Noname manuscript No.
(will be inserted by the editor)

Characterizing and Modeling Cloud Applications/Jobs on a Google Data Center

Sheng Di · Derrick Kondo · Franck Cappello

Received: date / Accepted: date

Abstract In this paper, we characterize and model Google applications and jobs, based on a one-month Google trace from a large-scale Google data center. We address four contributions: (1) we compute the valuable statistics about task events and resource utilization for Google applications, based on various types of resources and execution types; (2) we analyze the classification of applications via a K-means clustering algorithm with optimized number of sets, based on task events and resource usage; (3) we study the correlation of Google application properties and running features (e.g., job priority and scheduling class); (4) we finally build a model that can simulate Google jobs/tasks and dynamic events, in accordance with Google trace. Experiments show that the tasks simulated based on our model exhibit fairly analogous features with those in Google trace. 95+% of tasks' simulation errors are less than 20%, confirming a high accuracy of our simulation model.

Keywords Google data center, Cloud task, Characterization and Analysis, large-scale system trace

This work was supported by ANR project Clouds@home (ANR-09-JCJC-0056-01), also in part by the Advanced Scientific Computing Research Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

S. Di
INRIA (France)
Tel.: +1-919-308-8050
E-mail: sheng.di@inria.fr

D. Kondo
INRIA (France)
E-mail: derrick.kondo@inria.fr

F. Cappello
Argonne National Laboratory (USA)
E-mail: cappello@mcs.anl.gov

1 Introduction

Cloud computing [1,2] has emerged as a compelling paradigm for the easy-to-use and fine-grained resource consumption on the Internet. Workload characterization and modeling for cloud applications is essential for optimizing the system-wide resource allocation in cloud systems.

Google is a well-known cloud platform, on which there are millions of requests to process across hundreds of thousands of data centers everyday. In November of 2011, Google [3,4] released its one-month trace for researchers to study. The trace involves over 650k jobs across over 12k heterogeneous hosts from a data center. There are totally 40k applications, which are repeatedly called/used by thousands of users in the form of jobs, and each job is executed in the form of one or more tasks. Over 144 million task events are recorded. For confidentiality, Google intentionally hides some information like application names and absolute values of resource usage, yet hashed application names and released the relative values of resource usage, which will not impact the statistical analysis and simulation of Google trace.

In our previous work [5], we characterized the hostload for each host, by aggregating its running tasks' resource usage on different resource types over time. We also present some new insights about the differences of hostload between Cloud and Grid. We found Google hostload exhibits higher variance and larger noise than other Grid/HPC systems, because of the much shorter Google job length and higher job submission frequencies.

In this paper, we focus on the characterization of Google application features statistically. We will mainly answer four such questions:

- What are the particular statistics about workloads, task events and resource utilization, with respect to Google applications?
- Can we classify Google applications based on the way their corresponding jobs are executed? For example, can we find any correlations between task events and applications' execution types (a.k.a., application types) like whether the applications can run batch-tasks or not?
- From the perspective of applications, are there strong or weak correlations between task events and running features like job priority and task scheduling class[1]?
- How to build a simulation model based on the above characterization work, to emulate Google jobs/tasks and events? What is the accuracy of the simulation as compared to the original Google trace?

Our work will particularly benefit the further research on cloud resource allocation in the long run. As a matter of fact, many contemporary cloud resource allocation strategies already tried to optimize the performance, by taking advantage of pre-knowledge about application workload or features. For example, Meng et al. [6] endeavored to optimize the resource allocation

[1] scheduling class (0-3), according to [3], roughly represents how latency-sensitive a job/task is, with 3 representing a more latency-sensitive task and 0 representing a non-production task

by analyzing the compatibility of running applications encapsulated in virtual machines (VM) based on their resource usage patterns. Inter-cloud [7] and Stillwell's virtual resource allocation strategy [8], both assume application service workload and behaviors are predictable in their cloud service provisioning model. Other simulation research (such as [9,10]) on cloud computing always emulate the cloud application workload or events before its further investigation. Obviously, comprehensive characterization of cloud application features is a prerequisite for the further improvement of cloud system performance.

To the best of our knowledge, our work is the first attempt to comprehensively study the statistical features of cloud applications based on a real production trace. Although Google just released one-month period of trace data involved with 12k hosts, one can emulate Google job submissions and related task events based on our simulation model, for longer test period like one year and a larger system scale with more hosts.

The remainder of the paper is organized as follows. In Section 2, we briefly introduce the Google trace and show the overview of Google's job scheduling system, which serves as a fundamental background of the following analysis. In Section 3, we present some key findings about Google application properties, such as the distribution of task events based on application types and classification of applications with optimized K-means clustering algorithm. In Section 4, we explore the correlation between application properties and running features, including job priority and task scheduling class. In Section 5, we discuss how to simulate jobs based on the characterization of Google application features, and also evaluate the validity of our simulation model by comparing to the original trace. We comprehensively discuss the related works and highlight the key contributions of our work in Section 6. We conclude the paper with a vision of the future work in Section 7.

2 System Overview

A *Google data center* consists of thousands of hosts that are connected via a high-speed intra-network. One or more schedulers receive and process a large number of user requests (a.k.a., jobs), each of which is comprised of one or more tasks. For instance, a map-reduce [11] program will be treated as a job with multiple reducer tasks and mapper tasks. Different jobs are assigned with different scheduling priorities, and there are 12 priorities in total. Each task (actually represented as a Linux program possibly consisting of multiple processes) is always generated with a set of user-customized requirements (such as the minimum CPU rate and memory size).

According to Google's usage trace format [4], each task can only exist in one of the following four states, *unsubmitted*, *pending*, *running* and *dead*. The detailed task scheduling mechanism follows a state-transition graph, which can be found in [4]. The task states transit based on various task events, and there are totally 9 different event types, which are represented as 0 (task submission),

1 (schedule), 2 (evict), 3 (fail), 4 (finish), 5 (kill), 6 (lost), 7 (update_pending), and 8 (update_running) respectively.

Based on Google's task processing model [4], Google traced over 650k jobs that were scheduled across over 12000 heterogeneous machines within one month. More than six metrics are collected during the one month of task-event monitoring, such as CPU usage, assigned memory, observed real memory usage, page-cache memory usage, disk I/O time, and disk space.

Each job corresponds to a specific application, which is named as "logic job name" in Google trace. Berkeley's report [12] simply reveals some features of Google applications. For example, there are totally about 40k different applications in the trace. The number of jobs per application loosely follows a Zipf-like distribution, and a few applications are shared among an extremely large number of jobs (e.g., up to 22k). In this paper, we intensively characterize the Google application features in order to support a precise simulation of Google cloud environment.

3 Characterization of Google Application Properties

In this section, we mainly analyze the statistical properties of Google applications. For example, the distribution of the number of jobs/events per application, the distribution of task events based on various application types, the optimized clustering for applications based on task events and resource utilization.

3.1 Mass-count Disparity of Task Events and Resource Usage

We first present the distribution of the number of jobs/events per application, through mass-count disparity evaluation. Mass-count [13] is a very important metric used to extract the key features (such as heavy tails) for specific distributions. It is made up of the "count" distribution and the "mass" distribution. The "count" distribution simply refers to the cumulative distribution function (CDF) as it counts how many items are smaller than certain size. The "mass" distribution weights each item, specifying the probability that a unit of mass belongs to an item. Specifically, their values are calculated based on Formula (1) and Formula (2), where $f(t)$ refers to the probability density function.

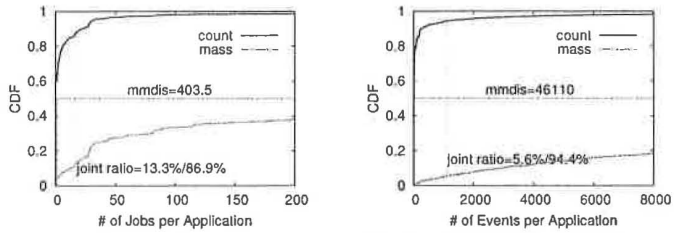
$$F_c(x) = \Pr(X < x) \tag{1}$$

$$F_m(x) = \frac{\int_0^x t \cdot f(t)\, dt}{\int_0^\infty t \cdot f(t)\, dt} \tag{2}$$

By comparing the two curves (mass and count), we can determine whether the distribution follows Pareto principle [14], heavy tails, or other statistical features. In the analysis, *joint ratio* (a kind of Gini coefficient [13]) is a critical measure index, defined as X/Y, meaning that X% of the items account for Y% of the mass and Y% of the items account for X% of the mass. A typical

Pareto principle means that X and Y are very small and very big respectively, for example, X=10% and Y=90%. The *mm-distance* (abbreviated as mmdis.) shown in the figure is defined as the horizontal distance of the two points that are right in the middle of the CDF of the Count curve and Mass curve. Longer distance means a stronger Pareto principle (a more non-uniform distribution about the mass).

Figure 1 (a) and (b) show the mass-count disparity, i.e., the mass per application versus the count per application.



(a) # of Jobs per Application (b) # of Events per Application

Fig. 1 Mass-Count Disparity of the Number of Jobs/Events per Application

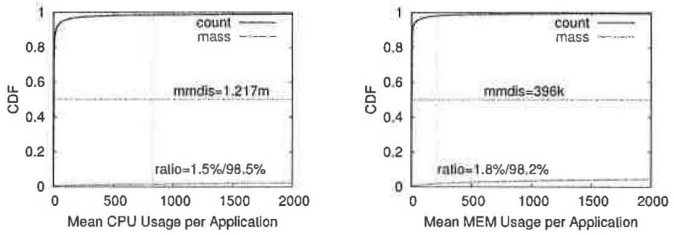
Statistics indicate that 60% of applications each just have only one job and over 80% of applications have no more than 82 task events. That is, the number of jobs/events per application for a large majority of applications is very small. Through the two figures, we can also clearly observe a typical Pareto principle. Specifically, 86.7% of jobs belong to only 13.3% of applications and 13.3% of jobs belong to 86.7% of applications. Similarly, Figure 1 (b) shows that only 5.6% of task events belong to 94.4% of applications and 94.4% of task events belong to only 5.6% applications. That is, the distribution of the number of jobs/events is extremely non-uniform and a large majority of applications only account for very few jobs/events.

We also study the distribution of the CPU workload (or CPU usage) and memory workload (or memory usage) per application, through mass-count disparity evaluation. The CPU workload is evaluated by *core seconds*. For example, if one job has two tasks executed in parallel, each of which is using 2 cores all the time and their lengths are 100 and 200 seconds respectively, then this job's CPU workload is equal to 100×2+200×2=600. A job's total memory workload is evaluated by *memory size seconds*. For example, if one job has two tasks, each of which consumes 0.05 memory size[2] on average, and their execution lengths are both 100 seconds, then, the job's total memory workload is equal to 0.05×100×2=10. One application's workload on CPU or

[2] Google trace does not expose the exact memory size used by jobs but their scaled values compared to the maximum memory capacity of each node. For example, suppose the maximum memory capacity on a host is 64GB, 0.05 memory size means 0.05×64=3.2GB

memory is computed as the average value of all of its job workloads in the trace.

In comparison to task events, we just take into account the resource usage of 18k valid applications, which are completed successfully as recorded in the trace. Via Figure 2, we find that both CPU workload and memory workload per application follow a considerably typical Pareto principle. Specifically, only 1.5% (1.8%) of applications contribute to up to 98.5% (98.2%) of CPU (memory) usage on average, and 98.5% (1.8%) of applications consume extremely few CPU (memory) usage, i.e., only 1.5% (1.8%). In other words, the resource utilization per application in a simulated cloud environment is supposed to conform to such a Pareto principle, otherwise, the emulated benchmark is skewed against the reality more or less.



(a) Mean CPU Usage per App (b) Mean MEM Usage per App

Fig. 2 Mass-Count Disparity of Resource Usage per Application

3.2 Task Event Distribution based on Application Types

Based on the job's intrinsic structure (i.e., how many tasks per job and how to connect them), we split the 40k applications into four execution types (or application types), single-task application, sequential-task application, batch-task application and mix-mode application. *Single-task application* means that the corresponding job just has only one task. *Sequential-task application* indicates that for this application, the tasks in each corresponding job are generated (or invoked, submitted) in series. That is, the workload of a whole job will be completed in form of many small tasks connected one by one, and no two tasks' execution periods overlap each other. Such an application type often implies frequent fail/evict/kill/lost events[3], which is consistent with Berkeley

[3] According to Google trace [4], there are different factors for task interruptions: (1) failure event: a task or job was descheduled (or, in rare cases, ceased to be eligible for scheduling while it was pending) due to a task failure; (2) evict event: a task or job was descheduled because of a higher priority task or job, because the scheduler overcommitted and the actual demand exceeded the machine capacity, because the machine on which it was running became unusable, or because a disk holding the task's data was lost; (3) kill event:

report [12] (many Google jobs suffer from the crash-loop phenomenon, wherein the tasks submitted are repeatedly failed, evicted, killed or lost). For *batch-task applications*, each job contains at least two tasks executed in an embarrassingly parallel pattern. *Mix-mode application* indicates a mixed type of the two application types, sequential-tasks and batch-tasks.

Table 1 shows the distribution of the number of applications and various task events per application type.

Table 1 Distribution of Events w.r.t. Application Types

	Sing.-Task	Seq.-Task	Batch-Task	Mix-Mode
# of Applications	25513	1015	9910	3286
# of Evict Events	41004	546302	1645534	3631513
# of Fail Events	165551	69565	11242109	2352544
# of Finish Events	368886	0	1654416	16194673
# of Kill Events	151932	260439	1934749	8002560
# of Lost Events	113	904	1138	6599

It is observed that most of applications (over 64%) each just have single task, and 25% applications correspond to batch tasks. Only 2.6% and 8.3% of applications raise sequential tasks and mix-mode tasks respectively. Through this table, not only can we realize that finish-events and fail-events account for the major portion in all of task events, we can also compare the number of task events based on different application types. For example, we find that evict events rarely appear for single-task jobs, but usually happen in either a batch-task application or a mix-mode application. Only 0.5% and 1.2% of fail events belong to single-task applications and sequential-task applications respectively, while about 81.3% belong to batch-task applications. Also, both the kill events and lost events mainly belong to batch-task application and mix-mode application. In addition, the *finish events* mainly exhibit with mix-mode applications. This means that mix-mode application type works much more effectively than other types. In contrast, there are no sequential-task applications finished normally based on the Google trace. This further confirms a typical crash-loop phenomenon, as reported by C. Reiss et al. [12].

3.3 Optimized Clustering of Google Applications

We further cluster applications based on the statistics of task events and resource utilization. We believe such a work revealed some crucial features about Google applications hidden in the trace, which will definitely benefit the in-depth understanding and simulation of a large-scale cloud benchmark in the long run.

The major methodology is K-means clustering algorithm [15], since it can effectively partition data into Voronoi cells [16]. Its outcome contains multiple

a task or job was canceled or another job or task on which this job was dependent died; (4) lost event: a task or job was presumably terminated with a missing record.

sets each containing a unique center and the Euclidean distances of the samples in a set to the set's center must be smaller than to any other sets' centers. Given various numbers of sets, the clustering may be largely different, hence, we also explored the optimized number (i.e., optimized K) of sets under different classification degrees.

The objective of our K-means clustering algorithm is to cluster the applications into several sets, such that the within-cluster sum of squares (WCSS) could be minimized under a specific degree of classification. We define *merge ratio (MR)* to be the ratio of distance(α,β) to the average distance among all centers (denoted by $\overline{d}$), where distance(α,β) denotes the distance between two set centers α and β. We also define a threshold, called *Merge Ratio Threshold (denoted by λ)*, to determine the degree of classification. Obviously, $\lambda\overline{d}$ will serve as the threshold in merging two set centers: if the two set centers are closer than $\lambda\overline{d}$, the corresponding sets should be merged. In general, λ is in the range (0,1]. We denote by KM(k,S,φ) the converged solution to the clustering on the sample set S via K-means clustering algorithm, and denote the set of corresponding centers as CS(k,S,φ), where φ indicates the initial set of k centers.

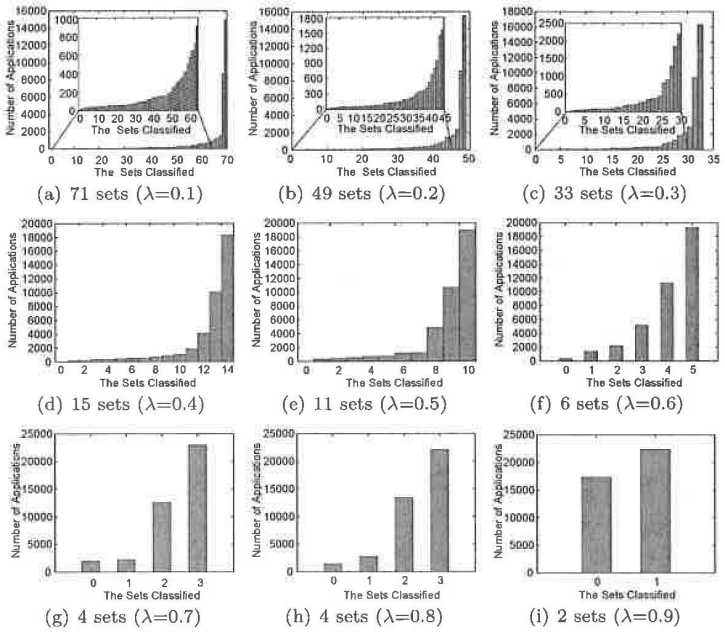
Algorithm 1 K-Optimization Algorithm

Input: N sample data, max # of sets (denoted M), λ (Merge Ratio Threshold);
Output: the number of sets, converged centers, and classified sets of data
begin
1: CS = the set of M Centers initialized by Forgy method [26].
2: **repeat**
3: Compute $KM(|CS|,S,CS)$;/*$|CS|$ denotes # of elements in CS*/
4: Compute average distance for CS, denoted as $\overline{d}$;
5: **if** ($\exists\ \alpha\in CS,\beta\in CS$, distance($\alpha$,$\beta$)$<\lambda\cdot\overline{d}$) **then**
6: CS=*MergeCenters*(CS,$\lambda\cdot\overline{d}$);/*Merge nearby centers in CS*/
7: **else**
8: break;
9: **end if**
10: **until** ($|CS|$=2);
11: Output $|CS|$, CS, and $KM(|CS|,S,CS)$;
end

Algorithm 1 aims to optimize the clustering for Google applications, with optimized number of sets. In our experiments, the initial maximum number of sets (i.e., M) is set to 100, and we use Forgy method to randomly find the optimal solution to the initial case. After that, two relatively nearby centers will be merged by using their middle point (line 6) and a new K-means clustering will be performed based on the new set of centers. The two steps will repeat until each pair of centers are farther than $\lambda\cdot\overline{d}$, where $\overline{d}$ refers to the average distance among the centers of the sets.

3.3.1 Task Event based Application Clustering

Figure 3 shows the optimized clustering of Google applications based on task event trace, under different degrees of classification. We zoom in the first three sub-figures for clear observation. Based on the 5 types of task events (as shown in Table 1), there are totally 5 dimensions per application. The overall clustering is performed based on the normalized probability of each task event type per application. For example, if the numbers of the five events are 100,200,300,400,500 respectively for one application, the coordinate of this application will be $(\frac{1}{15}, \frac{2}{15}, \frac{3}{15}, \frac{4}{15}, \frac{5}{15})$. The degree of classification is determined by *Merge Ratio Threshold* (denoted as λ), which is defined in Appendix. The lower λ is, the finer granularity of the classification is. For example, when λ is set to 0.1 and 0.5 respectively, all applications can be grouped into 71 sets and 11 sets respectively.



(a) 71 sets (λ=0.1) (b) 49 sets (λ=0.2) (c) 33 sets (λ=0.3)

(d) 15 sets (λ=0.4) (e) 11 sets (λ=0.5) (f) 6 sets (λ=0.6)

(g) 4 sets (λ=0.7) (h) 4 sets (λ=0.8) (i) 2 sets (λ=0.9)

Fig. 3 Distribution of Applications in The Optimized Clustering based on Task Events

From Figure 3 (a) - (i), we observe that the number of applications per set does not follow a uniform or normal distribution but a Pareto distribution with a long tail. That is, most of clustering sets each just contain dozens of or hundreds of applications, while minority of sets each contains thousands of applications or more. For example, Figure 3 (d) shows that the largest set

contains over 18k applications while the smallest set just contains 13 applications.

As shown by the above figure, the applications can be classified based on the distribution of task/job event types, based on which we can deeply understand and accurately simulate the task/job events with respect to application types. Table 2 shows the centers of 4-set clustering (λ=0.8) and 2-set clustering (λ=0.9) respectively. Based on the 4-set clustering, we find that if the task events of an application are mostly *evict* events, the remaining events will be likely kill events, which accounts 18% in the total number of events. Similarly, if the fail events dominate the task events for a particular application (its probability is about 67.5%), its kill events will also exhibit prominently in the rest of events (the likelihood is up to $\frac{0.18}{1-0.675}$=55.4%). By contrast, if an application is often finished (killed) eventually (e.g., with 92.6% finish rate), other tasks of this application will likely be finished or killed. In addition, since the numbers of applications in the four sets are about 2.7k, 1.5k, 13.5k, and 22k respectively, we know that most of applications should terminate with either a finish event or a kill event. Note that kill event [3] is mainly due to an external factor like the cancellation by its user or the death of another dependent job, thus it may also be counted as a normal event by excluding the external factors like interruption of users. Hence, it can be concluded that a large majority of applications are prone to be finished normally without considering external interruptions.

Table 2 Centers of Clustering Sets based on Task Events

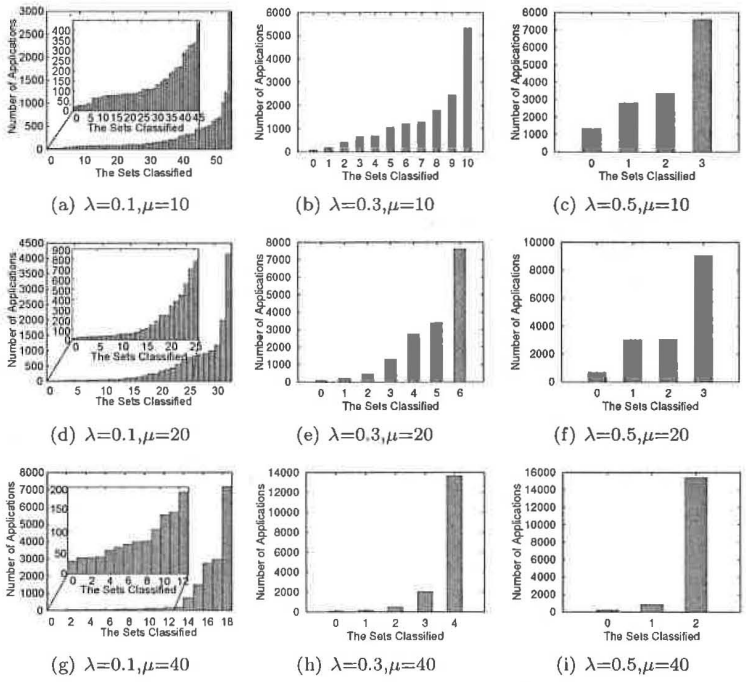
	(Evict rate, Fail rate, Finish rate, Kill rate, Lost rate)
4-Set Classification (λ=0.8)	(0.616 , 0.023 , 0.053 , 0.181 , 0.00025) (0.039 , 0.675 , 0.106 , 0.180 , 0.00035) (0.019 , 0.008 , 0.083 , 0.890 , 0.00025) (0.007 , 0.005 , 0.926 , 0.061 , 0.00025)
2-Set Classification (λ=0.9)	(0.012 , 0.011 , 0.919 , 0.058 , 0.00025) (0.109 , 0.059 , 0.074 , 0.737 , 0.00025)

3.3.2 Workload based Application Clustering

We also study the application clustering based on workload (or resource utilization). We calculate the mean CPU workload and mean memory workload for each application based on the workloads of its corresponding jobs. Then, we run our optimized clustering algorithm on all of 18k valid applications. For each application, there are two dimensions, which indicate the estimated CPU workload (core seconds) and memory workload (memory size seconds) over time respectively. Statistics show that a large majority of applications each use less than one core seconds on average and only a tiny of them (batch-task applications) each consume over 10 core seconds on average. Accordingly, we mainly focus on the major portion, i.e., the applications whose mean workloads are not extremely large. That is, we filter out the applications whose

radiuses (i.e., the distance between its coordinate to the origin point - (0,0)) are farther than a threshold (namely *radius threshold*, denoted as μ). We perform the clustering algorithm on all of Google applications, based on various merge rate thresholds (λ) and radius thresholds (μ).

In Figure 4, we present the distribution of the number of applications in the optimized clustering sets based on workload. We observe the number of applications per set always follows a Pareto-similar distribution (or power law). With bigger λ or bigger μ, the granularity of classification becomes coarser. For example, Figure 4 (i) shows there are only 3 clustering sets, if the distance between any pair of centers is kept bigger than 0.5 times as long as the average distance and any radius is limited to be no larger than 40.



(a) λ=0.1,μ=10 (b) λ=0.3,μ=10 (c) λ=0.5,μ=10

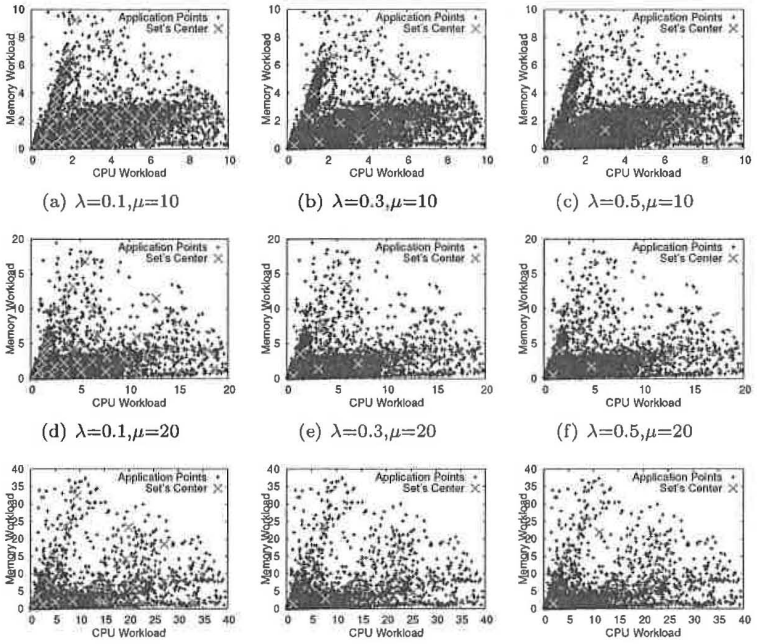
(d) λ=0.1,μ=20 (e) λ=0.3,μ=20 (f) λ=0.5,μ=20

(g) λ=0.1,μ=40 (h) λ=0.3,μ=40 (i) λ=0.5,μ=40

Fig. 4 Distribution of Applications in Optimized Workload Clustering

We further present the locations of the centers of the sets classified in Figure 5. The centers of the three classified sets shown in Figure 5 (i) are (11.198,21.835), (21.673,5.351), and (1.992,1.420) respectively. It is clearly observed that majority of applications are with quite low workload (or resource utilization) in the system. In addition, we observe that the number of applications in the sets classified based on the workload also follows a Pareto-similar

distribution. That is, for a few sets, each contains an extremely large number of applications with little resource consumption, while most of classified sets contain a small portion of applications with high resource utilization per set. Note that most of applications are located near to the origin point (0,0), which is due to either Google application's low CPU utilization and memory usage or its short execution length. This means that majority of Google applications' total workloads are tiny.



(a) λ=0.1,μ=10 (b) λ=0.3,μ=10 (c) λ=0.5,μ=10

(d) λ=0.1,μ=20 (e) λ=0.3,μ=20 (f) λ=0.5,μ=20

(g) λ=0.1,μ=40 (h) λ=0.3,μ=40 (i) λ=0.5,μ=40

Fig. 5 Application Workload and Centers of Clustering Sets

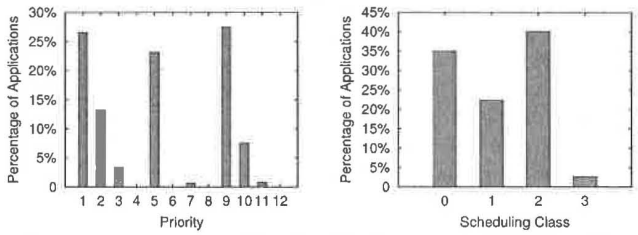
4 Correlation of Google Application Properties and Running Features

We study the correlation of Google application properties (mainly about statistics of different types of task events) and running features, including job priority and scheduling class. In Google trace, priority (1-12) and scheduling class (0-3) are used in job scheduling. Bigger priority value indicates higher execution priority. Bigger value of scheduling class implies a more latency-sensitive

task (e.g., serving revenue-generating user requests) while smaller value means a non-production task (e.g., development, non-business-critical analysis, etc.).

In our study, we find there exists a partial correlation (or weak correlation). In the following text, we first show the distribution of the number of applications based on running features, including scheduling class and job priority. And then, we characterize the correlation between application task events and the running features.

Figure 6 (a) presents the percentage of applications with respect to the job priority. It is observed that applications are distributed non-uniformly based on job priorities. All applications can be classified into three groups, low-priority, mid-priority, and high-priority. Most of applications are located at priority 1, 2, 3, 5, 9, and 10. Figure 6 (b) shows the percentage of applications with respect to job scheduling class. Since larger scheduling class value implies more latency-sensitive task and smaller value means a non-production task, Figure 6 (b) indicates that Google applications are not very sensitive to latencies and tend to be non-production tasks in general.

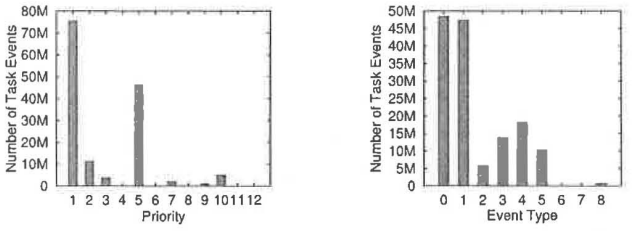


(a) Percentage of App w.r.t. Priority (b) Percentage of App w.r.t. Class

Fig. 6 Percentage of Applications w.r.t. Static States

Figure 7 shows the number of task events w.r.t. task features, including job priority and task event type. We can observe that a large majority of task events occur with relatively low job priorities like 1, 2, 3, and 5. The distribution of task events w.r.t. event types is also fairly non-uniform, based on Figure 7 (b). Specifically, in comparison to over 45 million task submission events, there are less than 20 million task finish events and about 10 million task kill events (Note that kill event is due to external interrupt by task user, so it may not be treated as abnormal event.). That is, there are over $\frac{45-20-10}{45}=\frac{1}{3}$ of abnormal task events, such as task fail, task evict, and task lost. In other words, when emulating a real cloud benchmark or environment, one has to carefully investigate the situation with such a high rate of abnormal task events.

We statistically exploit the distribution of task events based on job priorities, which reveals a partial correlation (or weak correlation) between task event types and job priorities. Through Figure 8, it is observed that a large majority of task evict events belong to the lowest-priority tasks, which is be-



(a) # Task Events w.r.t. Priority (b) # Task Events w.r.t. Event Type

Fig. 7 Distribution of Task Events w.r.t. Properties

cause of a fairly high probability of low-priority tasks being preempted by high-priority ones. We also observe that the number of normal task events (including task finish events and task kill events) decreases with increasing priorities for low-priority task events (e.g., priority value = 1, 2, 3). This is mainly due to the decreasing number of task events with increasing task priorities. In comparison, we find that the normal event ratios (defined as the ratio of the number of normal events to the total number of five types of events listed in Figure 8) for relatively low priorities (=1,2,3,5) are 35.6%, 68.9%, 94.0% and 95.5% respectively. This means that priority is a key factor that determines the task event ratio for low-priority tasks to a certain extent, because the tasks with lower priorities are prone to be preempted. However, such a rule does not fit the high-priority tasks. For example, the percentages of task finish events reach up to 100% for the three priorities, 6, 8 and 12, while the normal event ratios for priority 7, 9, 10, and 11 are limited to 92.2%, 92.6%, 17.3%, and 30.0% respectively.

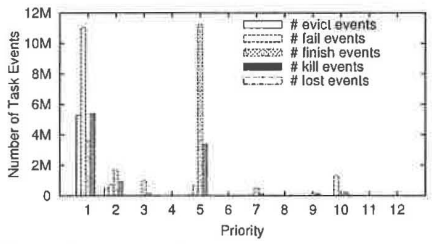


Fig. 8 Distribution of Event Types w.r.t. Priority

Finally, we present the distribution of task events based on scheduling class, in Figure 9. One interesting observation is that the number of task events for each of the three types (evict, fail and finish) decreases quickly with increasing scheduling class values. Moreover, unlike the correlation between task events and priority, the normal event ratios based on the four scheduling class values

are 57.3%, 57.9%, 82.6%, and 38.5% respectively, delivering a rather uniform distribution especially for low-scheduling-class tasks.

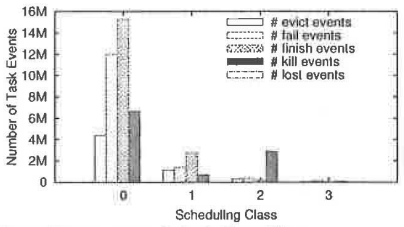


Fig. 9 Distribution of Event Types w.r.t. Scheduling Class

5 Simulation Model based on Google Application Features

Based on Google application features extracted from Google's one-month trace data, we build a statistical model that can simulate Google jobs and tasks submitted onto a large-scale Google data center. In this section, we first introduce the simulation model, and then evaluate it by comparing the generated workload/hostload to that of the original trace data.

5.1 Statistical Simulation Model of Job/Task Emulation on a Google Data Center

We present the simulation model for emulating Google jobs (tasks) in Figure 10. It can be split into five layers from bottom to top, to perform K-means clustering on applications, statistics analysis on applications, correlation analysis of applications, correlation analysis on jobs/tasks, and simulation of Google jobs/tasks respectively. The output of the lower layers serve as the input of the higher layers.

We describe each layer shown in Figure 10 as follows.

- *K-means clustering analyzer*: This layer is used to generate a group of set centers, which can differentiate applications based on their properties like resource utilization per application. It can help researchers to justify the applications' characteristics individually. Each clustering set corresponds to an application template, based on each of which we can generate a set of application instances, according to the simulation requirements.
- *Statistics analyzer of application types*: This layer is used to specify application types for the simulated application instances. It determines whether an application instance can have multiple tasks and how the tasks are connected per job.

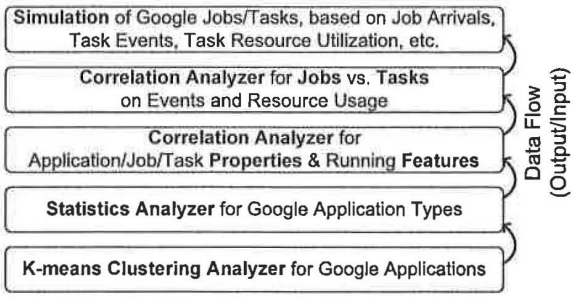


Fig. 10 Simulation Model of Cloud jobs/tasks based on Google trace

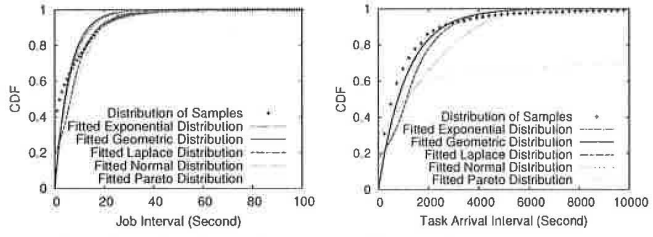
- *Analyzer of correlation between application/job/task properties and their running features*: This layer is used to specify the running features of applications/jobs/tasks, related to how many jobs per application, application's job priority and scheduling class, and so on. At this layer, there will be some job/task templates generated, which can be used to further emulate job/task instances.
- *Analyzer of correlation of jobs vs. tasks*: When emulating a job instance, a set of tasks and task events will be generated beforehand. Each task is associated with a set of requests and utilization on different types of resources like CPU rate and memory size. A task is also associated with a set of events such as evict and fail events. They will be simulated based on the statistics of events over Google trace.
- *Simulation of jobs/tasks*: Finally, we can finish the whole simulation according to the characterization of application properties. Each task simulation is executed in terms of the state transition graph, which can be found in [3]. Various job/task events (including submission, evict, failure, kill, and so on) are generated in accordance with the statistics of the job/task arrival intervals extracted from the one-month Google trace, to be shown in next section. The resource utilization of a simulated task is generated based on the usage statistics (or distribution) of all of the tasks with the same job.

Note that we do not specify job scheduling policy in our simulation model, because of two factors. On one hand, Google trace providers have not disclosed the details about their scheduling policies because of privacy, so this part is a black box for us. On the other hand, in general, the users actually intend to design their own particular scheduling policies for their own simulations to suit various purposes, thus we believe simulating Google jobs/tasks based on Google trace already meets majority of users' needs.

5.2 Evaluation of Simulation Model

In this part, we present the evaluation results based on our simulation model, to confirm its validity. We first make use of maximum likelihood estimate to analyze the probability distribution of job/task arrival interval. Then, we present the simulation effect through the five-layer simulation model, by comparing the properties like workload of the simulated jobs/tasks to those in original trace.

In order to simulate the dynamic arrivals of jobs/tasks, we need to study the job arrival intervals based on Google trace. In Figure 11 (a), we present the distribution of the overall job arrival intervals as well as some well-known probability distribution fitting curves generated via maximum likelihood estimate method. It is observed that the job arrival intervals do not follow any well-known probability distributions explicitly. On the other hand, we show in Figure 11 (b) the distribution of task arrival intervals on a particular Google host. It is observed that the best-fit probability distribution of task arrival intervals on each host is exponential distribution to a certain extent. That is, the task arrival follows a Poisson-similar process [17] in Google trace.



(a) Job Arrival Interval (b) Task Arrival Interval

Fig. 11 Distribution of Job/Task Arrival Interval

In our simulation, we do not adopt any off-the-shelf distributions (e.g., exponential distribution) to emulate the task arrival intervals, because thus would cause skewness as observed. Instead, we randomly select job/task intervals from all of the interval samples listed, such that the generated job/task intervals exactly approach the probability distribution of Google trace.

As follows, we evaluate the simulation effect of our job/task simulation model (Figure 10) through an example. Suppose a user wants to investigate and simulate the jobs/tasks based on particular applications with distinct resource utilization on CPU rate and memory size, and the parameters of K-means clustering algorithm are set as λ=0.5 and μ=20. Then, he/she can simply perform the simulation based on the statistics with our characterization.

We show the K-means clustering results (4 sets classified) based on resource utilization in Figure 12. In each set, we randomly select 20 applications as the basic application templates to generate jobs for further investigation.

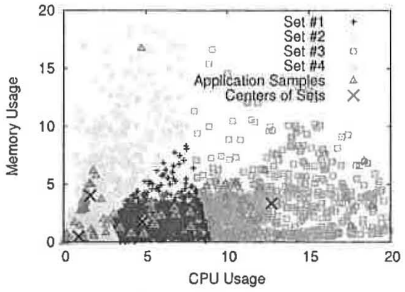
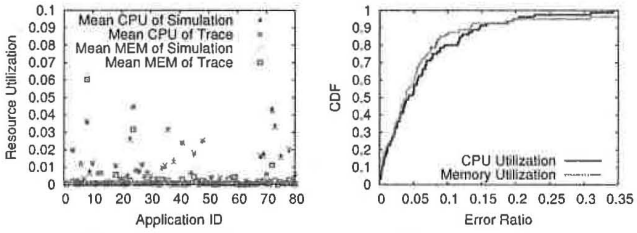


Fig. 12 K-means Clustering Sets and Sampled Applications

As follows, we simulate jobs/task events in accordance with the trace based job/task arrival intervals. Based on the generated jobs/tasks, we further compute the statistics (including average resource utilization per application and hostload values based on tasks' resource utilizations).

We compute the average CPU/memory utilization for each of the 80 sampled applications based on the simulated jobs/tasks, and compare them to that of original jobs/tasks in the trace, as shown in Figure 13. From Figure 13 (a), it is observed that the mean resource utilization of our simulated tasks per application and the statistics in the trace are fairly similar to each other. We also evaluate the simulated resource utilization via error ratio, which is defined as the ratio of the simulation error to the resource utilization of original tasks in the trace, i.e., $\frac{the\ difference\ of\ utilization\ between\ simulation\ \&\ trace}{the\ mean\ resource\ utilization\ in\ the\ trace}$. Through Figure 13 (b), we can observe that 95+% of tasks' simulation errors are less than 20%, w.r.t. the mean CPU and memory utilization per application.

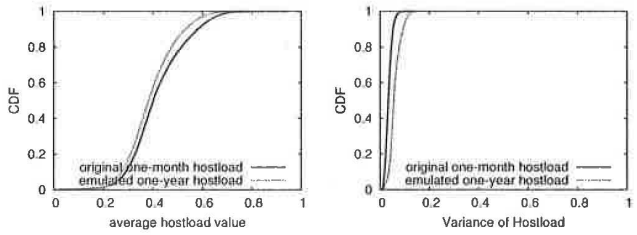


(a) Simulation vs. Trace (b) Error Ratio

Fig. 13 Comparison between Simulation & Trace w.r.t. Application

We also simulate the one-year hostload by summing the emulated tasks' resource utilization on the same host over time, and compare it to the statistics of the original one-month trace in Figure 14. It is observed that the distribution of the hostload values simulated is very close to that of the ones aggregated based on the original trace. With the same CDF values, the average values

of the emulated hostload and original hostload differ within 10%, confirming the accuracy of our simulation. We can also observe that the hostload values summed based on emulated tasks exhibit higher than that of the original trace. This is because the original Google tasks are executed with some constraints and scheduling policies while the emulated tasks are not.



(a) Mean Value (b) Variance

Fig. 14 Evaluation of Hostload Simulation

6 Related Work

Recently, there have been many existing works characterizing Google's trace for the in-depth understanding of the cloud environment. Sharma et al. [18] developed a new metric called Utilization Multiplier (UM) based on task placement constraints and machine properties, in order to precisely characterize the impact of task constraints to the task scheduling delay. They also studied how to synthesize representative task constraints and machine properties, and how to incorporate this synthesis into existing performance benchmarks. Mishra et al. [19] classified Google **task** workload by leveraging some off-the-shelf algorithms like K-means clustering [15], while our K-means clustering algorithm is performed on **application** workload. Zhang [20] characterized the task usage shapes in Google clusters. All of the three works are based on a rather small set of 4-day Google trace that was only used internally by Google Inc..

Since Google publicly released a large set of one-month trace [3,4], more and more researchers have been extensively studying the characteristics of Google cloud environment in different facets. Liu et al. [21] roughly characterized such a trace, including machine population, statistics of daily machine events and job/task events. In comparison to their work, our previous work [5] characterized Google workload and hostload more comprehensively, including various statistical analysis (such as mass-count disparity and Quantile-Quantile plot) about job priority, job/task length, job submission frequency, hostload fluctuation, peak resource usage, job queuing state, etc. We also provided a clear comparison between Google workload and Grid workload. As compared to our previous work, Reiss et al. [12] addressed some new insights

about Google trace. For example, they carefully characterized resource requests vs. resource usage, the relative distribution and discussed challenges in Google task scheduling.

In addition to the research based on Google trace, there are some characterization works based on other cloud systems. Ganapathi et al. [22] adopted a so-called *Kernel Canonical Correlation Analysis (KCCA)* method to model and predict the workload based on Hadoop Distributed File System (HDFS) [23] and map-reduce mechanism [11]. Li et al. [24] proposed a *CloudProphet* framework to predict the application performance, in terms of the non-production trace that is generated from a self-implemented prototype. Jackson et al. [25] provided a performance analysis of HPC applications on the Amazon Web Service platform. They show that network communication is a serious bottleneck for HPC applications when running on widely distributed sites over WAN.

In comparison to all of the research described above, we focus on the static/dynamic features and resource utilization of Google applications. We address at least two new insights about Google applications. On one hand, we characterize the features of application workload/events in terms of various **application types** and via **K-means clustering** with optimized number of sets. On the other hand, we comprehensively analyze the **correlation** between application properties and running features, which can serve as a foundational support to a more precise simulation of cloud benchmark in the long run. Finally, we build a simulation model based on the characterized Google application features, and confirm its accuracy by comparing the statistics between its emulated tasks to the ones in the original trace. Our work significantly advances beyond the simple workload characterization like [5,12,18–21].

To summarize, we compare all of related works to our work in Table 3.

Table 3 Summarization of Related Works

Related Work	Focus and Features	Limitation
Sharma et al. [18]	task placement constraints & machine properties	only 4-day trace
Mishra et al. [19]	classification of Google task workload	only 4-day trace
Zhang [20]	characterization of Google task usage shapes	only 4-day trace
Liu et al. [21]	population, task/macihne events,etc.	a rough analysis
our previous work [5]	compare Google trace and Grid trace	miss app features
Reiss et al. [12]	comprehensive study of Google trace	miss app features
Ganapathi et al. [22]	model and predict workload based on HDFS	miss app features
Li et al. [24]	prediction of app performance	non-real trace
Jackson et al. [25]	HPC performance analysis over AWS	miss app features

7 Conclusion and Future Work

In this paper, we glean some new insights about Google application properties and features based on a one-month Google trace with about 40k applications, and also build a five-layer model for simulating Google jobs/tasks. Some key findings are listed below.

- The number of jobs/events per application follows a typical Pareto principle (joint ratio $\approx$ 10%) and the resource utilization per application follows an extremely typical Pareto principle (joint ratio $<$ 2%).
- All applications can be split into 4 types based on whether they allow batch-task execution mode. There exists a certain correlation between task events and the four application types. For example, about 81.3% of fail task events belong to batch-task applications.
- We also design a K-means clustering algorithm with optimized number of sets to classify applications based on task events and resource utilization (or workload). We observe a Pareto-similar distribution on the number of applications per set.
- There exists a partial correlation between Google application properties and running features. The running states of tasks with low priorities are determined by priority levels. The normal event ratio increases with increasing priorities for low-priority tasks, while that exhibits a rather uniform distribution especially for low-scheduling-class tasks.
- The simulation model built with the above characterized application features can effectively emulate Google jobs/tasks and relative events in accordance with the original Google trace. 95+% of tasks' simulation errors are less than 20%, confirming a high accuracy of our simulation model.

We believe our work is fairly significant especially to the in-depth understanding of the characteristics and behaviors of cloud applications. In the future, we plan to further complete our simulation model by adding various scheduling policies, further simplifying the use of our model for users.

Acknowledgements We thank Google Inc, in particular Charles Reiss and John Wilkes, for making their invaluable trace data available. This work was supported by ANR project Clouds@home (ANR-09-JCJC-0056-01), also in part by the Advanced Scientific Computing Research Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

References

1. Armbrust M, Fox A, Griffith R, Joseph A et al (2009) Above the clouds: A berkeley view of cloud computing. EECS, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28.
2. Vaquero L, Rodero-Merino L, Caceres J, Lindner M (2009) A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39(1):50-55.
3. Wilkes J, More Google cluster data (2011) Google research blog, posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.
4. Reiss C, Wilkes J, Hellerstein J (2012) Google cluster-usage traces: format + schema. Google Inc., Mountain View, CA, USA, Technical Report.

5. Di S, Kondo D, Cirne W (2012) Characterization and comparison of cloud versus grid workloads. IEEE International Conference on Cluster Computing (Cluster'12), pp. 230-238.
6. Meng X, Isci C, Kephart J, Zhang L, Bouillet E, Pendarakis D (2010) Efficient resource provisioning in compute clouds via vm multiplexing. Proceeding of the 7th international conference on Autonomic computing (ICAC'10), New York, NY, USA: ACM, pp. 11-20.
7. Buyya R, Ranjan R, Calheiros R (2010) Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10), pp. 13-31.
8. Stillwell M, Vivien F, Casanova H (2012) Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. Preceedings of IEEE 26th International Conference on Parallel Distributed Processing Symposium (IPDPS'12), pp. 786-797.
9. Calheiros R, Ranjan R, Beloglazov A, De-Rose C, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exper, 41(1), pp. 23-50.
10. Di S, Wang C.-L. (2013) Dynamic optimization of multi-attribute resource allocation in self-organizing clouds. IEEE Transactions on Parallel and Distributed Systems (TPDS), 24(3):464-478.
11. Dean J, Ghemawat S (2004) MapReduce: Simplified data processing on large clusters. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04), pp. 137-150.
12. Reiss C, Tumanov A, Ganger G, Katz R, Kozuch M (2012) Towards understanding heterogeneous clouds at scale: Google trace analysis. Intel science and technology center for cloud computing, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. ISTC–CC–TR–12–101.
13. Feitelson D (2011) Workload Modeling for Computer Systems Performance Evaluation. [Online]. Available: http://www.cs.huji.ac.il/~feit/wlmod/.
14. Koch R (1997) The 80/20 principle: the secret of achieving more with less. Nicholas Brealey.
15. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability, pp. 281-297.
16. Okabe A, Boots B, Sugihara K, Chiu S (2000) Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2nd ed.). ser. Series in Probability and Statistics. John Wiley and Sons, Inc..
17. Ross S (2010), Introduction to Probability Models (10th Edition). Academic Press.
18. Sharma B, Chudnovsky V, Hellerstein J, Rifaat R, Das C (2011) Modeling and synthesizing task placement constraints in google compute clusters. Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11), New York, USA: ACM, 2011, pp. 3:1-3:14.
19. Mishra A, Hellerstein J, Cirne W, Das C.-R. (2010) Towards characterizing cloud backend workloads: insights from Google compute clusters. SIGMETRICS Perform. Eval. Rev., 37(4):pp. 34–41.
20. Zhang Q, Hellerstein J.L., Boutaba R (2011) Characterizing task usage shapes in google compute clusters. Large Scale Distributed Systems and Middleware Workshop (LADIS'11).
21. Liu Z, Cho S (2012) Characterizing machines and workloads on a Google cluster. 8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'12), pp. 397-403.
22. Ganapathi A, Chen Y, Fox A, Katz R.H., Patterson D.A. (2010) Statistics-driven workload modeling for the cloud, ICDE Workshops'10, pp. 87-92.
23. Shvachko K, Kuang H, Radia S, and Chansler R (2010) The hadoop distributed file system. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10), pp. 1-10.
24. Li A, Zong X, Kandula S, Yang X, Zhang M (2011) Cloudprophet: Towards application performance prediction in cloud. ACM SIGCOMM Student Poster, pp. 426-427.
25. Jackson K.R., Ramakrishnan L, Muriki K at al (2010) Performance analysis of high performance computing applications on the amazon web services cloud. Proceedings of the IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom'10). Washington, DC, USA: IEEE Computer Society, pp. 159-168.

26. Hamerly G, Elkan C (2002) Alternatives to the k-means algorithm that find better clusterings. Proceedings of the 17th international conference on Information and knowledge management(CIKM'02), New York, NY, USA: ACM, pp. 600–607.