Analysis and Lessons from a Publicly Available Google Cluster Trace

Yanpei Chen
Archana Sulochana Ganapathi
Rean Griffith
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-95
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.html

June 14, 2010

Analysis and Lessons from a Publicly Available Google Cluster Trace

Yanpei Chen, Archana Ganapathi, Rean Griffith, Randy H. Katz
RAD Lab, EECS Dept., UC Berkeley

Abstract

System designers in industry are often overwhelmed by large scale data, while researchers in academic often confront a lack of publicly available production data. In this paper, we analyze a large scale production workload trace recently made publicly available by Google. We offer a statistical profile of the data, with several interesting discoveries regarding job arrival patterns, CPU and memory consumptions, task durations, and others. We further perform k-means clustering to identify common groups of jobs, with several methodological departures and different findings compared with prior work on similar data. We also do correlation analysis between job semantics and job behavior, leading to helpful perspectives on capacity planning and system tuning. Our key finding is that while the limited dataset size prevents us from generalizing the trace behaviors observed, the analytical methods we describe nonetheless allow us to extract many system design insights.

1 Introduction

The computing industry has witnessed an explosion of system log data. Monitoring tools are improving and storage costs are decreasing. The sheer volume of data is often overwhelming, ranging from gigabytes to petabytes per day. System designers often describe analyzing such data as the "needle in a haystack" problem. There is a dire need for scalable data mining techniques that extract insights without "adding more hay".

In contrast, researchers in academia face a severe shortage of data. There have been few repositories of *publicly accessible* data. Notwithstanding anonymization tools, organizations hesitate to share more data due to the risk of reverse engineering trade secrets (e.g., datacenter scale and location) and the leakage of private user data. The public repositories available, such as the Computer Failure Data Repository [10], typically contain limited information. Efforts at developing innovative and scalable data mining algorithms are rarely accompanied by evaluation on realistic, large scale datasets. The alternatives, synthetic data or research workloads, are neither as rich nor as diverse as industry datasets [6].

Many researchers have successfully accessed industry datasets through non-disclosure agreements or employment relationships. Such collaboration models often place many restrictions on publication. Thus, there are few opportunities for data comparisons across organizations. Without such comparisons, we would not know whether each project is solving "one of a kind" problems or whether there are general system design insights to be extracted. Additionally, researchers cannot amortize the high overhead of building good system monitoring and data processing tools across multiple projects.

Recently, Google released limited system traces from one of their production clusters. The data is highly anonymized. We show in this paper that the traces nonetheless reveal many insights about the particular workload represented. We describe our tools and share our insights from a statistical analysis of this Google cluster trace. We discuss what can and cannot be done with the quantity and quality of data released. Our primary contributions are:

- We report on publicly available traces and validate their usefulness.
- We give a statistical profile of the trace showing both behavior across time and aggregate distributions (Section 4.1).
- We identify groupings of common jobs using k-means clustering analysis and compare our method and results with that in prior work (Section 4.2).
- We perform correlation analysis between job semantics and job behavior to enhance capacity planning and system tuning (Section 4.3).
- We discuss the system design implications from our findings (Section 5).

In the remainder of this paper, we begin with a re-

view of related work (Section 2) and an overview of the Google cluster data (Section 3). This is followed by our statistical analysis (Section 4) and discussion of design and methodological implications (Section 5). Our key finding is that while this limited dataset prevents us from generalizing the trace behaviors observed, the analytical methods we describe enable us to extract many system design insights regardless of dataset size.

2 Related Work

There is much prior work on characterizing trace data to derive design and operational insights for systems. Notably, there is a strong history of file system trace analysis to understand storage subsystem design implications [1, 6], database workload analysis to inform query performance optimization [12, 7, 3], and more recently, web server workload characterization to monitor trends and evolving access patterns [11].

There have been concerted efforts to make data publicly available to researchers to amortize the cost of data collection. The Computer Failure Data Repository [10] is an example repository of system failure data made available for analysis. The statistical machine learning community has also created several interesting public data repositories. Several companies have individually released traces, such as web server logs collected during the world cup in 1998 [5]. In all these cases, the data has already been cleansed of potential sensitive information, removing the need for per-user legal agreements.

Recently, Google followed suit by releasing anonymized log data from one of their clusters [4]. Our work is a successor of [9], in which authors from Google analyzed a similar dataset. We briefly outline the prior work there in Section 3, with a detailed comparison between the methods and results in Section 4.2.

This paper is also a direct successor of [2], in which we analyze MapReduce traces from two Internet services to build a workload replay mechanism. In contrast to our prior work, the Google dataset spans a much shorter timespan and contains highly anonymized data, as we will describe below.

3 Google Data Overview

3.1 Data format and content

The trace data we use comes from [4]. The dataset is a trace of production workloads running on Google clusters collected over 6 hours and 15 minutes. The workload consists of a set of *tasks*, where each task belongs to a single *job*. A job may have multiple tasks.

Each row in the dataset represents the execution of a single task during a five minute interval and contains the following features. For details see [4]:

- *Time* - time in seconds (int)
- *JobID* - unique job identifier
- *TaskID* - unique task identifier
- *JobType* - job type, an integer in [0, 1, 2, 3], a representation of job semantics
- *Normalized Task Cores* - Normalized number of CPU cores used
- *Normalized Task Memory* - Normalized value of the average memory consumed by the task

The dataset was anonymized in a number of ways. Task and job names have been replaced with numeric identifiers. All timestamps are relative to an unknown reference point, and timing information is reported in five-minute intervals. CPU core and memory consumption is normalized using an unknown a linear transformation. Additionally, there is no information on the semantics (i.e. the meaning) of the different job types.

The trace contains 75 five-minute reporting intervals. There are a total of 3,535,029 observations, 9,218 unique jobs and 176,580 unique tasks.

3.2 Data quality

We made several unexpected discoveries in analyzing the data. Some tasks have different job type markings during different time intervals, violating our mental model that there is a unique association between each tasks, the associated job, and the job type. We resolve this ambiguity by assigning to each task the last seen job type.

Also, some tasks have time gaps in their reports, i.e., some of the 5-minute intervals are missing. This violates our mental model that the system keeps trace of tasks continuously, even if they are temporarily idle. We simply ignore the missing time gaps when computing durations, averages, and other statistics.

We expected to see normalized disk and network utilization data to accompany the normalized CPU core and memory data. However, the trace does not contain such data. In [9], the authors analyzed a similar dataset, and noted that although the disk and network logs are available, they are not analyzed since CPU and memory are the bottleneck resources.

There are also two unexpected kinds of tasks in the dataset. Some tasks have 0 cores and >0 memory. We interpret these tasks as inactive idle tasks that need to be kept in memory. Some other tasks have 0 cores and 0 memory. We interpret these as tasks that consume no resources, but their task pointers are kept in the system for quick responses to infrequently requested services.

3.3 Prior analysis of similar datasets

3.3.1 Google datsets

We are not aware of any other published analysis of the particular dataset we discuss in this paper. However, in [9], the authors analyze a similar but not publicly available dataset collected within Google. The dataset there covers a much longer period of 20 cell (cluster)-days. The authors identified a set of resource-consumption "shapes" for tasks that can be used for cluster scheduling and capacity planning.

The authors in [9] seek to map observed data to task "shapes" by quantizing data for task duration, average CPU, and average memory into small, medium, and large levels. The authors then applied the k-means clustering with the initial clusters subject to the small, medium, and large level constraints. These initial clusters are then manually merged to form the final task signature shapes. The number of final clusters is set to 8, with the cluster merging process driven by lowering the within-cluster coefficient of variation. We later describe our own clustering methodology, with several points of departure from prior work. A detailed comparison is in Section 4.2.

3.3.2 Other datsets

Our predecessor work looked production system logs of Hadoop MapReduce at Facebook and another large Internet company [2]. MapReduce was initially developed at Google, and also has multiple tasks per job. Those Hadoop MapReduce logs cover a time period of up to several months, contain job name strings, which we cannot publish, and MapReduce-specific features such as map and reduce task information, input/shuffle/output data sizes, data locality information, and the like. Such information offers in-depth insight into the workload at the MapReduce application level, and forms the basis for our workload replay mechanism in [2].

Our experience with Hadoop MapReduce suggest that many MapReduce computations are data-intensive, making it very valuable to have information about data sizes and data locality. This is a major departure from the workload in Google, where the bottleneck resources are CPU and memory instead of disk and network. In fact, we found from our Hadoop MapReduce traces that different deployments of the same system have very different workloads. Such differences amplify the need to make public more traces, to enable comparisons across different organizations.

4 Data Analysis

This section presents the results of our statistical analysis. We begin by giving a general statistical profile of the workload. We describe both workload changes at different times during the trace and aggregate statistical distributions over the entire trace. We then perform a clustering analysis to identify common job groups in the workload. The clustering complements the job type assignments originally present in the workload, and helps set design priorities for the system. Lastly, we identify and examine any correlations between the job semantics (i.e. job types) with the job behavior (i.e. clusters). This helps planning for future workloads by connecting system needs with job type projections, which are in turn translated from future business needs. We italicize key observations at the end of the relevant paragraph of text.

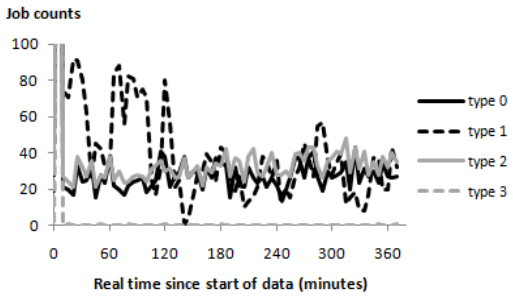

Figure 1: Time series of the number of job launches.

4.1 Workload profile

The first step of our analysis is to statistically characterize the workload. Our description below uses the time series and cumulative distribution functions (CDFs) of several performance dimensions. Depending on the data available, we can generalize this approach to other traces from production clusters.

4.1.1 Temporal behavior

Figure 1 contains a time series of job launches during the workload, where a job launch is defined as the interval during which we first see a task belonging to the job. There is a job launch spike for all jobs at 5 minutes into the traces. We believe this is either a logging artifact, or behavior due to scheduled batch processing or similar effects. Aside from this spike, we see that different job types have vastly different launch patterns. Job Types 0 and 2 both have launch rates fluctuating around 30 jobs per 5 minutes. Job Type 1 shows much larger fluctuations, with steep spikes and dips. Job Type 3 has hardly any new jobs launching, with the line barely visible above the horizontal axis. **Observation 1.** *Different job types have vastly different behaviors, with some jobs having very large spikes in arrival rates.*

Figure 2 contains a time series of the number of running tasks. "Running" is defined as the task being present in the trace. Thus, a running but idle task with 0 normalized cores would also be counted. Again, the shape of

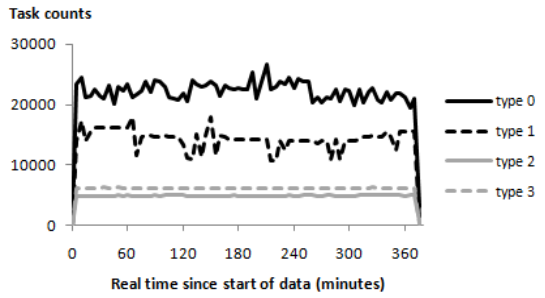

Figure 2: Time series of the number of running tasks.

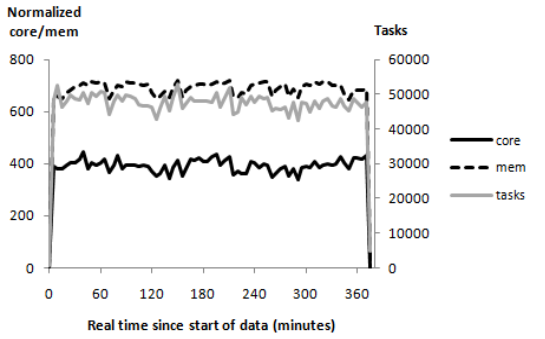

Figure 3: Normalized core and mem time series for all jobs.

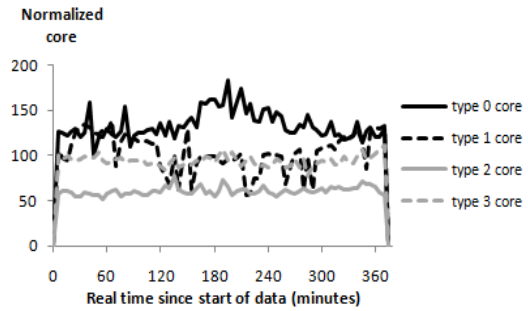

Figure 4: Time series of normalized cores.

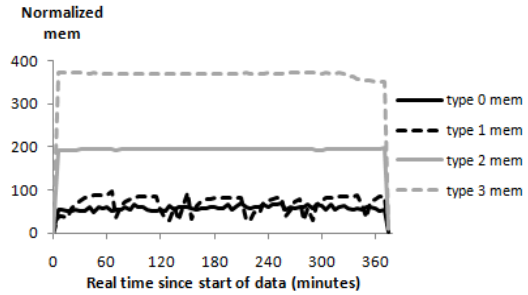

Figure 5: Time series of normalized memory.

the lines are different for different job types, with some subset of job types showing very similar behavior. For example, Job Types 2 and 3 both have near constant number of tasks. Also, for Job Types 0 and 1, the number of running tasks both fluctuate around a constant average.

Figure 3 compares the time series for the normalized cores and memory of all jobs, with the time series of all running tasks. We see that all three lines have highly correlated spikes and dips, suggesting that the average core/memory ratio of tasks does not change at a timescale of several hours.

Figure 4 and 5 respectively shows the normalized core and memory allocation to each job type. The normalized core allocation again shows fluctuations around a steady average. More interesting, the memory allocation to each job type shows barely any fluctuations at all. This is the case for Job Types 0, 2, 3. For Job Type 1, the normalized memory varies along with all other time series examined so far. **Observation 2.** *Each job type has near constant CPU and memory allocations, as well as a near constant number of running tasks.*

4.1.2 Aggregate distributions

Figure 6 summarize the task durations. Clearly, task durations show bimodal behavior. Job Types 0, 1, 2 have mostly short task with a few long tasks. Almost all tasks in Job Type 3 are long. Since our trace last only a few hours, so far we have no way of knowing whether the "long" tasks are tasks that take a long time to complete, or tasks that are suppose to be on-going services and would never complete. **Observation 3.** *Task durations are bimodal, with different job types having different task duration distributions.*

Figure 7 shows the distribution of tasks per job. Again, different job types show different behavior. Additionally, at 1, 2, and 9 tasks per job, all CDFs all show some discrete jumps (i.e., vertical lines). The distribution is also long-tailed. **Observation 4.** *Tasks per job can fall into discrete bins, and has a long-tailed distribution.*

Figures 8, 9, 10, 11, and 12 respectively show the distribution of total and average cores, total and average memory, and memory-core ratio per job. The most striking feature is the large number of vertical lines in the CDFs, corresponding to discrete jumps in the distribution. The exception is Job Type 3, which shows a smooth CDF in all these graphs. **Observation 5.** *Per-job core and memory allocations are highly discrete, although some job types have more continuous allocations.*

4.2 Job clustering

A clustering analysis identifies groups of common jobs in the workload. Knowing this helps set design priorities for the system. For example, if most jobs fall into only a small number of groups, then the system can be tuned specifically to the characteristics of those job groups.

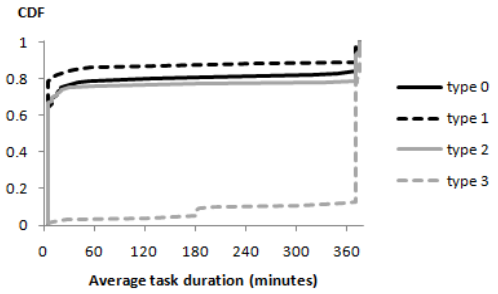

Figure 6: CDF of tasks durations.

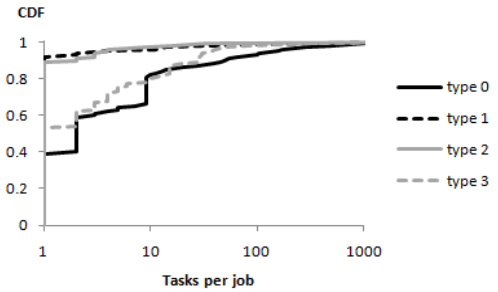

Figure 7: CDF of tasks per job.

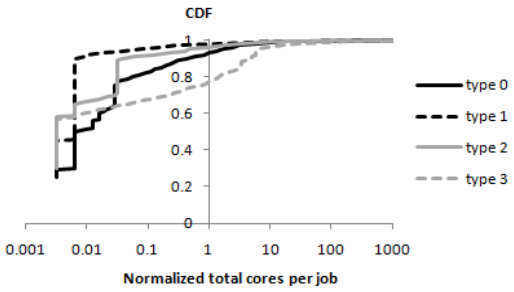

Figure 8: CDF of total normalized cores per job.

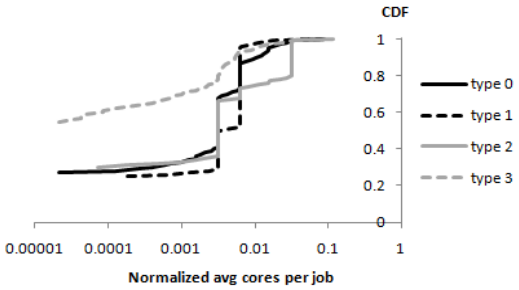

Figure 9: CDF of average normalized cores per job.

4.2.1 Clustering methodology

We use the k-means clustering, a well-known algorithm in statistical machine learning. K-means assigns inputs of n data points into k clusters, with k set by the user. The algorithm assigns each data point to the cluster with the closest mean. The algorithm randomly selects the initial cluster means, then iteratively improves them where the centroid a cluster's data points becomes the new cluster mean. The iteration continues until convergence.

The input data is vectors of job characteristics, one for each job. We use many job characteristics, including all characteristics in Section 4.1, and additional characteristics derived from the primary characteristics. Even if some of the characteristics are redundant, by including them all we ensure that we cover many characteristics that potentially separate clusters. The characteristics are total and average cores for the job, total and average memory, total and average memory while the job is active (i.e., greater than 0 cores), job duration, tasks, task counts multiplied by task duration, active task counts multiplied by active task duration, as well as average, standard deviation, and normalized standard deviation of the memory-core ratio. We normalize all values by linear scaling, such that the maximum value in each characteristic scales to 1. This ensures comparability between data characteristics that have different units.

We use the Matlab implementation of k-means. The initial cluster means are randomly selected. Also, we set k-means to use Euclidean distance. Further, when a cluster becomes empty during the iterative optimization process, we create a new cluster consisting of a single data point that is the furthest from the old cluster mean. Additionally, since different initial cluster centers may lead to different results, we repeat the clustering algorithm 2^{14} times. This ensures that for our 14-dimensional input data, each initial cluster mean will have probabilistically a high and low initial value for every dimension. Information about other k-means parameters are at [8]. We measure the clustering quality by the average distance from each data point to its assigned cluster mean. The `kmeans` function in Matlab returns the sum point-to-mean distance, from which we obtain the average by dividing by the number of data points.

Another subtlety is to select the appropriate value for k, i.e., the appropriate number of clusters. If k is too small or too large, then some clusters may escape detection or we may get artificial cluster boundaries. In particular, when $k = n$, each cluster is a single data point, and the average distance to cluster mean would be zero. Our approach is to increment k, and track both the average point-to-mean distances and the clustering assignment structure. If we are finding "natural" clusters, the average point-to-mean distances would decrease rapidly, and only one cluster at a time would be split. If we are finding artificial cluster boundaries, we would have data points re-arranging between many different clusters in-

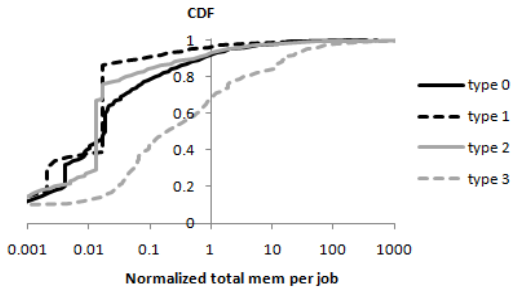

Figure 10: CDF of total normalized memory per job.

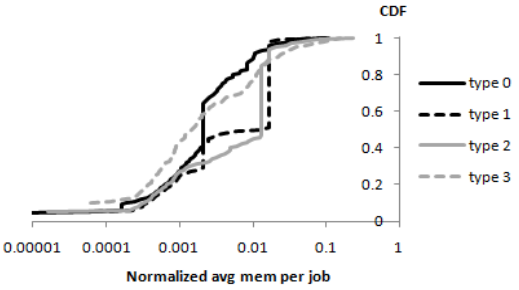

Figure 11: CDF of average normalized memory per job.

stead of a cluster split. Thus, we stop increasing k when we see slow decrease in average point-to-mean distances and cluster re-arrangements.

4.2.2 Clustering results

Figure 13 shows the clustering results for incremental values of k. Each node in the tree represents a cluster. Each arrow represents a large number of data points moving from the source cluster to the destination cluster. Thus, the tree structure indicates that for incremental values of k, clusters divide one at a time until we reach $k = 9$. The text columns on the left and right margins indicate the value of k, the average point-to-mean distance normalized by the number of data dimensions (i.e., D), and the percentage decrease in D for incremental k. We label the clusters by manual inspection of the cluster means. The labels for CPU (more cores) and mem (more memory) are relative. For example, going from $k = 4$ to $k = 5$, we have cluster of memory-heavy tasks split into clusters of very memory-heavy tasks, and memory-heavy tasks that are relatively higher in CPU.

The fact that clusters divide one at a time give us confidence that clear cluster boundaries are being found. If the data did not have such boundaries, e.g., grouped in a circle of constant density, then as we increment k, there would be significant rearrangement between neighboring clusters. Such rearrangement occurs going from $k = 8$ to $k = 9$, indicating that $k = 8$ is a good stopping point.

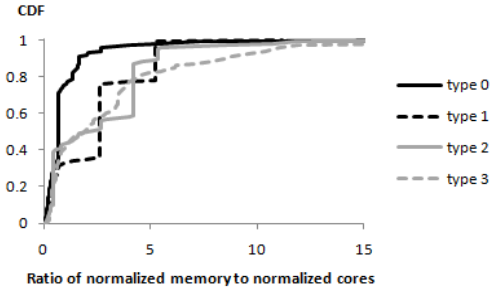

Figure 12: CDF of normalized memory to normalized core ratio.

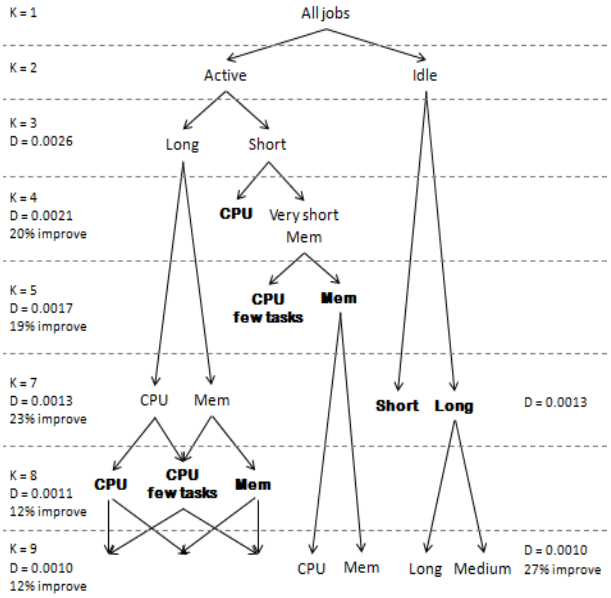

Figure 13: K-means clustering structure.

Also, at $k = 8$, the percentage decrease in D drops from ≈20% to ≈10% per additional cluster, also indicating a significant change in the clustering structure. Thus, we stop at $k = 8$, with the final clusters highlighted in bold.

Figure 14 gives the job count in each final cluster, and Figure 15 gives the numerical value of the final cluster means, along the four dimensions necessary for cluster labeling. The labels follow a hierarchical structure, with the CPU and memory labels being relative to branches at the same clustering hierarchy level. Thus, clusters labeled as "CPU" may in fact have higher memory/core ratio than clusters labeled as "Mem".

Stopping at $k = 8$ also enables easy comparison with the clustering analysis in [9], which also finds 8 clusters. The methodology in [9] starts with 18 initial clusters. The cluster means are constrained to be small/large duration, small/medium/large CPU, and small/medium/large memory. K-means finds the optimum cluster means

Cluster description	Active Long CPU	Active Long CPU Few tasks	Active Long Mem	Active Short CPU	Active Very short CPU Few tasks	Active Very short Mem	Inactive Short	Inactive Long	Total
Total	184	576	44	354	777	4818	1375	1090	9218
Type 0	37	93	6	212	72	1260	318	293	2291
Type 1	36	104	0	67	12	2280	531	203	3233
Type 2	90	237	28	75	692	1276	519	398	3315
Type 3	21	142	10	0	1	2	7	196	375

Figure 14: Number of jobs in each cluster.

Cluster description	Active Long CPU	Active Long CPU Few tasks	Active Long Mem	Active Short CPU	Active Very short CPU Few tasks	Active Very short Mem	Inactive Short	Inactive Long
Active task count × duration	0.03176	0.00247	0.04950	0.00423	0.00001	0.00005	0	0
Duration	0.95974	0.96592	0.97458	0.06958	0.01690	0.01727	0.03483	0.97473
Mem/core ratio	0.06671	0.04229	0.27867	0.01017	0.00821	0.04892	-	-
Tasks count	0.03876	0.00766	0.02799	0.04239	0.00049	0.00116	0.00087	0.00441

Figure 15: Numerical values of cluster means.

subject to the constraints, with the small/medium/large quantization boundaries being manually chosen. The 18 clusters are manually merged into 8 clusters, with the cluster count 8 being manually chosen also.

In contrast, except for cluster labeling, our methodology is fully automated. We increment k and justify stopping at $k = 8$ by observing that the rate of decrease in average points-to-mean distance slows considerably, and that cluster re-arrangements take place. We place no constraints on the cluster centroids, and build confidence in the clusters' quality by noting that each increment in k leads to only one cluster splitting at a time. The cluster tree structure also enable us to identify what the final clusters should be for any $k < 8$.

Another difference is the metric for clustering quality. The analysis in [9] measured cluster quality by within-cluster coefficient of variation (CV), the standard deviation of the data in each cluster normalized by the mean. This is a per-cluster metric, essential for choosing which clusters to merge into "better" clusters. Each data characteristic would have a separate CV, requiring unwieldy multi-dimensional comparisons when the number of data characteristics is large. In contrast, we use the all-clusters average sum of points-to-mean distances. This is a all-clusters metric, essential for identifying a good stopping point for incrementing k. Further, the average sum of points-to-mean distances allows the clustering quality at each k to be represented by a single number. As evident in Figure 13, the single number summary allow the stopping point for k to be easily determined.

A final, major difference is that our input data uses 14 job characteristics, which reduces to four characteristics necessary for the final cluster labels - activity level, duration, CPU-memory ratio, and number of tasks in the job. By having 14 initial characteristics and discovering that we only need 4, we build confidence that the other characteristics are truly redundant. In particular, [9] used average cores and average memory as task descriptors, two characteristics among our 14 initial characteristics but not within the final 4. For us, the relative memory/CPU intensity reduces to a single memory/core ratio. Thus, a good method to identify the most important job descriptor characteristics would be to start with many characteristics and then perform feature selection. Our feature selection method combines automatic k-means clustering with manual cluster labeling. It would be worthwhile to explore a fully automated algorithm.

Cluster description	Active Long CPU	Active Long CPU Few tasks	Active Long Mem	Active Short CPU	Active Very short CPU Few tasks	Active Very short Mem	Inactive Short	Inactive Long
Type 0	-0.02	-0.06	-0.03	**0.26**	-0.12	0.01	-0.04	0.04
Type 1	-0.05	-0.11	-0.06	-0.09	**-0.27**	**0.33**	**0.22**	**-0.22**
Type 2	0.04	0.05	0.05	-0.14	**0.41**	**-0.27**	0.01	-0.01
Type 3	0.10	**0.40**	0.11	-0.04	-0.05	**-0.24**	**-0.32**	**0.32**

Figure 16: Correlation coefficient between different job types and clusters.

4.3 Correlating semantics and behavior

It is desirable to correlate job semantics and behavior. Identifying such correlations is important because job semantics represent application semantics, which changes according to evolving user and business needs. On the other hand, job behavior is a system property, which should be continuously optimized in response to evolving job semantics. Thus, to truly do capacity planning and system tuning, system designers need to translate job semantics into job behavior.

In this Google trace, we treat job types as a proxy for job semantics, since they are assigned prior to trace collection. Job behavior is represented by the final cluster means, which describe the job along several characteristics. Figure 14 gives the number of jobs per cluster per job type. However, this information remains insufficient. When a cluster has many jobs of a type, we need to know whether the job type is common and has many jobs in all clusters, or a strong correlation actually exists. Likewise, we need to know the anti-correlations.

Figure 16 shows the correlation coefficient between the cluster assignments and job types. The correlation coefficient ranges from 1 (strong correlation) to 0 (no correlation) to -1 (strong anti-correlation). The strongest correlations and anti-correlations are highlighted in bold.

We see that Figure 16 is also insufficient for capacity planning. Our intuition is that strong correlations are

associated with many data points. However, correlation measures only the relative number of jobs. For clusters with many jobs, a cluster-job-type pair with strong anti-correlation may still contain many jobs. Likewise, for clusters with a few jobs, a cluster-job-type pair with strong correlation may give only a few jobs. We need both Figure 14 and Figure 16 to understand the distribution of job types across the clusters.

5 Discussion

5.1 System design implications

One can derive many design insights from the statistical characteristics we described. We highlight here some of the most striking implications.

Scheduling algorithms must be able to handle very large spikes in job arrival rates (Observation 1 in Section 4). Also, if scheduling algorithms seek to keep constant the number of tasks, CPU, and memory allocation for each job type (Observation 2), such algorithms would need to handle jobs with pathological number of tasks, or CPU and memory demands. Otherwise, application designs can game the scheduler to horde resources by rewriting their jobs with different tasks, CPU, and memory demands. Even if there is no intentional gaming, having game-resistant schedulers can partially compensate for tasks with specialist task slot, CPU, and memory needs. Additionally, schedulers need to account for highly discrete resource demands (Observations 4 and 5), with particular attention to any boundary effects.

The results of our clustering analysis suggest that the largest clusters are very short active jobs and inactive jobs, while the smallest clusters are the long active jobs (Figure 14). This means that the cluster management system need to keep many inactive jobs in memory. Also, the small clusters of long active jobs indicate that they are the biggest contributors to the overall clustering sum of points-to-mean distances. Thus, it is important to optimize for the resource allocation and placement for these jobs. Any sub-optimal allocation or placement would impact system performance over a long time. The long duration of such jobs may require migrating the task slots to different machines. The large number of short active jobs also make them a prime target for performance optimization. Fortunately, the bimodal nature of job durations (Observation 3) makes it easy to classify the jobs.

For capacity planning, we need to translate projections of application needs to system resource demands. Our correlation analysis indicates that a good translation is difficult. We propose to use the empirical distribution of each job type across the clusters. For example, if we project Job Type 3 to increase by, say, 100%, while the rest of the systems remain fixed, we can do capacity planning by doubling the number of Type 3 jobs in every cluster. The result is many more long duration jobs, since Job Type 3 is highly correlated with long jobs and highly anti-correlated with short jobs (Figure 16). We can then optimize the new system accordingly.

5.2 Limitation of the traces

There are several limitations to the types of analysis enabled by the data currently public data. Although we extracted some insights, there is insufficient information for generalizing our findings to longer time periods or other workloads at Google. In particular, with longer timespans of observations, we could potentially build a per-job progress indicator that predicts, given information about the job's completed tasks, when the job will complete execution and what its resource requirements are.

The high degree of anonymization of the data prevents us from gaining insights on the semantics of the job scheduler. As a result, we are unable to evaluate the pros and cons of alternate scheduling algorithms. This limitation would be addressed with more fine-grained data on task execution and queuing times, and more information about the job type semantics.

A major benefit of having real production traces is that we can inform design decisions based on common jobs and resource requirements. However, to evaluate these design decisions, we still need a realistic and representative workload, which is seldom extractable from the traces. A worthy research goal is to create a system-generic workload synthesis and replay tools that reproduce the original workload's behavior without the overhead of reproducing the system configuration and raw data. Our work in [2] is an initial step in this direction for Hadoop MapReduce workloads.

6 Conclusions and Future Work

In this paper, we demonstrated that publicly available traces are invaluable regardless of dataset size. Clearly, more data would allow us to generalize findings and gain additional system design insights. We hope the Google public data release foreshadows a seachange in attitudes towards making production traces publicly available. We argue for a public production system trace repository similar to the Computer Failure Data Repository. To facilitate such a repository while addressing trade secret and user privacy concerns, future work should develop a toolkit with anonymizers, data format converters, and standard algorithms for detailed statistical analysis. We hope our paper represents the first step in this direction.

References

[1] BAKER, M. G., HARTMAN, J. H., KUPFER, M. D., SHIRRIFF, K. W., AND OUSTERHOUT, J. K. Measurements of a Dis-

tributed File System. In *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1991), ACM, pp. 198–212.

[2] CHEN, Y., GANAPATHI, A. S., GRIFFITH, R., AND KATZ, R. H. Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay. Tech. Rep. UCB/EECS-2010-81, EECS Department, University of California, Berkeley, May 2010.

[3] ELNAFFAR, S., MARTIN, P., AND HORMAN, R. Automatically Classifying Database Workloads. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management* (2002), pp. 622–624.

[4] HELLERSTEIN, J. L. Google Cluster Data. `http://googleresearch.blogspot.com/2010/01/google-cluster-data.html`.

[5] INTERNET TRAFFIC ARCHIVE. FIFA World Cup 1998 access logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

[6] LEUNG, A. W., PASUPATHY, S., GOODSON, G., AND MILLER, E. L. Measurement and Analysis of Large-scale Network File System Workloads. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference* (2008), pp. 213–226.

[7] LO, J. L., BARROSO, L. A., EGGERS, S. J., GHARACHORLOO, K., LEVY, H. M., AND PAREKH, S. S. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture* (1998), pp. 39–50.

[8] MATLAB. `kmeans` documentation. `http://www.mathworks.com/access/helpdesk/help/toolbox/stats/kmeans.html`.

[9] MISHRA, A. K., HELLERSTEIN, J. L., CIRNE, W., AND DAS, C. R. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *SIGMETRICS Perform. Eval. Rev. 37*, 4 (2010), 34–41.

[10] SCHROEDER, B., AND GIBSON, G. The Computer Failure Data Repository (CFDR): collecting, sharing and analyzing failure data. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (New York, NY, USA, 2006), ACM, p. 154.

[11] WILLIAMS, A., ARLITT, M., WILLIAMSON, C., AND BARKER, K. Chapter 1 Web Workload Characterization: Ten Years Later.

[12] YU, P. S., CHEN, M.-S., HEISS, H.-U., AND LEE, S. On Workload Characterization of Relational Database Environments. *IEEE Trans. Softw. Eng. 18*, 4 (1992), 347–355.