Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters

Asit K. Mishra[†] * Joseph L. Hellerstein[§] Walfredo Cirne[§] Chita R. Das[†]
[†]The Pennsylvania State University
University Park, PA-16801, USA
{amishra,das}@cse.psu.edu
[§]Google Inc.
Mountain View, CA 94043, USA
{jlh,walfredo}@google.com

Abstract

The advent of cloud computing promises highly available, efficient, and flexible computing services for applications such as web search, email, voice over IP, and web search alerts. Our experience at Google is that realizing the promises of cloud computing requires an extremely scalable backend consisting of many large compute clusters that are shared by application tasks with diverse service level requirements for throughput, latency, and jitter. These considerations impact (a) capacity planning to determine which machine resources must grow and by how much and (b) task scheduling to achieve high machine utilization and to meet service level objectives.

Both capacity planning and task scheduling require a good understanding of task resource consumption (e.g., CPU and memory usage). This in turn demands simple and accurate approaches to workload classification—determining how to form groups of tasks (workloads) with similar resource demands. One approach to workload classification is to make each task its own workload. However, this approach scales poorly since tens of thousands of tasks execute daily on Google compute clusters. Another approach to workload classification is to view all tasks as belonging to a single workload. Unfortunately, applying such a coarse-grain workload classification to the diversity of tasks running on Google compute clusters results in large variances in predicted resource consumptions.

This paper describes an approach to workload classification and its application to the Google Cloud Backend, arguably the largest cloud backend on the planet. Our methodology for workload classification consists of: (1) identifying the workload dimensions; (2) constructing task classes using an off-the-shelf algorithm such as k-means; (3) determining the break points for qualitative coordinates within the workload dimensions; and (4) merging adjacent task classes to reduce the number of workloads. We use the foregoing, especially the notion of qualitative coordinates, to glean several insights about the Google Cloud Backend: (a) the duration of task executions is bimodal in that tasks either have a short duration or a long duration; (b) most tasks have short durations; and (c) most resources are consumed by a few tasks with long duration that have large demands for CPU and memory.

1. Introduction

Cloud Computing has the potential to provide highly reliable, efficient, and flexible computing services. Examples of cloud services or applications are web search, email, voice over IP, and web alerts. Our experience at Google is that a key to successful cloud computing is providing an extremely scalable backend consisting of many large compute clusters that are shared by application tasks with diverse service level requirements for throughput, latency, and jitter. This paper describes a methodology for classifying workloads and the application of this methodology to the Google Cloud Backend.

Google applications are structured as one or more job that run on the Google Cloud backend consisting of many large compute clusters. Jobs consist of one to thousands of tasks, each of which executes on a single machine in a compute cluster. Tasks have varied service level requirements in terms of throughput, latency, and jitter; and tasks place varied demands on machine resources such as CPU, memory, disk bandwidth, and network capacity. A compute cluster contains thousands of machines, and typically executes tens of thousands of tasks each day.

Our role at Google has been closely connected with scaling the cloud backend, especially capacity planning and task scheduling. Capacity planning determines which machine resources must grow by how much to meet future application demands. Effective capacity planning requires simple and accurate models of the resource demands of Google tasks in order to forecast future resource demands that are used to determine the number and configuration of machines in a compute cluster. Scheduling refers to placing tasks on machines to maximize machine utilizations and to meet service level objectives. This can be viewed as multi-dimensional bin packing in which bin dimensions are determined by machine configurations (e.g., number of cores, memory size). Here too, we need simple and accurate models of the resource demands of Google tasks to construct a small number of "task resource shapes" in order to reduce the complexity of the bin packing problem. Certain aspects of capacity planning and scheduling can benefit from models of task arrival rates. Although such models are not a focus of this paper, work such as [17] seems directly applicable.

In this paper, the Google Cloud Backend workload is a collection of tasks, each of which executes on a single machine in a compute cluster. We use the term workload characterization to refer to models of the machine resources consumed by tasks. The workload models should be simple in that there are few parameters to estimate, and the models should be accurate in that model predictions of task resource consumption have little variability.

A first step in building workload models is **task classification** in which tasks with similar resource consumption are grouped together. Each task class is referred to as a workload. One approach to task classification is to create a separate class for each task. While models based on such a fine-grain classification can be quite accurate for frequently executed tasks with consistent resource demands, the fine-grain approach suffers from complexity if there are a large number of tasks, as is the case in the Google Cloud Backend. An alternative is use a coarse-grain task classification where there is a single task class and hence a single workload model to describe the consumption of each resource. Unfortunately, the diversity of Google tasks means that the predictions based on a coarse-grain approach have large variances.

To balance the competing demands of model simplicity and model accuracy, we employ a medium-grain approach to task classification. Our approach uses well-known techniques from statistical clustering to implement the following methodology: (a) identify the workload dimensions; (b) construct task clusters using an off-the-shelf algorithm such as k-means; (c) determine the break points of qualitative coordinates within the workload dimensions; and (d) merge adjacent task clusters to reduce the number of model parameters. Applying our methodology to several Google compute clusters yields eight workloads. We show that for the same compute cluster on successive days, there is consistency in the characteristics of each workload in terms of the number of tasks and the resources consumed. On the other hand, the medium-grain characterization identifies differences in workload characteristics between clusters for which such differences are expected.

* This work was done while interning at Google during summer 2009.

This paper makes two contributions. The first is a methodology for task classification and its application to characterizing task resource demands in the Google Cloud Backend, arguably the largest cloud backend on the planet. In particular, we make use of qualitative coordinates to do task classification, a technique that provides an intuitive way to make statements about workloads. The second contribution is insight into workloads in the Google Cloud Backend that make use of our qualitative task classifications. Among the insights are: (a) the duration of task executions is bimodal in that tasks either have a short duration or a long duration; (b) most tasks have short durations; and (c) most resources are consumed by a few tasks with long duration that have large demands for CPU and memory.

The rest of the paper is organized as follows. Section 2 uses the coarse-grained task classification described above to characterize resource consumption of the Google Cloud Backend. Section 3 presents our methodology for task classification, and Section 4 applies our methodology to Google compute clusters. Section 5 applies our results to capacity planning and scheduling. Section 6 discusses related work. Our conclusions are presented in Section 7.

2. Coarse-Grain Task Classification

This section characterizes the resource usage of Google tasks using a coarse-grain task classification in which there is a single workload (task class). Our resource characterization model is simple—for each resource, we compute the mean and standard deviation of resource usage. The data we consider are obtained from clusters with stable workloads that differ from one another in terms of their applications mix. The quality of the characterization is assessed based on two considerations: (1) the characterization should show similar resource usage from day to day on the same compute cluster; and (2) the characterization should evidence differences in resource usage between compute clusters.

We begin by describing the data[1] used in our study. The data consist of records collected from five Google production compute clusters over four days. A record reports on a task's execution over a five minute interval. There is an identifier for the task, the containing job for the task, the machine on which the task executed, and the completion status of the task (e.g., still running, completed normally, failure). CPU usage is reported as the average number of cores used by the task over the five minute interval; and memory usage is the average gigabytes used over the five minute interval.

In general, there are many other resources that should be considered, such as disk capacity and network bandwidth. However, for the clusters that we evaluate in our study, CPU and memory usage are the most important characteristics in terms of scheduling. Hence, we only consider these resources in our evaluations. We use these data to define a multi-dimensional representation of task resource usage or **task shape**. The dimensions are time in seconds, CPU usage in cores, and memory usage in gigabytes. When the Google Task Scheduler places a task on a machine, it must ensure that this multi-dimensional shape is compatible with the shape of the other tasks on the machine given the resource capacities of the machine. For example, on a machine with 4GB of main memory, we cannot have two tasks that simultaneously require 3GB of memory.

Although task shape has three dimensions (and many more if other resources are considered), we often use metrics that combine time with a resource dimension. For example, in the sequel we use the metric core-hours. The core-hours consumed by a task is the product of its duration and average core usage (divided by 12 since records are for five minute intervals). We compute GB-hours in a similar way.

Figure 1 shows the task mean core-hours resulting from the coarse-grain workload characterization for the twenty compute-cluster-days in our data. Observe that there is consistency in mean core-hours from day to day on the same compute cluster. However, there is little difference in mean core-hours between compute clusters. Indeed, in almost all compute clusters, tasks consume approximately 1 core-hour of CPU. Similarly, Figure 2 shows the mean GB-Hours for the same data. Here too, we see consistency from day to day on the same compute cluster. However, there is little difference between compute clusters in that mean GB-hours is about 3.0. This observation remains unchanged if we use the median value instead of the mean.

There is a further problem with the coarse-grain classification—it results in excessive variability in mean core-hours and GB-hours that makes it even more difficult to draw inferences about resource usage. We quantify variability in terms of coefficient of variation (CV), the ratio of the standard deviation to the mean (often expressed as a percent). One appeal of CV is that it is unitless. Our guideline is that CV should be much less than 100%. Figure 3 plots CV for mean core-hours, mean GB-hours, and other metrics. We see that for the coarse-grain task classification, CVs are consistently in excess of 100%,

3. Methodology for Constructing Task Classifications

This section presents our methodology for constructing task classifications. The objective is to construct a small number of task classes such that tasks within each class have similar resource usage. Typically, we decompose resource usage into average usage and task duration. We use the term **workload dimensions** to refer to the tuple consisting of task duration and resource usages.

Intuitively, tasks belong to the same workload if they have comparable magnitudes for each of their workload dimensions. We find it useful to express this qualitatively. Typically, the **qualitative coordinates** are *small*, *medium*, and *large*. Qualitative coordinates provide a convenient way to distinguish between workloads. For example, two workloads might both have small CPU and memory usage, but one has small duration and the other has large duration. Ideally, we want to minimize the number of coordinates in each dimension to avoid an explosion in the number of workloads. To see this, suppose there are three workload dimensions and each dimension has three qualitative coordinates. Then, there are potentially 27 workloads. In practice, we want far fewer workloads, usually no more than 8 to 10.

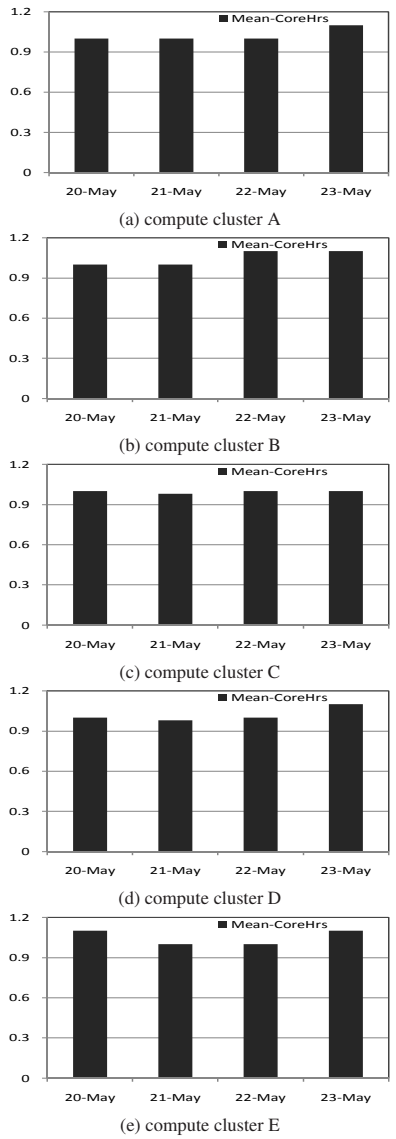
Our methodology for workload characterization must address three challenges. First, tasks within a workload should have very similar resource demands as quantified by their within class CV for each workload dimension. Second, our methodology must provide a way to compute numeric breakpoints that define the boundaries between qualitative coordinates for each workload dimension. Third, we want to minimize the number of workloads.

Figure 4 depicts the steps in our methodology for constructing task classifications. The first step is to identify the workload dimensions. For example, in our analysis of the Google Cloud Backend, the workload dimensions are task duration, average core usage, and average memory usage. In general, the choice of workload dimensions depends on the application of the workload characterization (e.g., what criteria are used for scheduling and how charge-back is done for tasks running on shared infrastructure).

The second step in our methodology constructs preliminary task classes that have fairly homogeneous resource usage. We do this by using the workload dimensions as a feature vector and applying an off-the-shelf clustering algorithm such as k-means. One consideration here is ensuring that the workload dimensions have similar scales to avoid biasing task classes.

The third step in our methodology determines the break points for the qualitative coordinates of the workload dimensions. This step is manual and requires some judgement. We have two considerations. First, break points must be consistent across workloads. For example, the qualitative coordinate *small* for duration must have

[1] [1] is a publicly released Google Cluster data very similar to the one used in this study.



(a) compute cluster A

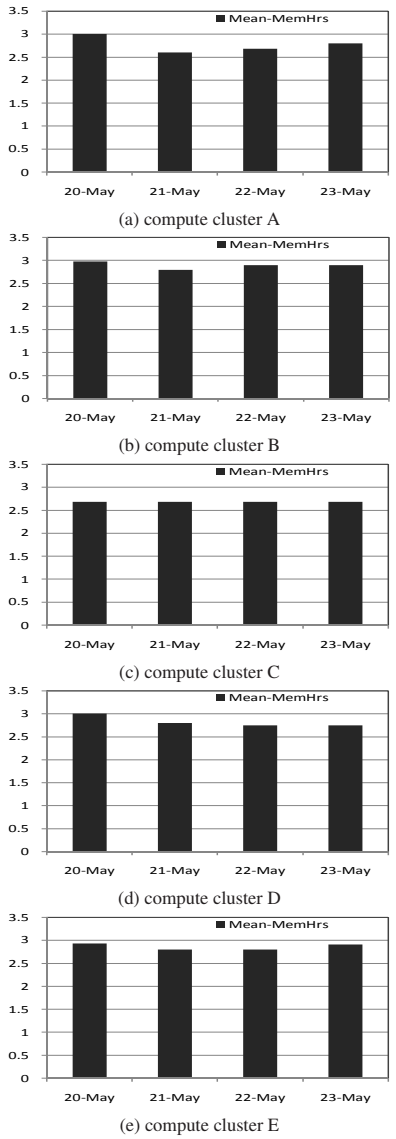
(b) compute cluster B

(c) compute cluster C

(d) compute cluster D

(e) compute cluster E

Figure 1. Task Mean Core-Hours for five compute clusters for four days



(a) compute cluster A

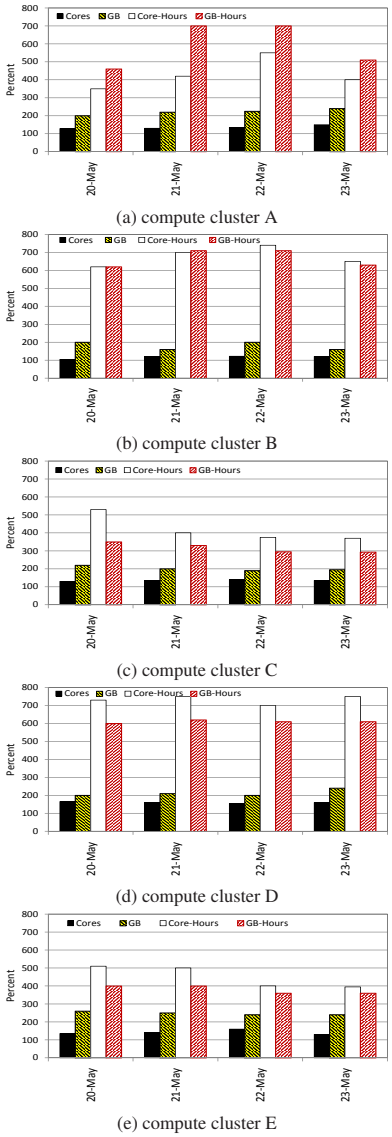
(b) compute cluster B

(c) compute cluster C

(d) compute cluster D

(e) compute cluster E

Figure 2. Task Mean GB-Hours for five compute clusters for four days



(a) compute cluster A

(b) compute cluster B

(c) compute cluster C

(d) compute cluster D

(e) compute cluster E

Figure 3. Coeff. of Variation in Cores, GBs, Core-Hours and GB-Hours of five compute clusters for four days

the same break point (e.g., 2 hours) for all workloads. Second, the result should produce low within-class variability (as quantified by CV) for each resource dimension.

The fourth and final step in our methodology merges classes to form the final set of task classes; these classes define our workloads. This involves combining "adjacent" preliminary task classes. Adjacency is based on the qualitative coordinates of the class. For example, in the Google data, duration has qualitative coordinates *small* and *large*; for cores and memory, the qualitative coordinates are *small*, *medium*, *large*. We use *s* to denote *small*, *m* to denote *medium*, and *l* to denote *large*. Thus, the workload *smm* is adjacent to *sms* and *sml* in the third dimension. Two preliminary classes are merged if the CV of the merged classes does not differ much from the CVs of each of the preliminary classes. Merged classes are denoted by the wild card "*". For example, merging the classes *sms*, *smm* and *sml* yields the class *sm**.

4. Classification and Resource Characterization for Google Tasks

This section applies the methodology in Figure 4 to several Google compute clusters.

4.1 Task Classification

The first step in our methodology for task classification is to identify the workload dimensions. We have data for task usage of CPU, memory, disk, and network. However, in the compute clusters that we study, only CPU and memory are constrained resources. So, our workload dimensions are task duration in seconds, CPU usage in cores, and memory usage in gigabytes.

The second step of our methodology constructs preliminary task classes. Our intent is to use off-the-shelf statistical clustering techniques such as k-means [16]. However, doing so creates a challenge because of differences in scale of the three workload dimensions. Duration ranges from 300 to 86,400 seconds; CPU usage ranges from 0 to 4 cores; and memory usage varies from 0 to 8 gigabytes. These differences in scaling can result in clusters

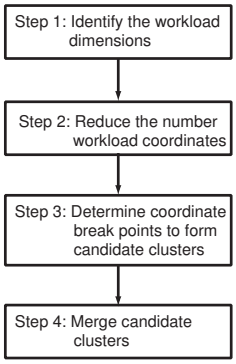


Figure 4. Methodology for constructing task classifications

Preliminary Class	Duration(Hours)	CPU (cores)	Memory (GBs)
1	Small 0.0833	Small 0.08	Small 0.48
2	Small 0.0834	Small 0.19	Medium 0.67
3	Small 0.0956	Small 0.18	Large 1.89
4	Small 0.0888	Medium 0.34	Small 0.38
5	Small 0.4466	Medium 0.47	Medium 0.81
6	Small 0.4166	Medium 0.38	Large 1.04
7	Small 0.4366	Large 1.23	Small 0.44
8	Small 0.8655	Large 0.98	Medium 0.91
9	Small 0.4165	Large 1.39	Large 1.54
10	Large 18.34	Small 0.12	Small 0.48
11	Large 19.34	Small 0.16	Medium 0.85
12	Large 22.23	Small 0.16	Large 1.66
13	Large 22.83	Medium 0.38	Small 0.38
14	Large 19.34	Medium 0.28	Medium 0.77
15	Large 16.89	Medium 0.41	Large 1.76
16	Large 17.57	Large 1.89	Small 0.48
17	Large 22.23	Large 2.34	Medium 0.97
18	Large 20.81	Large 2.22	Large 2.09

Table 1. Clustering results with 18 task classes (numbers represent mean value for each dimension)

that are largely determined by task duration. We address this issue by re-scaling the workload dimensions so that data values have the same range. For our data, we use the range 0 to 4. CPU usage is already in this range. For duration, we subtract 2 from the natural logarithm of the duration value; the result lies between 0 and 4. For memory usage, we divide by 2.

To apply k-means, we must specify the number of preliminary task classes. One heuristic is to consider three qualitative coordinates for each workload dimension. With three workload dimensions, this results in 27 preliminary task classes. This seems excessive. In the course of our analysis of task durations, we observed that that task duration is bimodal; that is, tasks are either short running or very long running. Hence, we only consider the qualitative coordinates *small* and *large* for the duration dimension. Doing so reduces the number of preliminary task classes from 27 to 18.

We applied k-means to the re-scaled data to calculate 18 task classes for each compute cluster. The results were manually adjusted so that the CV within each task class is less than 100%, and no task class has a very small number of points. Table 1 displays the preliminary task classes for one compute cluster.

Step 3 determines the break point for the qualitative coordinates. Our starting point is Table 1. We have annotated the cells of the table with qualitative values. For duration, there are just two qualitative coordinates, as expected by the bimodal distribution of duration. For CPU and memory usage, the qualitative coordinates are as *small, medium and large*. As shown in Table 2, the values of the qualitative coordinates for a workload dimension are chosen so as to cover the range of observed values of the workload dimension.

Step 4 reduces the total number of task classes by merging adjacent classes if the CV of the merged task class is much less than 100%. Table 3 displays the results. For example, final class 3 *slm* in Table 3 is constructed by merging preliminary task classes 7 and 8 in Table 1 along the memory dimension. Similarly, final task class 8 is formed by merging preliminary task classes 15 and 18 along the CPU dimension.

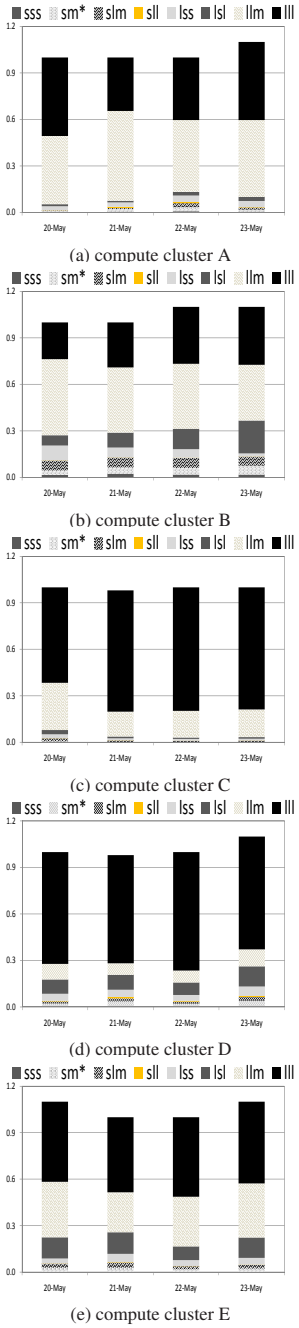


Figure 5. Fraction contribution of 8 task classes to Mean Core-Hours for five compute clusters for four days

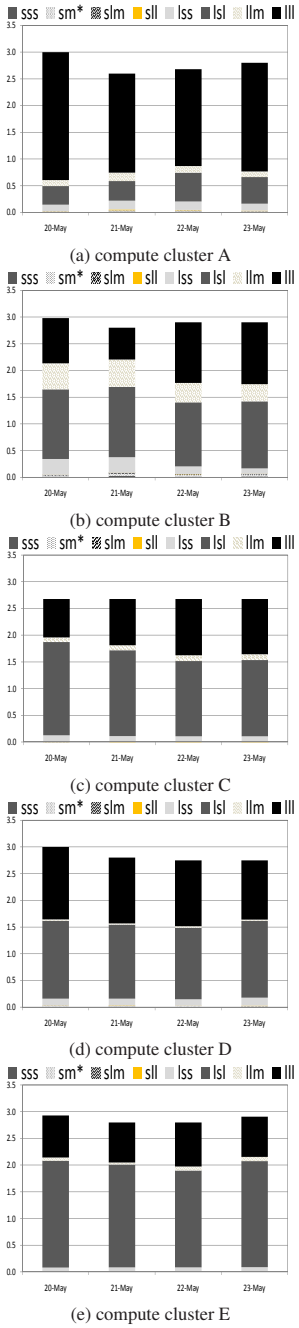


Figure 6. Fraction contribution of 8 clusters to Mean GB-Hours for five compute clusters for four days

4.2 Assessments

Although we construct task classes using data from a single compute cluster and a single day, it turns out that the same task classes provide a good fit for the data for the other 19 compute-cluster-days. Figure 5 displays the mean core-hours for five compute clusters across four days using the workloads defined in Table 3. The

Qualitative Coordinate	Duration (Hours)	CPU (cores)	Memory (GBs)
Small/Small	0 -2	0 - 0.2	0 - 0.5
Medium		0.2 - 0.5	0.5 - 1
Large/Large	2-24	0.5 - 4	> 1

Table 2. Breakpoints for Small, Medium and Large for duration, cpu and memory

Final Class	Duration(Hours)	CPU (cores)	Memory (GBs)
1: sss	Small	Small	Small
2: sm*	Small	Med	**all**
3: slm	Small	Large	Small+Med
4: sll	Small	Large	Large
5: lss	Large	Small	Small
6: lsl	Large	Small	Large
7: llm	Large	Med+Large	Small+Med
8: lll	Large	Med+Large	Large

Table 3. Final task classes (workloads)

height of each bar is the same as in Figure 1, but the bars are shaded to indicate the fraction of mean core-hours that is attributed to each task class.

Our first observation is that the contribution of the task classes to mean core-hours is consistent from day to day for the same compute cluster. For example, in compute cluster A, task class *lll* accounts for approximately 0.3 to 0.4 cores for all A cluster-days. Our second observation is that there are significant differences between compute clusters as to the contribution of task classes. For example, in compute cluster D, task class *lll* consistently accounts for 0.7 cores of mean core-hours. Thus, even though mean core-hours is approximately the same in compute clusters A and D, there are substantial difference in the contributions of the task classes to mean core-hours.

The foregoing observations apply to mean GB-hours as well. Figure 6 displays the contribution of task classes to mean GB-hours for five compute clusters. As with core-hours, we see that the contributions by task class (shaded bars) to mean GB-hours are quite similar from day to day within the same compute cluster. Further, there are dramatic differences between compute clusters in terms of the contribution to mean core-hours and mean GB-hours by task class. For example, task class *lll* consistently accounts for 2.5 core-hours in compute cluster A, but *lll* accounts for less than 1 GB-hour in compute cluster E.

Figure 7 provides a different visualization of the data in Figure 5 and Figure 6. Each plot has as its horizontal axis the 8 final task classes, with bars grouped by day. In this way, we can more readily tell if there is day to day consistency. The columns of the figure are for different metrics. Column 1 is the percentage of tasks by class within the cluster-day; column 2 is the average core-hours; and column 3 is the average GB-hours. The rows represent different compute clusters. Although there is an occasional exception to day to day consistency within the same compute cluster, in general, the bar groups are very similar. However, along any plot column with the same categorical coordinate, we see considerable difference. Consider the first column, and compare compute clusters C and D for *sss*. In cluster C, *sss* consistently accounts for 10% of the task executions, but in cluster D, *sss* consistently accounts for over 45% of the task executions.

Figure 8 plots CV for the task classes in Table 3. We see that CV is always less than 100%, and is consistently less than 50%. This is a significant reduction from the large CVs in the coarse-grain task classification presented in Section 2. There is a subtlety in the foregoing comparison of CVs. Specifically, mean core-hours for a cluster-day is the sum of mean core-hours for the eight task classes. Given this relationship, is it fair to compare the CV of the sum of metrics with the CVs of the individual metrics?

To answer this question, we construct a simple analysis. We denote the individual metrics by $x_1, \cdots, x_n$, and the sum by $y = x_1 + \cdots + x_n$. For simplicity, we assume that the x's are identically distributed random variables with mean μ and variance σ^2. Further, since task classes contain a large number of tasks, it is reasonable to assume that the x_i are independent. The CV of each x_i is $\frac{\sigma}{\mu}$. But the CV of y is $\frac{\sqrt{n}\sigma}{n\mu}$, which is $\frac{1}{\sqrt{n}}$ of the CV of the x's. Applying this insight to our data, $n = 8$ and so we expect the CV of the sum to be about 30% of the CVs of the task class metrics. Instead, the CVs of mean core-hours for the cluster-day is at least twice as large as the CV of mean core-hours for task classes. This analysis applies to mean GB-hours as well, and yields a similar result.

We also assess the task classification in terms of the consistency of the class resource usages in compute cluster. Figure 9(a) plots mean core-hours by task class for each task-cluster-day studied with error bars for one standard deviation. Although task class *lll* has substantial variability, the mean values of the other task classes are grouped close to the center of the error bars. We observe a similar behavior for GB-hours in Figure 9(b).

4.3 Insights from Task Classification

The task classification constructed using the methodology in Figure 4 provides interesting insights into tasks running in the Google Cloud Backend. From the preliminary task classes in Table 1, we see that task durations are bimodal, either somewhat less than 30 minutes or larger than 18 hours. Such behavior results from the characteristics of application tasks running on the Google Backend Cloud. There are two types of long-running tasks. The first are user-facing. These tasks run continuously so as to respond quickly to user requests. A second type of long-running tasks are compute-intensive, such as processing web logs. Tasks handling end-user interactions are likely *lss* during periods of low user request rates, and *lll* during periods of high user request rates.

From Figure 7, we see that tasks with short duration dominate the task population. These tasks reflect the way the Google Cloud parallelizes backend work (often using map reduce). There are several types of short-running tasks. *sss* tasks are short, highly parallel operations such as index lookups and searches. *sml* tasks are short memory-intensive operations such as map reduce workers computing an inverted index. And, *slm* tasks are short cpu-intensive operations such as map reduce workers computing aggregations of log data.

Last, observe that a small number of long running tasks consume most of the CPU and memory. This is apparent from columns 2 (core-hours) and 3 (GB-hours) of Figure 7 by summing the contributions of task classes whose first qualitative coordinate is *l* (i.e., large duration). There are two kinds of tasks that account for this resource consumption. The first are computationally intensive, user-facing services such as work done by a map reduce master in processing web search results. The second kind of long-running tasks relate to log-processing operations, such as analysis of click throughs.

5. Applications

This section describes applications of task classification to capacity planning and task scheduling.

Capacity planning selects machine, network, and other resources with the objective of minimizing cost subject to the constraint that application tasks meet their service level objectives over a planning horizon. Typically, capacity planning is done iteratively using the following steps: (1) forecast application growth over a planning horizon (e.g., six months); (2) propose machine configurations; and (3) model or simulate application throughputs, resource utilizations and task latencies for the forecast workload on the proposed machine configurations. The task classifications developed in this paper provide a way for Google to forecast application growth by tracking changes in task resource consumption by task class.

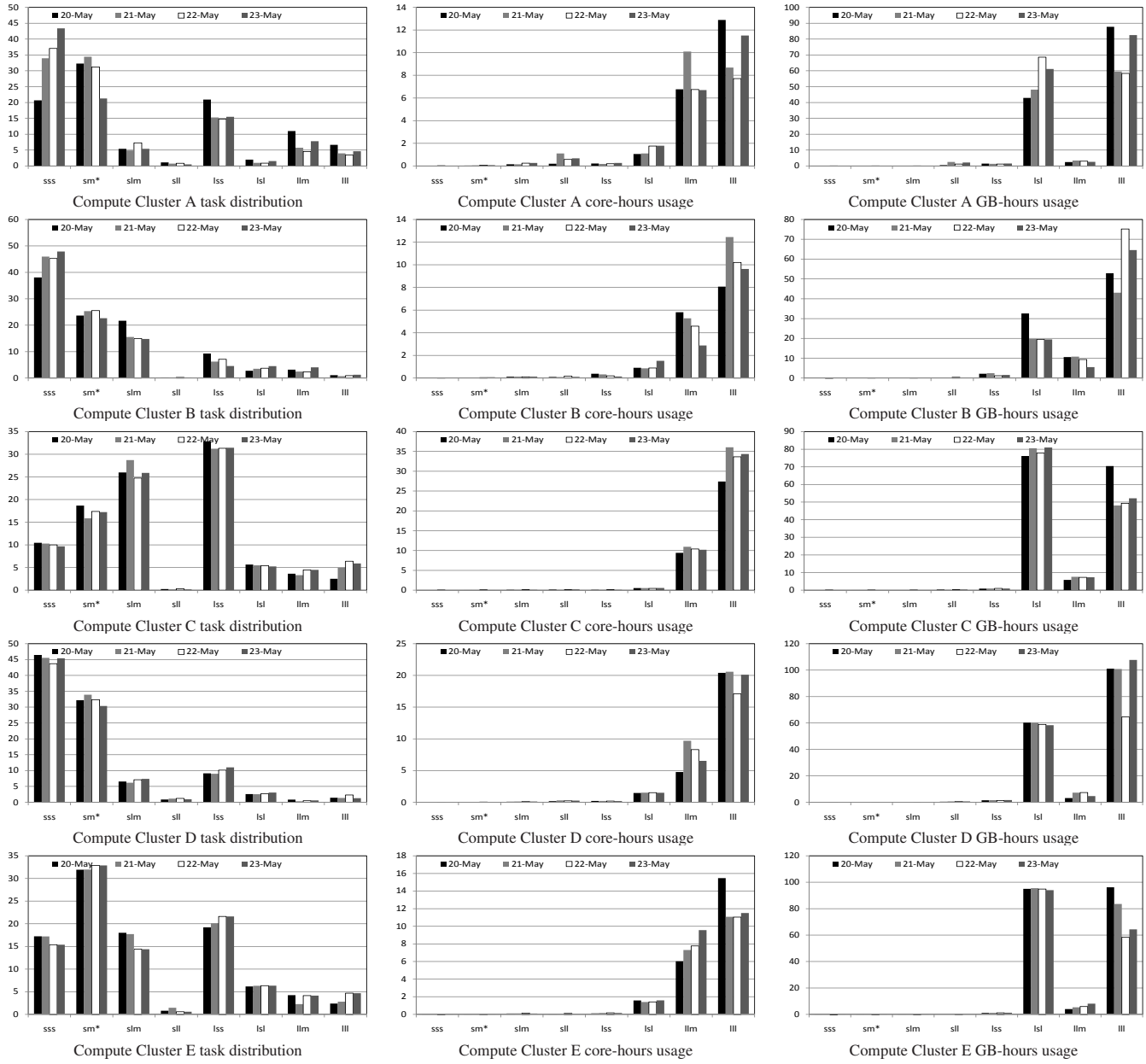


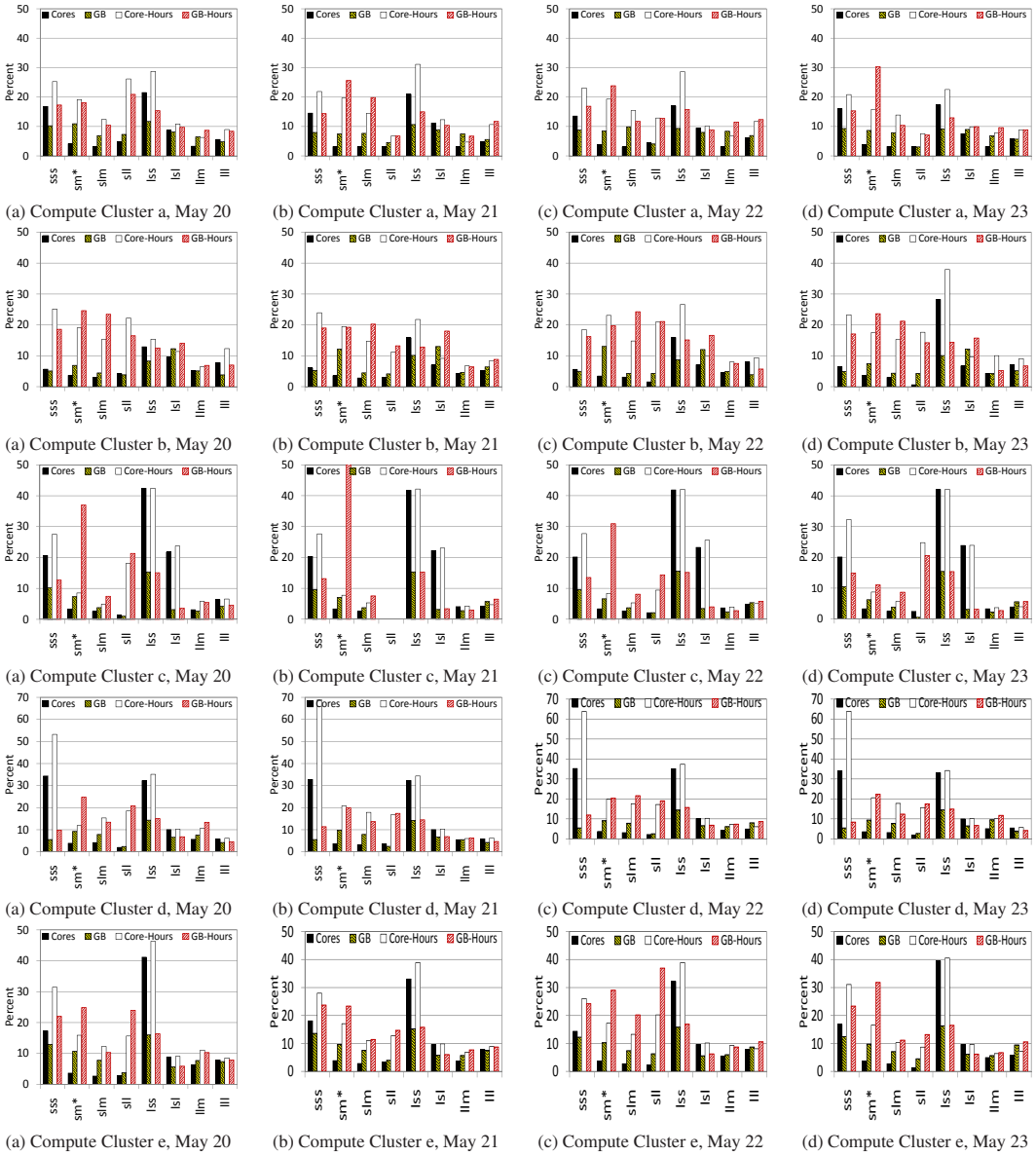
Figure 7. Percentage distribution of each class and their corresponding resource utilization for five compute clusters from 20 May - 23 May

A second application of our task classifications is to task scheduling. As noted previously, task scheduling can be viewed as multi-dimensional bin packing over time. Doing a good job of bin packing requires knowing the shape of the tasks running on machines (so that we know how much space remains) and knowing the shape of the task to be scheduled. We can use runtime statistics of tasks on machines to determine which task class they belong to. For example, consider a task that has run for more than two hours, has an average core usage of 0.4, and average memory usage of 0.2 GB. This task is likely *lss*. We can estimate the membership of a newly arrived task based on the (prior) distribution of task cluster memberships. Note that since most tasks have a short duration, inefficiencies are short lived if there is an incorrect classification of a newly arrived class. If it turns out that a long running task is assigned to an overloaded machine, we can stop and restart the task elsewhere. Although the stop and restart strategy introduces some inefficiencies, this happens rarely since there are few long-running tasks.

6. Related Work

There is a long history of contributions to workload characterization. For example, [3] uses statistical clustering to characterize workloads in IBM's Multiple Virtual Storage (MVS) Operating System. [12] describes the characterization of network traffic in 3-tier data centers; [8, 18, 4] model web server workloads; [24] address workloads of distributed file systems; and [6, 7, 13] describe large scientific workloads in compute clusters. The novel aspects of our work are the insights we provide into workloads for a large cloud backend with a diverse set of applications. These insights are facilitated by our use of clustering and the notion of qualitative coordinates. We note that others have questioned the efficacy of employing clustering in workload characterization [15, 5]. However, clustering is an important part of our methodology for task classification, although we do not rely entirely on automated approaches.

At a first glance, it may seem that workloads in the cloud backend are very similar to large scientific workloads, hereafter referred



(a) Compute Cluster a, May 20 (b) Compute Cluster a, May 21 (c) Compute Cluster a, May 22 (d) Compute Cluster a, May 23

(a) Compute Cluster b, May 20 (b) Compute Cluster b, May 21 (c) Compute Cluster b, May 22 (d) Compute Cluster b, May 23

(a) Compute Cluster c, May 20 (b) Compute Cluster c, May 21 (c) Compute Cluster c, May 22 (d) Compute Cluster c, May 23

(a) Compute Cluster d, May 20 (b) Compute Cluster d, May 21 (c) Compute Cluster d, May 22 (d) Compute Cluster d, May 23

(a) Compute Cluster e, May 20 (b) Compute Cluster e, May 21 (c) Compute Cluster e, May 22 (d) Compute Cluster e, May 23

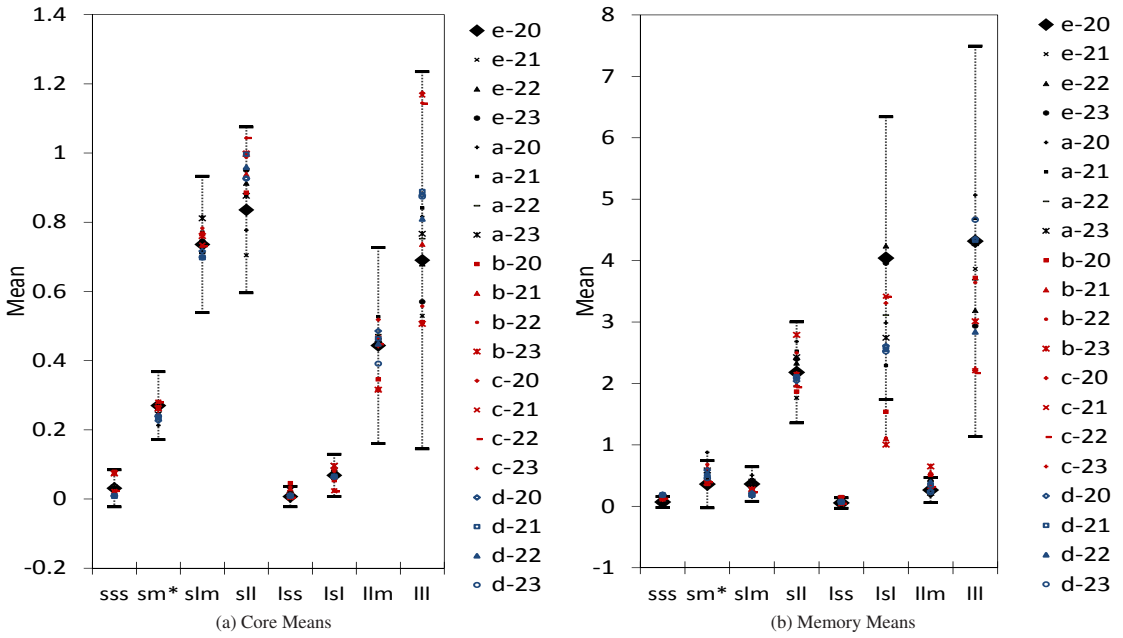
Figure 8. Coefficient of Variation by Task Class

to as high performance computing (HPC) workloads. There is well-developed body of work that models HPC workloads [7, 10, 11, 14, 21, 22]. However, HPC researchers do not report many of the workload characteristics discussed in this paper. For example, we report task durations that have a strong bimodal characteristic. Although HPC researchers report a large concentration of small jobs, such jobs are thought to be the result of mistakes in configuration and other errors [9]. In contrast, short duration tasks are an important part of the workload of the Google Cloud Backend as a result of the way work is parallelized. Further, there is a substantial difference in HPC long-running work compared to that in the Google Cloud Backend. HPC long-running work typically has characteristics of "batch jobs" that are compute and/or data intensive. While compute and data-intensive work runs in the Google Cloud Backend, there are also long-running tasks that are user-facing whose resource demands are quite variable.

Finally, HPC workloads and the Google Cloud Backend differ in terms of the workload dimensions and relevant aspects of hardware configurations. For example, in HPC there is considerable investigation into the relationship between requested time and execution time, a fertile ground for optimization [23] and debate [19]. In cloud computing, services such as web search and email do not have a requested time or deadline. Moreover, in cloud backends, the use of admission control and capacity planning ensure that jobs have enough resource to run with little wait. Such considerations are usually foreign to HPC installations. Still another area of difference is the importance of inter-task communication and therefore the bandwidth of the inter-connect between computers. We observe that for some HPC applications there is a small ratio of computation-to-communication, and thus there is a requirement for specialized interconnects [20, 2]. Such considerations are relatively unimportant in the Google Cloud Backend.

7. Conclusions

This paper develops a methodology for task classification, and applies the methodology to the Google Cloud Backend. Our methodology for workload classification consists of: (1) identifying the workload dimensions; (2) constructing task classes using an off-the-shelf algorithm such as k-means; (3) determining the break points for qualitative coordinates within the workload dimensions; and (4)



(a) Core Means (b) Memory Means

Figure 9. Consistency of class means with respect to cpu and memory across compute-cluster-days

merging adjacent task classes to reduce the number of workloads. We use the foregoing, especially the notion of qualitative coordinates, to glean several insights about the Google Cloud Backend: (a) the duration of task executions is bimodal in that tasks either have a short duration or a long duration; (b) most tasks have short durations; and (c) most resources are consumed by a few tasks with long duration that have large demands for CPU and memory.

Although, our current study considers a modest amount of data (20 cell days), we believe our results are suggestive of a general trend in the Google Cloud Backend. As such, our results can aid system designers with capacity planning and task scheduling. Additionally, the technical approach used - forming qualitative coordinates using clustering - is simple and produces intuitively reasonable results.

We plan to extend this study by evaluating more data and more cells. We also hope to address characterization of the task arrival process, and extend our task classification to consider job constraints (e.g., co-locating two tasks in the same machine).

References

[1] *Google Cluster Data, http://googleresearch.blogspot.com/2010/01/google-cluster-data.html*.

[2] Y. Aridor, T. Domany, O. Goldshmidt, E. Shmueli, J. Moreira, and L. Stockmeier. *Multi-Toroidal Interconnects: Using Additional Communication Links to Improve Utilization of Parallel Computers*. Job Scheduling Strategies for Parallel Processing, LNCS, 2004.

[3] H. P. Artis. Capacity Planning for MVS Computer Systems. *SIGMETRICS PER*, 8(4):45–62, 1980.

[4] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *SIGMETRICS PER*, 1998.

[5] M. Calzarossa and D. Ferrari. A Sensitivity Study of the Clustering Approach to Workload Modeling. In *SIGMETRICS '85: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1985.

[6] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *Proceedings of the IEEE*, 81(8):1136–1150, Aug 1993.

[7] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In *Job Scheduling Strategies for Parallel Processing, LNCS*. 1999.

[8] L. Cherkasova and P. Phaal. Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites. *IEEE Transactions on Computers*, 51(6):669–685, 2002.

[9] W. Cirne and F. Berman. A Comprehensive Model of the Supercomputer Workload. In *4th Workshop on Workload Characterization*, pages 140–148, Dec 2001.

[10] M. E. Crovella. Performance Evaluation with Heavy Tailed Distributions. In *Job Scheduling Strategies for Parallel Processing, LNCS*. 2001.

[11] A. B. Downey and D. G. Feitelson. The Elusive Goal of Workload Characterization. *SIGMETRICS PER*, 1999.

[12] D. Ersoz, M. S. Yousif, and C. R. Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *ICDCS '07: Proceedings of the 27th Intl. Conference on Distributed Computing Systems*, 2007.

[13] D. G. Feitelson. Metric and Workload Effects on Computer Systems Evaluation. *Computer*, 36(9):18–25, Sep 2003.

[14] D. G. Feitelson and B. Nitzberg. Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing, LNCS*. 1995.

[15] D. Ferrari. On The Foundations of Artificial Workload Design. In *SIGMETRICS '84: Proceedings of the 1984 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 1984.

[16] J. A. Hartigan. *Probability and Mathematical Statistics*. John Wiley, 1975.

[17] J. L. Hellerstein, F. Zhang, and P. Shahabuddin. A Statistical Approach to Predictive Detection. *Computer Networks*, 2001.

[18] E. Hernández-Orallo and J. Vila-Carbó. Web Server Performance Analysis using Histogram Workload Models. *Comput. Netw.*, 2009.

[19] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are User Runtime Estimates Inherently Inaccurate? In *Job Scheduling Strategies for Parallel Processing, LNCS*. 2004.

[20] F. Petrini, E. Frachtenberg, A. Hoisie, and S. Coll. Performance Evaluation of the Quadrics Interconnection Network. *Cluster Comput.*, 6(2):1125–142, Apr 2003.

[21] B. Song, C. Ernemann, and R. Yahyapour. Parallel Computer Workload Modeling with Markov Chains. In *Job Scheduling Strategies for Parallel Processing, LNCS*. 2004.

[22] D. Talby, D. G. Feitelson, and A. Raveh. A Co-Plot Analysis of Logs and Models of Parallel Workloads. *ACM Transactions on Modeling & Comput. Simulation (TOMACS)*, 12(3), Jul 2007.

[23] D. Tsafrir and D. G. Feitelson. The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help. In *IEEE International Symposium on Workload Characterization (IISWC)*, 2006.

[24] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty. File System Workload Analysis for Large Scale Scientific Computing Applications. In *Proc. of the 21st IEEE / 12th NASA Goddard Conf. on Mass Storage Systems and Tech.*, 2004.