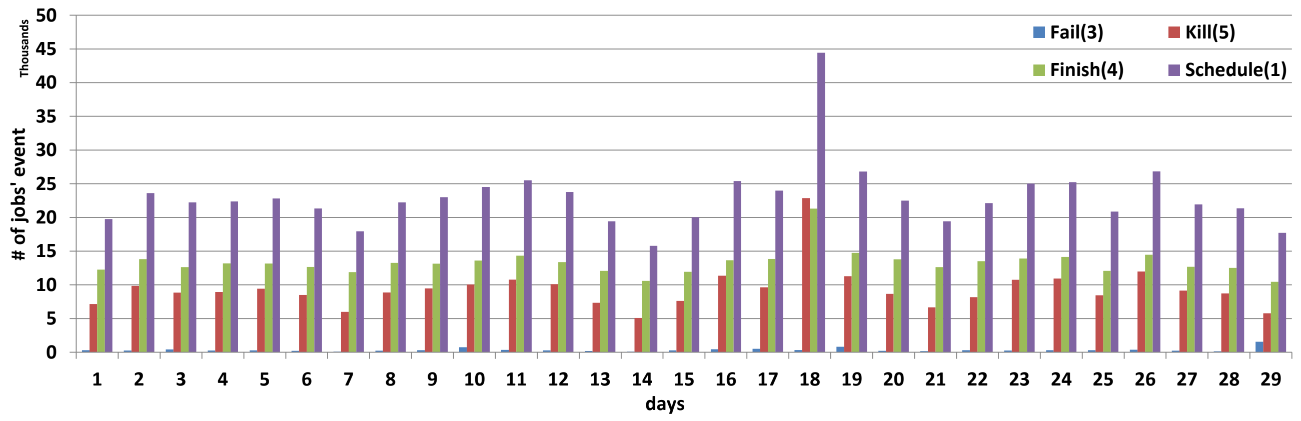

Fig. 5. Time series of the number of different job events (Schedule, Fail, Finish and Kill).

as 40.52% of all scheduled jobs are killed at least once.

In order to gain further insights about job characteristics, in what follows, we will focus only on the finished jobs, not counting killed jobs. Moreover, whenever it is relevant, we will present result split into four *scheduling classes*. Each job in the trace dataset has a scheduling class, an integer value from 0 to 3. A larger value implies that the job is more sensitive to latency. Fig. 6 presents the number of job events according to their event type and scheduling class.

We have several interesting findings from the result. First, scheduled jobs are frequently *killed*—much more frequently than they *failed*. Job kill probability is shown to be fairly high for all scheduling classes. Jobs may be killed by the user (e.g., after serving testing objectives) or by the job scheduler (e.g., when a job's resource requirements can't be met within a reasonable amount of time). Second, over 50% of kill events occurred to jobs in the scheduling class 0 (i.e., latency-insensitive jobs). We suspect that some of these jobs are test jobs submitted by Google engineers. Lastly, latency-insensitive jobs saw (relatively) few failures compared to jobs that are sensitive. This is because these jobs are relatively short (e.g., killed early) and their resource usages are light.

Next, we will examine the size of jobs using two measures, the number of tasks per job and the execution time. Fig. 7 plots the result of the first measure and shows that: (1) Many jobs have a small number of tasks (less than 100)—in fact, a very large number of jobs have a single task; and (2) The tail is quite long—a few jobs have over 2,000 tasks. What is shown implies in our result that *the overall system throughput is determined by the jobs with a few tasks rather than a few jobs with many tasks*. Our trace dataset does not disclose the nature of each job that can directly explain the number of tasks. Still, we expect that jobs having a relatively small number of tasks are easier to schedule.

Fig. 8 further presents the job size distribution based on the execution time. Jobs with a scheduling class from 0 to 2 showed similar behavior—the majority of the jobs run for a fairly short amount of time of less than 15 minutes, while the tail of the distribution is long—there are a number of jobs that run longer than 300 minutes. There were a limited number

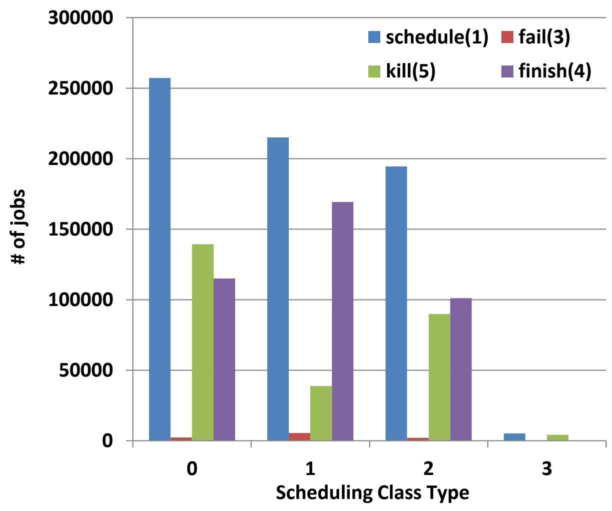

Fig. 6. Job level event counts per scheduling class.

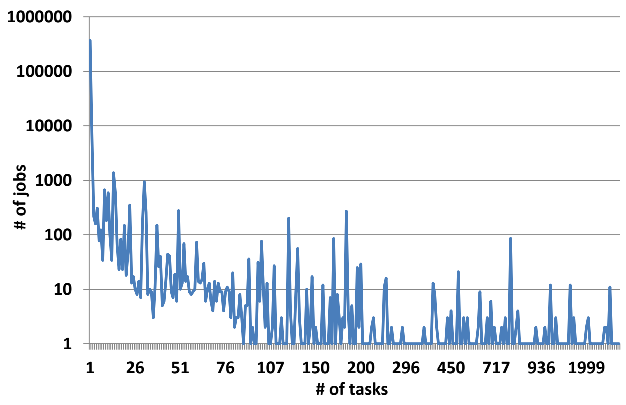

Fig. 7. Job size distribution in terms of tasks per job.

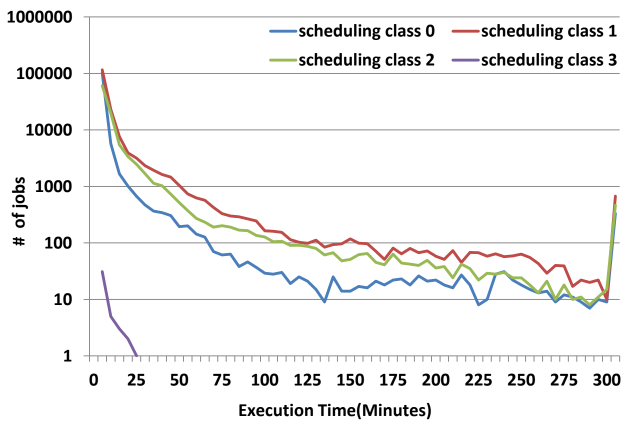

Fig. 8. Job size distribution in terms of execution time.

of, only a few tens of scheduling class 3 jobs (i.e., latency-sensitive jobs) in the dataset and their execution times were all shorter than 30 minutes.

## B. Scheduling constraints

An important factor that affects the operation of a task scheduler is the *scheduling constraints* that have been attached to jobs and tasks. Each constraint defines a condition to meet by the scheduler; for example, a task constraint could limit scheduling of the task to a machine with a smaller memory capacity than specified.

The task constraints dataset lists in total 28,485,619 constraints, 55,272 jobs and 1,405,572 tasks. On average, a job has 20 constraints attached to its tasks. Each constraint has a comparison operator like EQUAL, NOT EQUAL, LESS THAN and GREATER THAN. LESS THAN and GREATER THAN represent a machine attribute in an integer value (e.g., "1,024" MB memory). EQUAL and NOT EQUAL treat a machine attribute as a string (e.g., kernel version number). Table III presents the number of constraints according to their comparison operator.

The result shows that the EQUAL operator dominate the constraints. This is because many jobs and tasks require specific constraints—for instance, specific machine architecture, core count, Ethernet speed and platform. LESS THAN does not appear frequently at all, which is natural because a task would normally require a machine that exceeds a threshold for a particular attribute (e.g., core count, memory capacity). Examples of more concrete task scheduling constrains and their effects on scheduling were reported recently by Sharma et al. [6].

Fig. 9 plots the number of jobs according to their number of constraints in a cumulative form. It is shown that more than 93% of jobs have four or fewer constraints and more than 98% of jobs have less than 20 constraints. While not many, there are jobs having a relatively large number of—over 400 constraints. This indicates that the scheduling algorithm used in the Google cluster must be robust and scalable in handling jobs having a varied number of constraints.

TABLE III
JOB-LEVEL CONSTRAINTS DISTRIBUTION.

| Constraint Op. | EQUAL | NOT EQUAL | LESS T. | GREATER T. |
|---|---|---|---|---|
| Count (%) | 91.35 | 3.98 | 0.05 | 4.62 |

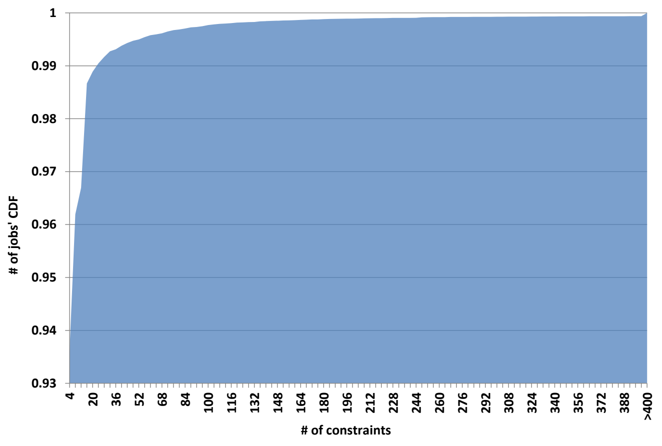

Fig. 9. Per-job constraints distribution (cumulative).

## C. Temporal task level resource usage

Finally, this section will focus on analyzing the CPU usage of tasks based on the Task Usage table. We are particularly interested in looking into how many CPU cycles are spent toward tasks that complete normally ("useful cycles"), tasks that fail ("lost cycles") and tasks that are killed (potentially "wasted cycles"). Because the dataset is quite large, we selected a day's worth of data for presentation out of the 29-day span, which corresponds to 288 5-minute measurement samples.

We used the following method to calculate the CPU cycles. For each 5-minute measurement period $i$ and a task $j$, the *resource usage* (RS) is defined as: $RS_i = T_j.end\_time - T_j.start\_time) \times T_j.cpu\_rate$. Since there is no *event_type* information in each entry in the Task Usage table, we link the information in the Task Usage and the Task Event tables to derive the information. Algorithm 1 lists the actions of this process. In the algorithm, $mid$ is machine id, $jid$ is job id, $tidx$ is task index, $st$ is start time, $et$ is end time, $cpu$ is CPU rate, $type$ is event type, and $tstamp$ is time stamp.

**Algorithm 1** Locate event types for Task Usage entries

```
// Given a 5-min. period i, $MP_i$ and a task $T_j$ from Task Usage falls in $MP_i$
1. Select $T_j.mid$, $T_j.jid$, $T_j.tidx$,$T_j.st$, $T_j.et$, $T_j.cpu$
2. Find events $E_k$s from Task Event who match $T_j.mid$, $T_j.jid$, $T_j.tidx$ where $E_k.type \in \{Kill, Fail, Finish\}$
3. Find $t = arg\min_k(E_k.tstamp - T_j.et)$
4. Set $T_j.type = E_t.type$
```

The result is presented in Fig. 10, which splits the CPU cycles into three categories: cycles spent for tasks that were killed, task that finished normally and tasks that failed. Note

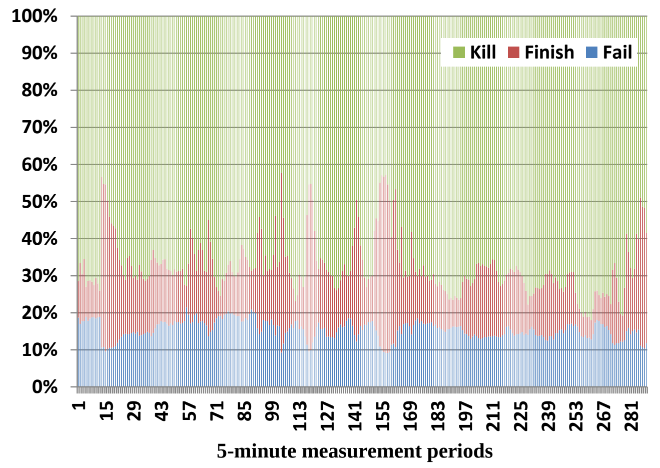

Fig. 10. Proportions of temporal CPU usage measured every 5 minutes.

that the result reports the relative portions of the CPU cycles spent on the tasks and does not consider the CPU cycles that are *idle* and not spent (i.e., CPU cycles of a machine that is idle).

The result shows that about 60% of CPU cycles are used for tasks that were killed. Because tasks are mostly killed by the job owners (e.g., after answering certain test questions or after finding a bug), these CPU cycles are potentially wasted. About only 10% to 15% of the CPU cycles are used towards tasks that complete normally; we consider this portion to be fairly small. It is also noteworthy that a non-trivial portion of the CPU cycles, between 10% and 20%, are spent on tasks that eventually fail.

Our result implies that there are opportunities to improve the scheduling process in the presence of constrains, and in general, *hints*. For example, if the scheduler is offered hints about the nature ("this job is submitted for testing purposes") and periodicity of the submitted jobs (e.g., "can we reserve the resources predictably?"), it may specialize resource management decisions in order to save energy and reduce performance variations of important jobs.

## V. Prior Related Work

The problem of server workload characterization and performance prediction has been widely studied in the past decades [7]–[11]. However, the focus of these studies was on single server workload analysis. More recently, research has been done toward characterizing the workload in a cloud computing environment. For example, Mishra et al. [12] describe an approach to workload classification and its application to the Google Cloud Backend. Chen et al. [13] demonstrate a systematic approach to reasoning about cloud performance tradeoffs using a tool called Statistical Workload Analysis and Replay for MapReduce (SWARM). Khan et al. [14] treat server workload data samples as multiple time series and develop a co-clustering technique to identify correlated workload patterns and introduce Hidden Markov Model (HMM) to characterize the temporal correlations in server clusters. However, these studies concern with single level analysis: job-level analysis, task-level analysis, or server (machine)-level analysis. In this paper, we make a comprehensive statistical analysis on all granularities and reveal some relations between those different level analyses.

There are studies that derive statistical results from realistic, large-scale measurement data. Kavulya et al. [15] study traces collected from a Yahoo! M45 production cluster but the cluster has been available to only select universities. They provide a description of the statistical properties of their trace data in understanding the performance and failure characteristics of Hadoop jobs and present a simple analytical model with configurable Hadoop-specific parameters to predict job-completion times. Chen et al. [16] performed k-means clustering to identify common groups of jobs and did correlation analysis between job semantics and job behaviors. They used a Google cluster trace dataset of early 2010, which covers a relatively short time period of only seven hours (c.f., 29 days of the new Google cluster trace dataset we used in this study). Compared to these studies, our work reveals new interesting results, especially with regard to machine population and daily maintenance events, job and task level behavior of the cluster, and its CPU cycle usage.

## VI. Concluding Remarks

In this work we studied a recent Google cluster usage trace dataset. This trace dataset is considered a valuable addition to the community and discloses much information about how (some) Google's clusters are operated. Our initial investigation reveals several interesting findings. First, machines are continuously taken off-line and on-line to combat failures and to apply upgrades. Updates and upgrades will inevitably increase the heterogeneity of the cluster. However, we find that the machines in the studied cluster are fairly homogeneous—93% of all machines have the same CPU capacity and 86% of all machines have either of the only two distinct memory capacities. On the other hand, about 6% of all machines are considered "newest"—having the largest CPU and memory capacity. Second, we find that there are many jobs submitted each day that are not latency sensitive. Many of these jobs could be codes that are run for testing and debugging purposes. Indeed, we find that there are more jobs that are "killed" in the cluster than jobs that complete normally. We also find that many jobs have relatively few tasks of 100 or less. Lastly, we find that many CPU cycles are put into jobs and tasks that will be eventually killed or will fail. This implies that if the scheduler is offered hints about the nature and periodicity of the submitted jobs, it may specialize the resource management decisions in order to save energy and reduce performance variations of important jobs.

The new Google cluster usage trace dataset is large and has rich information. This work analyzed only a fraction of its information to offer first-order insights from the dataset. An interesting future work may look closely into the relation that exists among the dataset, such as scheduling constraints and scheduling efficiency and delay, job scheduling class and scheduling delay, user id and job characteristics, resource requests and actual resource usage, and resource usage and failure probability.

## Acknowledgment

This work was supported in part by NSF grants CCF-1064976, CCF-1059283 and CNS-1012070.

## References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.

[2] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2009.

[3] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer*. Morgan & Claypool, 2009.

[4] Traces of Google workloads, http://code.google.com/p/googleclusterdata/. [Online]. Available: http://code.google.com/p/googleclusterdata/

[5] C. Reiss, J. Wikes, and J. Hellerstein, "Google cluster-usage traces: format+schema," November 2011.

[6] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038919

[7] M. Calzarossa and G. Serazzi, "Workload characterization: a survey," in *Proceedings of the IEEE*, vol. 81, no. 8, 1993, pp. 1136 – 1150.

[8] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *IEEE International Workshop on Workload Characterization*, 2001, pp. 140 – 148.

[9] P. Barford and M. Crovella, "Generating representative web workloads for network an server performance evaluation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1, 1998, pp. 151–160.

[10] H. P. Artis, "Capacity planning for mvs computer system," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 8, no. 4, 1979, pp. 45–62.

[11] A. B. Downey and D. G. Feitelson, "The elusive goal of workload characterization," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 4, 1999.

[12] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, 2010.

[13] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Towards understanding cloud performance tradeoffs using statistical workload analysis and replay," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-81, May 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-81.html

[14] S. T. Arijit Khan, Xifeng Yan and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Cloudman*, 2012.

[15] R. G. Soila Kavulya, Jiaqi Tan and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," Carnegie Mellon University, Tech. Rep., 2009.

[16] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95, Jun 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.html