SLO-aware Colocation of Data Center Tasks Based on Instantaneous Processor Requirements

Pawel Janus
Institute of Informatics, University of Warsaw
Warsaw, Poland
pj320664@students.mimuw.edu.pl

Krzysztof Rzadca
Institute of Informatics, University of Warsaw
Warsaw, Poland
krz@mimuw.edu.pl

ABSTRACT

In a cloud data center, a single physical machine simultaneously executes dozens of highly heterogeneous tasks. Such colocation results in more efficient utilization of machines, but, when tasks' requirements exceed available resources, some of the tasks might be throttled down or preempted. We analyze version 2.1 of the Google cluster trace that shows short-term (1 second) task CPU usage. Contrary to the assumptions taken by many theoretical studies, we demonstrate that the empirical distributions do not follow any single distribution. However, high percentiles of the total processor usage (summed over at least 10 tasks) can be reasonably estimated by the Gaussian distribution. We use this result for a probabilistic fit test, called the Gaussian Percentile Approximation (GPA), for standard bin-packing algorithms. To check whether a new task will fit into a machine, GPA checks whether the resulting distribution's percentile corresponding to the requested service level objective, SLO is still below the machine's capacity. In our simulation experiments, GPA resulted in colocations exceeding the machines' capacity with a frequency similar to the requested SLO.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → *Process management*; • **Computing methodologies** → *Planning and scheduling*;

KEYWORDS

scheduling, resource management, stochastic bin packing

ACM Reference Format:
Pawel Janus and Krzysztof Rzadca. 2017. SLO-aware Colocation of Data Center Tasks Based on Instantaneous Processor Requirements. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 13 pages.
https://doi.org/10.1145/3127479.3132244

1 INTRODUCTION

Quantity or quality? When a cloud provider colocates more tasks on a machine, the infrastructure is used more efficiently. In the short term, the throughput increases. In the longer term, packing more densely reduces future investments in new data centers. However, if the tasks' requirements exceed available resources, some of the tasks might be throttled down or preempted, affecting their execution time or performance.

A standard solution to the quantity-quality dilemma is the Service Level Agreement (SLA): the provider guarantees a certain quality level, quantified by the Service Level Objective (SLO, e.g., a VM with an efficiency of one x86 core of 2Ghz) and then maximizes the quantity. However, rarely is the customer workload using the whole capacity the whole time. The provider has thus a strong incentive to oversubscribe (e.g., to rent 11 single-core virtual machines all colocated on a single 10-core physical CPU), and thus to change the quality contract to a probabilistic one (a VM with a certain efficiency at least 99% of the time).

In this paper, we show how to maintain a probabilistic SLO based on *instantaneous* CPU requirements of tasks from a Google cluster [23]. Previous approaches packed tasks based on the *maximum* or the *average* CPU requirements. The maximum corresponds to no oversubscription, while the average can severely overestimate the quality of service (QoS). Consider the following example with two machines with CPU capacity of 1.0 each and three tasks t_1, t_2, t_3. Tasks t_1 and t_2 are stable with constant CPU usage of 0.5. In contrast, task t_3's CPU usage varies: assume that it is drawn from a uniform distribution over $[0, 1]$. If the resource manager allocates tasks based only on their average CPU usage, t_3, having mean 0.5, can end up packed to a machine shared with either t_1 or t_2; thus, this machine will be overloaded half of the time. An alternative allocation, in which t_1 and t_2 share a machine, and t_3 has a dedicated machine, uses the same number of machines, and has no capacity violations.

To the best of our knowledge, until recently, all publicly available data on tasks' CPU usage in large systems had a very low time resolution. The Standard Workload Format [6] averages CPU usage over the job's entire runtime. The Google cluster trace [23, 31] in versions 1.0 and 2.0 reports CPU usage for tasks (Linux containers) averaged over 5-minute intervals (the `mean CPU usage rate` field). Relying on this field, as our toy example shows, can result in underestimation of the likelihood of overload.

The Google Cluster Trace in version 2.1 extended the `resource usage` table with a new column, the `sampled CPU usage`. The `resource usage` table contains a single record for each 5 minutes runtime of each task. The `sampled CPU usage` field specifies the CPU usage of a task averaged over *a single second* randomly chosen from these 5 minutes. Different tasks on the same machine are not guaranteed to be sampled at the same moment; and, for a task, the sampling moment is not the same in different 5-minute reporting

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SoCC '17, September 24–27, 2017, Santa Clara, CA, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5028-0/17/09…$15.00
https://doi.org/10.1145/3127479.3132244

periods. In contrast, another field, the `mean CPU usage rate`, shows the CPU usage averaged over the whole 5 minutes. Later on, to avoid confusion, we refer to `sampled CPU usage` as the *instantaneous (inst)* CPU usage, and to `mean CPU usage rate` as the *(5-minute) average (avg)* CPU usage.

As we show in this paper, the data on instantaneous CPU usage brings a new perspective on colocation of tasks. First, it shows how variable cloud computing tasks are in shorter time spans. Second, it is one of the few publicly available, realistic data sets for evaluating stochastic bin packing algorithms. The contributions of this paper are as follows:

- The instantaneous usage has a significantly higher variability than the previously used 5-minute averages (Section 3). For longer running tasks, we are able to reconstruct the complete distribution of the requested CPU. We show that tasks' usage do not fit any single distribution. However, we demonstrate that we are able to estimate high percentiles of the total CPU demand when 10 or more tasks are colocated on a single machine.
- We use this observation for a test, called the Gaussian Percentile Approximation (GPA), that checks whether a task will fit into a machine on which other tasks are already allocated (Section 4). Our test uses the central limit theorem to estimate parameters of a Gaussian distribution from means and standard deviations of the instantaneous CPU usage of colocated tasks. Then it compares the machine capacity with a percentile of this distribution corresponding to the requested SLO. According to our simulations (Section 5), colocations produced by GPA have QoS similar to the requested SLO.

The paper is organized as follows. To guide the discussion, we present the assumptions commonly taken by the stochastic bin packing approaches and their relation to data center resource management in Section 2. Section 3 analyzes the new data on the instantaneous CPU usage. Section 4 proposes GPA, a simple bin packing algorithm stemming from this analysis. Section 5 validates GPA by simulation. Section 6 presents related work.

2 PROBLEM DEFINITION

Rather than trying to mimic the complex mix of policies, algorithms and heuristics used by real-world data center resource managers [24, 29], we focus on a minimal algorithmic problem that, in our opinion, models the core goal of a data center resource manager: collocation, or VM consolidation i.e. which tasks should be colocated on a single physical machine and how many physical machines to use. Our model focuses on the crucial quantity/quality dilemma faced by the operator of a datacenter: increased oversubscription results in more efficient utilization of machines, but decreases the quality of service, as it is more probable that machine's resources will turn out to be insufficient. We model this problem as stochastic bin packing i.e., bin packing with stochastically-sized items [13, 18]. We first present the problem as it is defined in [13, 18], then we discuss how appropriate are the typical assumptions taken by theoretical approaches for the data center resource management.

In stochastic bin packing, we are given a set S of n items (tasks) $\{X_1, \ldots, X_n\}$. X_i is a random variable describing task i's resource requirement. We also are given a threshold c on the amount of resources available in each node (the capacity of the bin); and a maximum admissible overflow probability ϱ, corresponding to a Service Level Objective (SLO). The goal is to partition S into a minimal number m of subsets $S_1, \ldots, S_m$, where a subset S_j corresponds to tasks placed on a (physical) machine j. The constraint is that, for each machine, the probability of exceeding its capacity is smaller than the SLO ϱ, i.e., $Pr[\sum_{i:X_i \in S_j} X_i > c] < \varrho$.

Stochastic bin packing assumes that there is *no notion of time*: all tasks are known and ready to be started, thus all tasks should be placed in bins. While resource management in a datacenter typically combines bin packing and scheduling [24, 27, 29], we assume that the schedule is driven by higher-level policy decisions and thus beyond the optimization model. Moreover, even if the schedule can be optimized, eventually the tasks have to be placed on machines using a bin-packing-like approach, so a better bin-packing method would lead to a better overall algorithm.

Stochastic bin packing assumes that the items to pack are *one-dimensional*. Resource usage of tasks in a data center can be characterized by at least four measures [22, 26]: CPU, memory, disk and network bandwidth. One-dimensional packing algorithms can be extended to multiple dimensions by vector packing methods [19, 26].

Stochastic bin packing assumes that tasks' resource requirements are stochastic (random) *variables*, thus they are *time-invariant* (in constrast to stochastic *processes*). The analysis of the previous version of the trace [22] concludes that for most of the tasks the hour to hour ratio of the average CPU usage does not change significantly. This observation corresponds to an intuition that datacenter tasks execute similar workload over longer time periods. Moreover, as the instantaneous usage is just a single 1-second sample from a 5-minute interval, any short term variability cannot be reconstructed from the data. For instance, consider a task with an oscillating CPU usage rising as a linear function from 0 to 1 and then falling with the same slope back to 0. If the period is smaller than the reporting period (5 minutes), the "sampled CPU usage" would show values between 0 and 1, but without any order; thus, such a task would be indistinguishable from a task that draws its CPU requirement from a uniform distribution over $[0, 1]$. We validate the time-invariance assumption in Section 3.4.

To pack tasks, we need information about their sizes. Theoretical approaches commonly assume *clairvoyance*, i.e., perfect information [7, 13, 26, 30]. In clairvoyant stochastic bin packing, while the exact sizes—realizations—are unknown, the distributions X_i are known. We test how sensitive the proposed method is to available information in Section 5.5, where we provide only a limited fraction of measurements to the algorithms. Clearly, a data center resource manager is usually unable to test a task's usage by running it for some time before allocating it. However, a task's usage can be predicted by comparing the task to previously submitted tasks belonging to the same or similar jobs (similarity can be inferred from, e.g., user's and job's name). Our limited clairvoyance simulates varying quality of such predictions. Such prediction is orthogonal to the main results of this paper. We do not rely on user supplied information, such as the declared maximum resource usage, as these are rarely achieved [22]. In contrast to standard stochastic bin packing, our solution does not use the distributions

X_i of items' sizes; it only requires two statistics, the mean $\mu_i(X_i)$ and the standard deviation $\sigma_i(X_i)$.

Algorithms for stochastic bin packing typically assume that the items' distributions $\{X_i\}$ are *independent*. In a data center, a job can be composed of many individual tasks; if these tasks are, e.g., instances of a large-scale web application, their resource requirements can be correlated (because they all follow the same external signal such as the popularity of the website). If correlated tasks are placed on a single machine, estimations of, for instance, the mean usage as the sum of the task's means are inexact. However, in a large system serving many jobs, the probability that a machine executes many tasks from a single job is relatively small (with the exception for data dependency issues [8]). Thus, a simple way to extend an algorithm to handle correlated tasks is not to place them on the same machine (CBP, [28]). While we acknowledge that taking into account correlations is an important direction of future work, the first step is to characterize how frequent they are; and analyses of the version 2.0 of the trace [11, 22] did not consider this topic.

While a typical data center executes tasks of different importance (from critical production jobs to best-effort experiments), stochastic bin packing assumes that all tasks have the same priority/importance. Different priorities can be modeled as different requested SLOs; simultaneously guaranteeing various SLOs for various groups of colocated tasks is an interesting direction of future work.

We also assume that all machines have equal capacities (although we test the impact of different capacities in Section 5.4).

Finally, we assume that exceeding the machine's capacity is undesirable, but not catastrophic. Resources we consider are rate-limited, such as CPU, or disk/network bandwidth, rather than value-limited (such as RAM). If there is insufficient CPU, some tasks slow down; on the other hand, insufficient RAM may lead to immediate preemption or even termination of some tasks.

3 CHARACTERIZATION OF INSTANTANEOUS CPU USAGE

In this section we analyze `sampled CPU usage`, which we call the instantaneous (inst) CPU usage, introduced in version 2.1 of the Google trace [23]. We refer to [11, 22] for the analysis of the previous version of the dataset.

We use the following notation. We denote by T_i the number of records about task i in the `resource usage` table (thus, effectively, task's i duration as counted by 5-minute intervals). We denote the t-th value of task i *instantaneous (inst) usage* as $x_i(t)$; and the t-th value of task i *5-minute average usage* as $y_i(t)$. We reserve the term *mean* for a value of a statistic $\bar{x_i}$ computed from a (sub)sample, e.g., for the whole duration $(x_i(1), \ldots, x_i(T_i))$, $\bar{x_i} = \frac{1}{T_i}\sum_{t=1}^{T_i} x_i(t)$. We denote by X_i the empirical distribution generated from $(x_i(1), \ldots, x_i(T_i))$.

3.1 Data preprocessing

We first discard all failing tasks as our goal is to characterize a task's resource requirements during its complete execution (in our future work we plan to take into account also the resource requirements of these failing tasks). We define task as failing if it contains at least one of `EVICT(2)`, `FAIL(3)`, `KILL(5)`, `LOST(6)` events in the events table. We then discard 209 940 tasks (1.2% of all tasks in the trace) that show zero instantaneous usage for their entire duration: these tasks correspond to measurement errors, or truly non-CPU dependent tasks, which have thus no impact on the CPU packing we want to study.

We replace 13 records of average CPU usage higher than 1 by the corresponding instantaneous usage (no instantaneous usage records were higher than 1). The trace normalizes both values to 1 (the highest total node CPU capacity in the cluster). Thus, values higher than 1 correspond to measurement errors (note that these 13 records represent a marginal portion of the dataset).

Task lengths differ: 16 055 428 (95% of all) tasks are shorter than 2 hours and thus have less than 24 CPU measurements. We partition the tasks into two subsets, the *long* tasks (2 hours or longer) and the remaining *short* tasks. We analyze only the *long* tasks. We do not consider the short tasks, as, first, they account for less than 10% of the overall utilization of the cluster [22], and, second, the shorter the task, the less measurements we have and thus the less reliable is the empirical distribution of the instantaneous usage (see Section 3.2).

Finally, some of our results (normality tests, percentile predictions, experiments in Section 5) rely on repeated sampling of instantaneous and average CPU usage of tasks. For such experiments, we generate a random sample of $N = 100\,000$ long tasks. For each task from the sample, we generate and store $R = 10\,000$ realizations of both instantaneous and average CPU usage. The instantaneous realizations are generated as follows (averages are generated analogously). From $(x_i(1), \ldots, x_i(T_i))$, we create an empirical distribution (following our assumption of time invariance). We then generate R realizations of a random variable from this distribution. Such representation allows us to have CPU usage samples of equal length independent of the actual duration T_i of the task. Moreover, computing statistics of the total CPU usage with such long samples is straightforward: e.g., to get samples of the total CPU usage for 3 tasks colocated on a single node, it is sufficient to add these tasks' sampled instantaneous CPU usage, i.e., to add 3 vectors, each of 10 000 elements. Our data is available at http://mimuw.edu.pl/~krzadca/sla-colocation/.

3.2 Validation of Instantaneous Sampling

We start by evaluating whether tasks' instantaneous samples are representative of their true usage, i.e., whether the method used to produce instantaneous data was unbiased. While we don't know the true usage, we have an independent measure, the 5-minute averages. Our hypothesis is that the mean of the instantaneous samples should converge to the mean of the 5-minute average samples. Figure 1 shows the distribution of the relative difference of means as a function of the number of samples. For a task i we compute the mean of the average CPU usage $\bar{y_i} = \frac{1}{T_i}\sum_t y_i(t)$ (taking into account all measurements $y_i(t)$ during the whole duration of the task). We then compute the mean of a given number k of instantaneous CPU usage $\bar{x_i}^{(k)} = \frac{1}{k}\sum_{t\in S_k} x_i(t)$ ($k \in \{1, 2, 5, \ldots, 500, 1\,000\}$, S_k is a randomly chosen subset of $\{1, \ldots, T_i\}$ of size k). For each k independently, we compute the statistics over all tasks having at least k records: thus $k = 1$ shows a statistics over all *long* tasks, and $k = 1\,000$ over tasks longer than 83 hours.

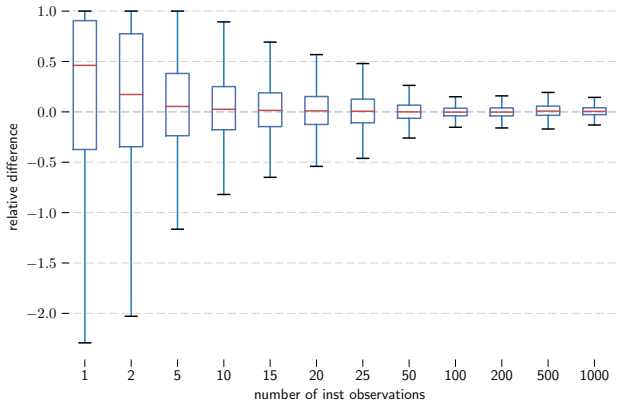


Figure 1: Distributions of the relative differences $((\bar{y_i} - \bar{x_i}^{(k)})/\bar{y_i})$ between the means computed from 5-minute average y_i and instantaneous x_i CPU usage as a function of the number of instantaneous samples k for all tasks at least 2 hours long. Here, and in the remaining boxplots, the line inside the box denotes the median; the box spans between the first and the third quartile (the interquartile range, IQR); and the whiskers extend to the most extreme data point within 1.5 × IQR.

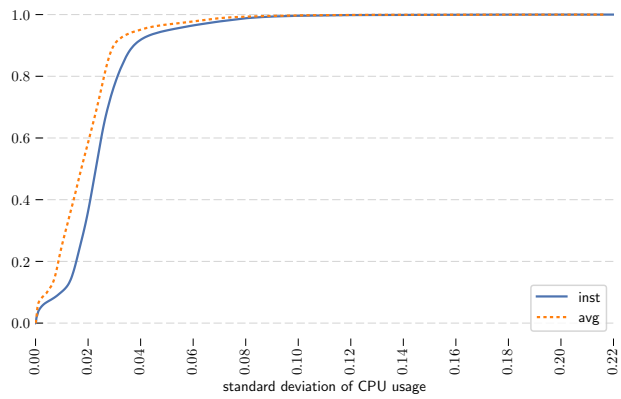


Figure 2: CDF of the distribution of standard deviations of CPU usage for all tasks at least 2 hours long.

The figure shows that, in general, the method used to obtain the instantaneous data is unbiased. From approx. 15 samples onwards, the interquartile range, IQR, is symmetric around 0. The more samples, the smaller is the variability of the relative difference, thus, the closer are means computed from the instantaneous and the average data.

3.3 Variability of instantaneous and of average usage

Next, we characterize the variability of the instantaneous usage as characterized by standard deviations of instantaneous $\sigma_{inst}(i)$ and average $\sigma_{avg}(i)$ usage. Figure 2 shows the CDFs of the standard deviations across all long tasks from the trace. Instantaneous usage is more variable than 5-minute averages. Furthermore, as Figure 3 shows, standard deviations depend on the mean CPU usage: the higher task's mean CPU usage, the higher its standard deviation: compared to the avg trend line (linear regression) the inst trend line has both steeper slope (0.36 vs. 0.31) and higher intercept (0.015 vs. 0.010).

3.4 Time invariance

We now test our assumption that the instantaneous loads are drawn from a random distribution that does not depend on time. For each of the long tasks, we divide observations into windows of consecutive $\Delta = 12$ records (a window corresponds to 1 hour): a single window is thus $(x_i(k\Delta + 1), x_i(k\Delta + 2), \ldots, x_i(k\Delta + \Delta - 1))$ (where k is a non-negative integer). We then compare the distributions of two windows picked randomly (for two different k values, k_1 and k_2; k_1 and k_2 differ between tasks). Our null hypothesis is that these samples are generated by the same distribution. In contrast, if there is a stochastic *process* generating the data (corresponding to, e.g., daily or weekly usage patterns), with high probability the two distributions would differ (for a daily usage pattern, assuming a long-running task, the probability of picking two hours 24-hours apart is 1/24).

To validate the hypothesis, we perform a Kolmogorov-Smirnov test. For roughly 30% of tasks the test rejects our hypothesis at the significance level of 5% (the results for $\Delta = 24$ and $\Delta = 36$ are similar). Thus for roughly 30% of tasks the characteristics of the instantaneous CPU usage changes in time. On the other hand, the analysis of the average CPU usage [22] shows that the hour-to-hour variability of individual tasks is small (for roughly 60% of tasks weighted by their duration, the CPU utilization changes by less than 15%). We will further investigate these changing tasks in future work.

3.5 Variability of individual tasks

Many theoretical approaches (e.g., [13, 30, 33]) assume that items' sizes all follow a specific, single distribution (Gaussian, exponential, etc.). In contrast, we discovered that the distributions of instantaneous loads in the Google trace vary significantly among tasks.

To characterize the common types of distributions of roughly 800 000 long tasks, we clustered tasks' empirical distributions. Our method is the following. First, we generate histograms representing the distributions. We set the granularity of the histogram to 0.01. Let h_i be the histogram of task i, a 100-dimensional vector. $h_i[k]$ (with $0 \le k \le 99$) is the likelihood that an instantaneous usage sample falls between $k/100$ and $(k + 1)/100$, $h_i[k] = |\{x_i(t) : k/100 \le x_i(t) < (k + 1)/100\}|/T_i$.

Then, we use the k-means algorithm [15] with the Euclidean distance metric on the set of histograms $\{h_i\}$. The clustering algorithm treats each task as a 100-dimensional vector. To compute how different tasks i and j are, the algorithm computes the Euclidean distance between h_i and h_j. Typically k-means is not considered an algorithm robust enough for handling high-dimensional data. However, a great majority of our histograms are 0 beyond 0.30, thus the data has effectively roughly 30 significant dimensions. After

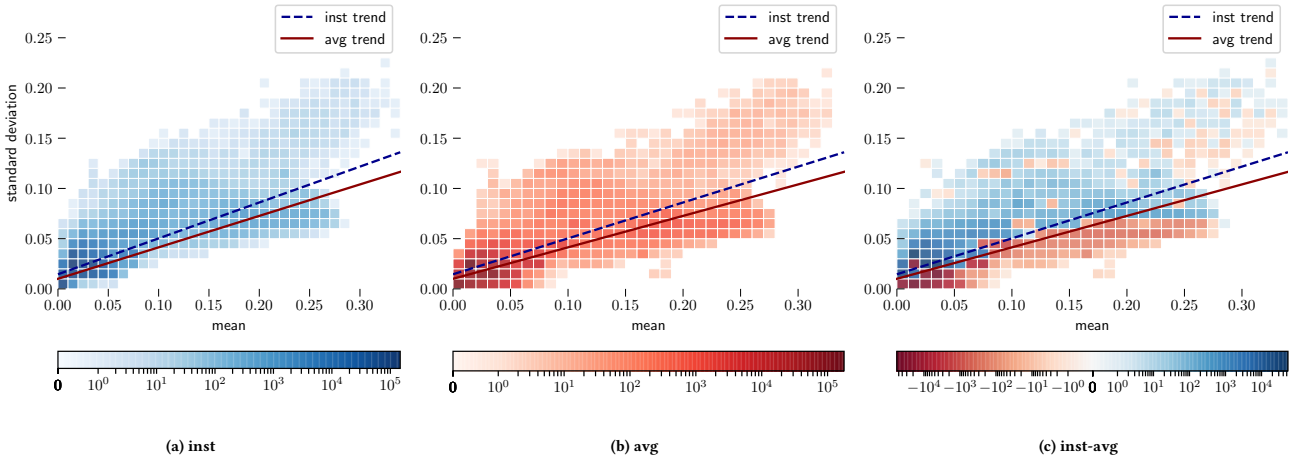


Figure 3: Heatmaps showing standard deviations of CPU usage as a function of means for long tasks. Figure (c) highlights the differences between (a) and (b): blue areas correspond to (mean × std) parameters matching more inst than avg samples.

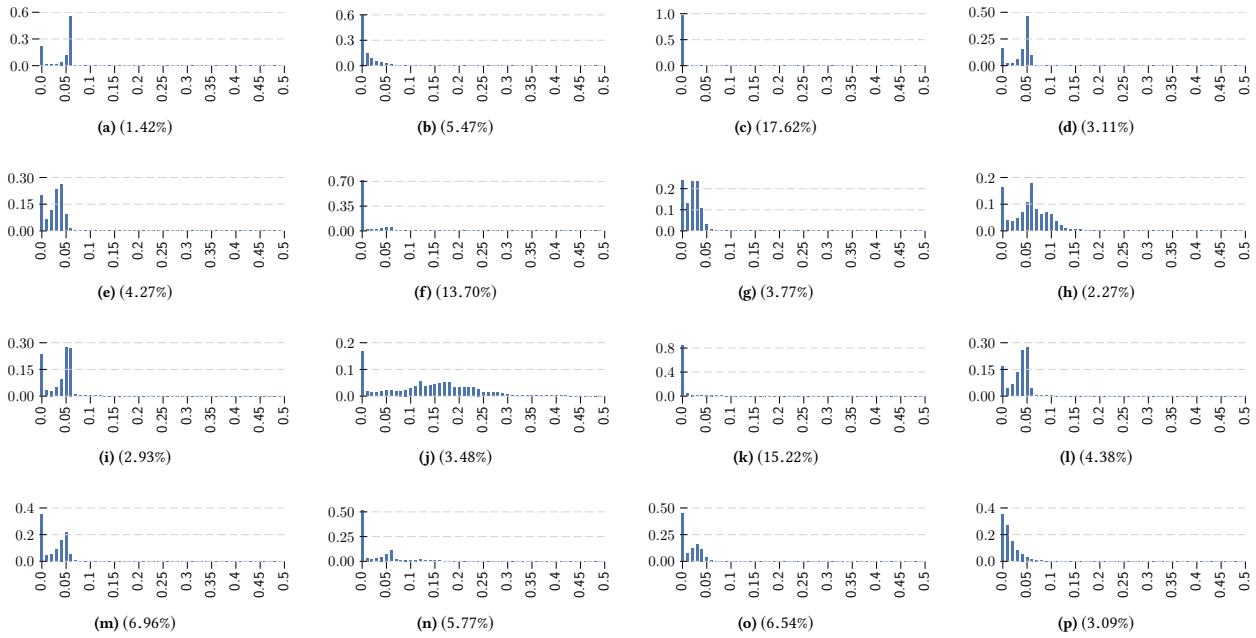
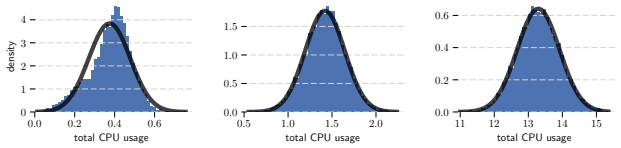


Figure 4: Typical distributions of task's instantaneous CPU usage. Each sub-figure corresponds to a center of one of the 16 clusters produced by the k-means clustering algorithm. X—instantaneous usage (cut to $[0, 0.50]$); Y—% share of occurrences (ranges differ between plots).

a number of initial experiments, we set $k = 16$ clusters, as larger k produced centroids similar to each other, while smaller k mixed classes that are distinct for $k = 16$. Figure 4 shows clusters' centroids, i.e., the average distribution from all the tasks assigned to a single cluster by the k-means algorithm.

First, although number of tasks in a cluster varies considerably (from 1.4% to 17.6% of all long tasks), no cluster strictly dominates the data, as the largest cluster groups less than 1/5th of all the tasks. Second, the centroids vary significantly. Some of the centroids correspond to simple distributions, e.g., tasks that almost always have 0 CPU usage (c), (f), (k); or exponential-like distributions (b) and (p); while others correspond to mixed distributions (h), (j), (m). Both observations demonstrate that no single probability distribution can describe all tasks in the trace. Consequently, this



(a) $N = 10$ (b) $N = 50$ (c) $N = 500$

Figure 5: Example empirical distributions of the total instantaneous CPU usage summed over randomly-chosen samples of $N = 10$, $N = 50$ and $N = 500$ tasks. The black line denotes a fitted Gaussian distribution. Note the different ranges of X and Y axes.

data set does not satisfy the assumption that tasks follow a single distribution.

3.6 Characterizing the total usage

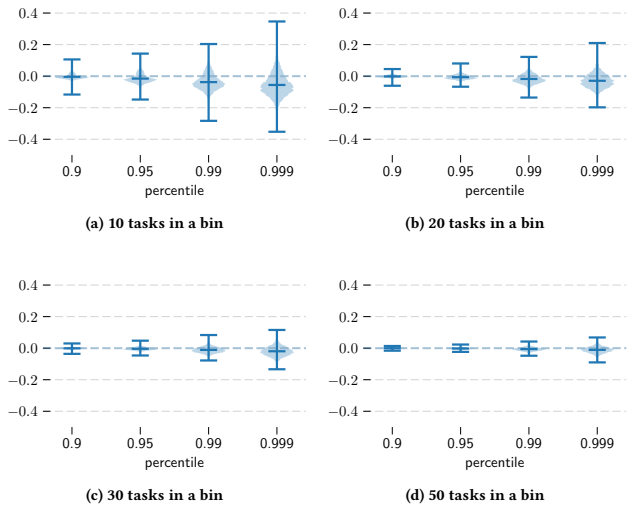
The previous section demonstrated that individual tasks' usage distributions are varied. However, the scheduler is more concerned with the *total* CPU demand of the tasks colocated on a single physical machine. The central limit theorem states that the sum of random variables tends towards the Gaussian distribution (under some conditions on the variables, such as Lyapunov's condition). The question is whether the tasks in the Google trace are small enough so that, once the Gaussian approximation becomes valid, their total usage is still below the capacity of the machine.

To test this hypothesis, from our set of 100 000 tasks (Section 3.1), we take random samples S_j of $N = \{10, 20, \ldots, 500\}$ tasks (repeating the sampling 10 000 times for each size). For each sample, we calculate the resulting empirical distribution based on 10 000 realizations of the instantaneous CPU usage. Additionally, for each sample S_j, we fit a Gaussian distribution $N(\mu_j, \sigma_j)$. We use standard statistics over the tasks $i \in S_j$ to estimate parameters: $\mu_j = \sum_{i \in S_j} \mu_i$ and $\sigma_j = (\sum_{i \in S_j} \sigma_i^2)^{\frac{1}{2}}$, where μ_i is the mean usage of task i, $\mu_i = \bar{x_i}$, and σ_i its standard deviation. Figure 5 shows empirical distribution for three randomly-chosen samples and the fitted Gaussian distributions.

As the empirical distributions resemble the Gaussian distribution, we used the Anderson-Darling (A-D) test to check whether the resulting cumulative distribution is *not* Gaussian (the A-D test assumes as the null hypothesis that the data is normally distributed with unknown mean and standard deviation). Table 1 shows aggregated results, i.e., fraction of samples of a given size N for which the A-D test *rejects* the null hypothesis at significance level of 5%. A-D rejection rates for smaller samples (10-100 tasks) are high, thus the distributions are not Gaussian. For instance assume that $N = 50$ tasks are collocated on a single machine; if they are chosen randomly, in 78% of cases the resulting distributions of total instantaneous CPU usage is not Gaussian according to the A-D test. On the other hand, for 500 tasks, although A-D rejection rate is roughly 9%, the mean cumulative usage is 13.5, i.e., 13.5 times larger than the capacity of the largest machine in the cluster from which the trace was gathered.

N	**10**	**20**	**30**	**50**	**100**	**250**	**500**
AD	0.991	0.965	0.923	0.784	0.493	0.183	0.087
$\bar{\mu_j}$	0.27	0.55	0.82	1.37	2.74	6.85	13.69

Table 1: H0 rejection rates (middle row) for the normality of the total instantaneous CPU usage by the number of tasks in a sample. Anderson-Darling test at significance level of 5%. The bottom row shows mean (over all samples) μ_j, the estimated mean of the CPU usage for the given number of tasks.



(a) 10 tasks in a bin (b) 20 tasks in a bin

(c) 30 tasks in a bin (d) 50 tasks in a bin

Figure 6: Relative differences $(F^{-1}_{\mu_j,\sigma_j}(k) - P_k)/P_k$ between the k=95th, k=99th and the k=99.9th percentiles of the inverse CDF F^{-1} of the normal distribution and the corresponding values of the empirical distribution P_k. Violin plots with outlines showing a (slightly smoothed) histogram and whiskers—the distributions of the differences. Each violin shows a statistics over 10 000 independent samples.

However, to solve the packing problem, we need a weaker hypothesis. Rather than trying to predict the whole distribution, we only need the probability ϱ of exceeding the machine capacity c; this probability corresponds to the value of the survival function $(1 - CDF)$ in a specific point c. As our main positive result we show that it is possible to predict the value of the empirical survival function with a Gaussian usage estimation. The following analysis does not yet take into account the machine capacity c; here we are generating a random sample S_j of 10 to 50 tasks and analyze their total usage. In the next section we show an algorithm that takes into account the capacity.

As ϱ corresponds to the requested SLO, typically its values are small, e.g., 0.1, 0.01 or 0.001; these values correspond to the $k = 90$th, $k = 99$th or $k = 99.9$th percentiles of the distribution. The question is thus how robust is the prediction of such a high percentile.

To measure the robustness, we compute the estimated usage at the kth percentile as the value of the inverse distribution function

Algorithm 1: GPA algorithm: find the first machine j to which task t fits.

Notation:
$\mathtt{F}(x|\mu, \sigma^2)$—the value of the CDF of the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ in a point x

```
1  FitBin(k, j, ρ)
2  |  μ'_j = μ_k + Σ_{i∈S_j} μ_i;
3  |  σ'_j = (σ_k^2 + Σ_{i∈S_j} σ_i^2)^{1/2};
4  |  return ρ − (1 − F(c|μ'_j, (σ'_j)^2));
5  FindBin(k, ρ)
6  |  for j in 1..m do
7  |  |  if FitBin(k, j, ρ) ≥ 0 then
8  |  |  |  return j;
9  |  m ← m + 1;
10 |  return m;
```

$F^{-1}_{\mu_j,\sigma_j}$ of the fitted Gaussian distribution $N(\mu_j, \sigma_j)$ at the requested percentile k. We compare $F^{-1}_{\mu_j,\sigma_j}(k)$ with P_k, the k-th percentile of the empirical distribution. Figure 6 shows $(F^{-1}_{\mu_j,\sigma_j}(k) - P_k)/P_k$, i.e., the relative difference between the Gaussian-based estimation and the empirical percentile. We see that, first, medians are close to 0, which means that the Gaussian-based estimation of the total usage is generally accurate. Second, the Gaussian-based estimation underestimates rare events, i.e., it underestimates the usage for high percentiles. Third, if there are more tasks, the variance of the difference is smaller.

4 STOCHASTIC BIN PACKING WITH GAUSSIAN PERCENTILE APPROXIMATION (GPA)

The main positive result from the previous section is that a Gaussian estimation estimates values of high percentiles of the total instantaneous CPU usage. In this section, we formalize this observation into GPA, a fit test that uses the central limit theorem to drive a stochastic bin packing algorithm. GPA stems from statistical multiplexing, a widely-used approach in telecommunications, in which individual transmissions, each with varying bandwidth requirements, are to be packed onto a communication channel of a fixed capacity (although our models, following [13, 18], do not consider packet buffering, making the models considerably easier to tackle). A related test, although assuming that the packed items all have Gaussian distributions, was proposed in [30] for multiplexing VM bandwidth demands.

A standard bin-packing algorithm (such as First Fit, Best Fit, etc.) packs items $\{x_i\}$ to bins S_j sequentially. For instance, the First Fit algorithm, for each item x_k, finds the minimal bin index j, such that x_k fits in the bin S_j. The fitting criterion is simply that the sum of the sizes of items S_j already packed in j and the current item x_k is smaller than the bin capacity c, $x_k + \sum_{x_i \in S_j} x_i \leq c$.

Our method, the Gaussian Percentile Approximation (GPA, 1) replaces the fitting criterion with an analysis of the estimated Gaussian distribution. For each open (i.e., with at least one task) machine j, we store the current estimation of the mean μ_j and of the standard deviation σ_j. We use standard statistics over the tasks $i \in S_j$ to estimate these values: $\mu_j = \sum_{i \in S_j} \mu_i$ and $\sigma_j = (\sum_{i \in S_j} \sigma_i^2)^{\frac{1}{2}}$. When deciding whether to add a task k to a machine j, we recompute the statistics taking into account task k's mean μ_k and standard deviation σ_k: $\mu'_j = \mu_j + \mu_k$; $\sigma'_j = (\sigma_k^2 + \sum_{i \in S_j} \sigma_i^2)^{\frac{1}{2}}$. The task k fits in the machine j if and only if the probability that the total usage exceeds c is smaller than ϱ. We use the CDF of the Normal distribution $\mathtt{F}_{\mu'_j,\sigma'_j}$ to estimate this probability, i.e., a task fits in the bin if and only if $\mathtt{F}_{\mu'_j,\sigma'_j}(c) \geq 1 - \varrho$.

Algorithm 1 shows First Fit with GPA. Best Fit can be extended analogously: instead of returning the first bin for which `FitBin` ≥ 0, Best Fit chooses a bin that results in minimal among positive `FitBin` results.

As both First Fit and Best Fit are greedy, usually the last open machine ends up being underutilized. Thus, after the packing algorithm finishes, to decrease the probability of overload in the other bins, we rebalance the loads of the machines by a simple heuristics. Following the round robin strategy, we choose a machine from $\{1, \ldots, m-1\}$, i.e., all but the last machine. Then we try to migrate its first task to the last machine m: such migration fails if the task does not fit into m. The algorithm continues until $max\_failures$ failed attempts (we used $max\_failures = 5$ in our experiments). Note that many more advanced strategies are possible. Any such rebalancing makes the algorithm not on-line as it reconsiders the previously taken decisions (in contrast to First Fit or Best Fit, which are fully on-line).

5 VALIDATION OF GPA THROUGH SIMULATION EXPERIMENTS

The goal of our experiments is to check whether GPA provides empirical QoS similar to the requested SLO while using a small number of machines.

5.1 Method

Our evaluation relies on Monte Carlo methods. As input data, we used our random sample of $N = 100\,000$ tasks, each having $R = 10\,000$ realizations of the instantaneous and the 5-minute average loads generated from empirical distributions (see Section 3.1). To observe algorithms' average case behavior we further sub-sample these $N = 100\,000$ tasks into 50 *instances* each of $N' = 1\,000$ tasks. A single instance can be thus compactly described by two matrices $x[i][t]$ and $y[i][t]$, where x denotes the instantaneous and y the 5-minute average usage; $i \in \{1, \ldots, N'\}$ is the index of the task and $t \in \{1, \ldots, R\}$ is the index of the realization.

Many existing theoretical approaches to bin-packing implicitly assume clairvoyance, i.e., the sizes of the items are known in advance. We test the impact of this assumption by partitioning the matrices' columns into the observation and the evaluation sets. The bin-packing decision is based only on the data from the observation set O, while to evaluate the quality of the packing we use the data from the evaluation set E (O and E partition R, i.e., $O \cap E = \emptyset$ except in the fully clairvoyant scenario, in which $O = E = R$). For instance, we might assume we are able to observe each tasks' 100 instantaneous usage samples before deciding where to pack it: this corresponds to the observation set $O = \{1, \ldots, 100\}$, i.e., $x[i][1 \ldots 100]$, and the evaluation $E = \{101 \ldots 10\,000\}$, set of $x[i][101 \ldots 10\,000]$. In this case our algorithms will compute statistics based on $x[i][1 \ldots 100]$

(e.g., the Gaussian Percentile Approximation will compute the mean μ_i as $\mu_i = \frac{1}{100}\sum_{t=1}^{100} x[i][t]$). As we argued in Section 2, scenarios with limited clairvoyance simulate varying quality of prediction of the resource manager. An observation set equal to the evaluation set corresponds to clairvoyance, or the perfect estimations of the usage; smaller observation sets correspond to worse estimations.

We execute GPA with four different target SLO levels, $\varrho = 0.10$ corresponding to the SLO of 90%; $\varrho = 0.05$ (SLO of 95%); $\varrho = 0.01$ (SLO 99%); and $\varrho = 0.001$ (SLO 99.9%).

We compare GPA with the following estimation methods:

- **Cantelli** (proposed for the data center resource management in [16]): items with sizes equal to the mean increased by b times the standard deviation. According to Cantelli's inequality, for any distribution $Pr[X > \mu + b\sigma] < \frac{1}{1+b^2}$. Thus $b = 4.4$ ensures SLO of 95%. If X has a Normal distribution (which is not the case for this data, Figure 4), the multiplier can be decreased to $b = 1.7$ with keeping SLOs at the same level [16]. In initial experiments, we found those values to be too conservative and decided to also consider an arbitrary multiplier $b = 1.0$.
- **av** : Items with sizes proportional to the *mean* μ_i from the observation period. The mean is multiplied by a factor $f \in \{1.0, 1.25, 2.0\}$. Factors larger than 1.0 leave some margin when items' realized size is larger than its mean.
- **perc** : Items with sizes equal to a certain percentile from the observation period. We use the $\{50, 70, 90, 95, 99, 100\}$th percentile. The *maximum* (100th percentile) corresponds to a conservative policy that packs items by their true maximums—this policy in fully clairvoyant scenario is essentially packing tasks by their observed maximal CPU consumption. Lower percentiles correspond to increasingly aggressive policies.

All these methods use either First Fit or Best Fit as the bin packing algorithm. We analyze the differences between the two in Section 5.2. All use rebalancing; we analyze its impact in Section 5.6.

We use two metrics to evaluate the quality of the algorithm. The first metric is the number of used machines (opened bins). This metric directly corresponds to the resource owner's goal—using as few resources as possible. Different instances might have vastly different usage, resulting in different number of required machines. Thus, for a meaningful comparison, for each instance we report a *normalized* number of machines m. We compute m as $m = m_{\text{abs}}/m_{\text{norm}}$, i.e., m_{abs}, the number of machines used by the packing algorithm, divided by a normalization value m_{norm}. The normalization value computes the total average CPU requirements in the instance, and then divides it by machine capacity, $m_{\text{norm}} = \lceil \frac{1}{c}\sum_i \bar{x_i} \rceil$.

The second metric is the measured frequency q of exceeding the machine capacity c: the higher the q, the more often the machine is overloaded. $(1-q)$ corresponds to the observed (empirical) QoS. We compute q by counting $q(j)$: independently for each machine j, how many of E realizations of the total instantaneous usage resulted in total machine usage higher than c: $q(j) = \sum_{t\in E}\langle(\sum_{i\in S_j} x[i][t]) > c\rangle$, where $\langle pred\rangle$ returns 1 if the predicate $pred$ is true. We then average these values over all m machines and the complete evaluation period E, $q = \frac{1}{m|E|}\sum_{j=1}^{m} q(j)$.

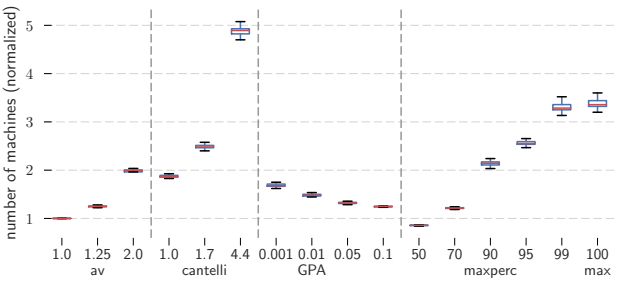


Figure 7: Number of machines (normalized to the lower bound) by different estimation algorithms. Clairvoyance, $c = 1$. Here and in the remaining boxplots, the statistics for each box are computed over 50 instances.

The base case for the experiments is the full clairvoyance (i.e., observation set equal to the evaluation set, $O = E = R$); machine capacity $c = 1$; estimation based on instantaneous (*inst*) data. The following sections test the impact of these assumptions.

5.2 Comparison between algorithms

Both FirstFit and the BestFit algorithms lead to similar outcomes. The number of machines used m_{abs} differed by at most 1. The mean values for q differed by less than 1%. Consequently, to improve presentation, we report data just for BestFit.

The tasks in the trace have small CPU requirements, as already shown in Figure 5, which reported for 500 tasks the mean total requirement of 13.5. *av 1*, packing tasks by their mean requirements, confirms this result, packing 1000-task instances into, on the average, $\bar{m}_{\text{abs}} = 28.0$ machines.

The *Cantelli* estimation is very conservative (for $b = 1.0$, mean $\bar{q} = 3 \times 10^{-4}$; for $b = 1.7$, mean $\bar{q} = 2 \times 10^{-6}$). The *max* estimation never exceeds capacity; and *perc* with high percentiles is similarly conservative: 99th percentile leads to $\bar{q} = 2 \times 10^{-8}$; the 95th to $\bar{q} = 5{\times}10^{-6}$; and the 90th to $\bar{q} = 7{\times}10^{-5}$. The resulting packings use significantly more machines than *av* and *GPA* estimations (Figure 7).

Figure 8 shows the normalized number of machines m and the empirical capacity violations q for the remaining algorithms (*GPA*, *av* and *perc*). No estimation Pareto-dominates others—different methods result in different machine-QoS trade-offs. The resulting (m, q) can be roughly placed on a line, showing that q decreases exponentially with an increase in the number of machines, *m*. *perc 70*, *av 1.25* and *GPA 0.1* result in comparable *m-q*; and, to somewhat smaller degree, *perc 90*, *av 2.0* and *GPA 0.001*. Such similarities might suggest that to achieve a desired q, it is sufficient to use *av* with an appropriate value of the multiplier. We test this claim in Section 5.4.

Figure 9 analyses for GPA the relative error between the measured frequency of capacity violations q and the requested SLO ϱ, i.e.: $(q - \varrho)/\varrho$. The largest relative error is for the smallest target $\varrho = 10^{-3}$: GPA produces packings with the measured frequency of capacity violations of $q = 1.6 \times 10^{-3}$, an increase of 60%. This result follows from the results on random samples (Figure 6): estimating the sum from the Gaussian distribution underestimates rare

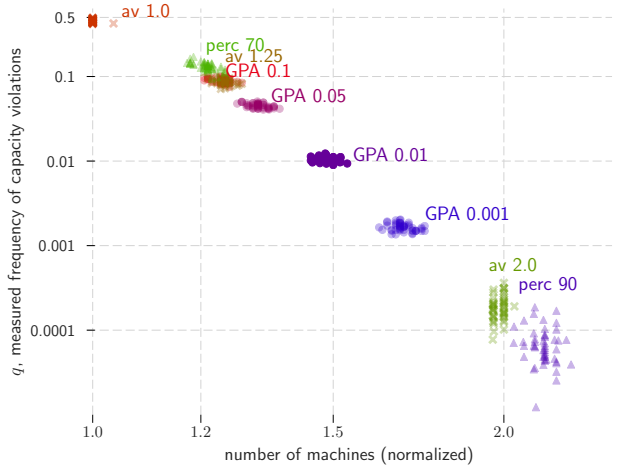


Figure 8: Comparison of the number of used machines (X axis, normalized to the lower bound) and the empirical frequency of capacity violations q (Y axis) between *GPA*, *av* and *perc*. Each dot represents a single instance. Clairvoyance, $c = 1$

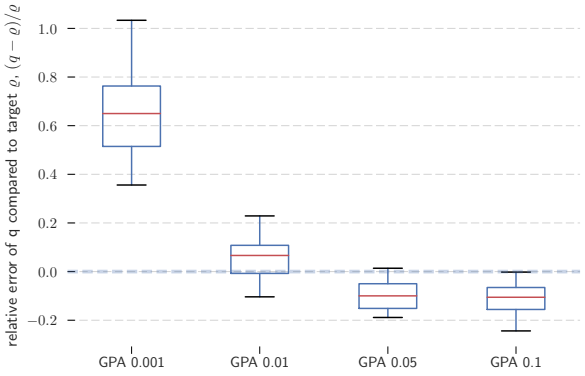


Figure 9: Relative error $(q-\varrho)/\varrho$ of *GPA* for various requested SLO ϱ values. Clairvoyance, $c = 1$.

events. As a consequence, for SLOs of 99th or 99.9th percentile, the ϱ parameter of GPA should be adjusted to a smaller value.

5.3 Observation of the 5-minute average usage

In this series of experiments we show that if algorithms use statistics over 5-minute averages (the low-frequency data), the resulting packing has more capacity violations. Estimations that use statistics of tasks' variability (such as the standard deviation in GPA and Cantelli) are more sensitive to less accurate avg data. This is not a surprise: as we demonstrated in Section 3, the averages report smaller variability than instantaneous usage.

Figure 10 summarizes q for various estimation methods. The figure does not show Cantelli with $b = 4.4$, as on both datasets the mean $\bar{q}$ is 0. Similarly, for *max* (*perc 100*) estimation, the mean

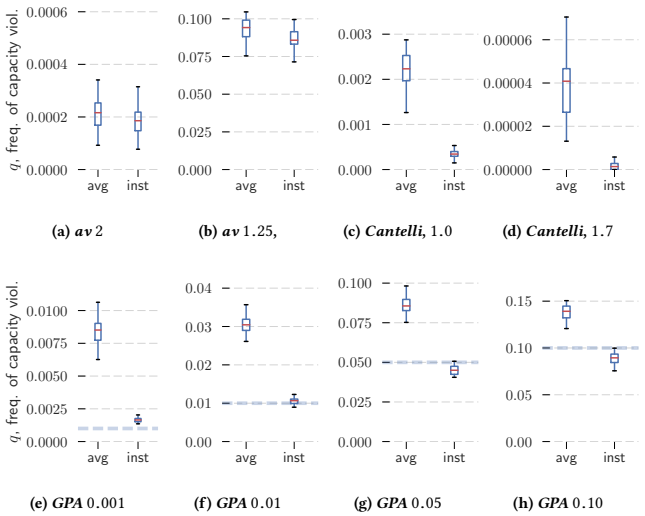


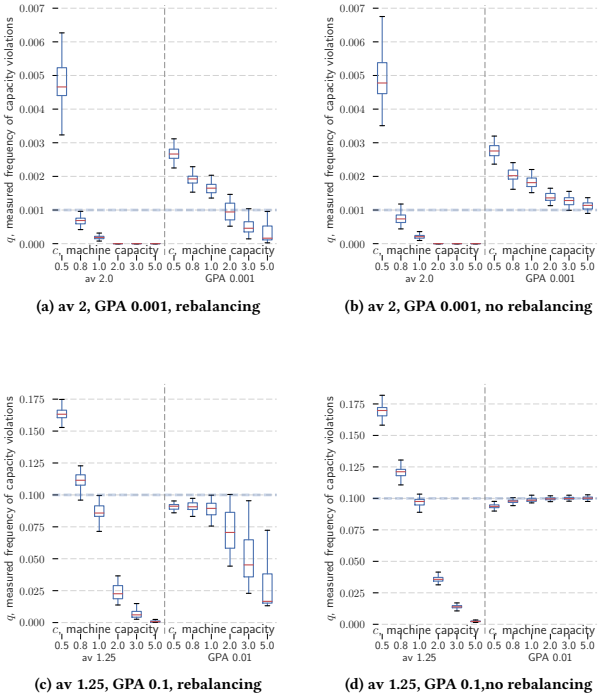
Figure 10: Comparison of the measured frequency of capacity violations q when each algorithm uses either the 5-minute averages (*avg*), or 1-second instantaneous (*inst*) data. Clairvoyance, $c = 1$. For GPA, the target SLO is marked by a thicker line. Note that Y scales differ (see the discussion in Section 5.2 for comparison between these algorithms).

$\bar{q}$ using 5-minute averages (*avg*) is very small (albeit non-zero, in contrast to *inst*): 3×10^{-7} for $e = 0.8$ and 5×10^{-5} for bin size multiplier $e = 1.0$. High frequency *inst* data significantly reduces the number of capacity violations for estimations that use the standard deviation. For Cantelli, the improvement is roughly 10 times; for GPA roughly 2-3 times. In contrast, as expected, *av* estimation has similar q for both instantaneous and 5-minute average observations.

5.4 Smaller and larger machines' capacities

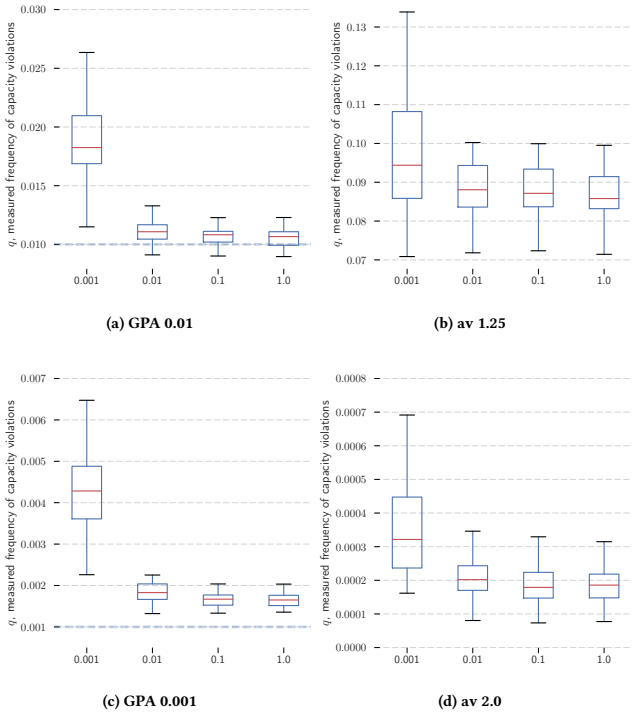
By varying the bin capacity c, we are able to simulate different ratios of job requirements to machine capacity. (Note that for $c < 1$, some of the tasks might not fit into any available machine having, e.g., the mean usage greater than c; however, as large tasks are rare, it was not the case for the 50 instances considered in the experiments). As both the number of machines used and q are normalized, we expect these values to be independent of c. Figure 11 compares *av 2* to *GPA 0.001*; and *av 1.25* to *GPA 0.1* as for capacity $c = 1$ these pairs resulted in similar (q, m) combinations (see Figure 8).

Overall, GPA results in similar q for different machine capacities. The differences in GPA results can be explained by two effects. When capacities are smaller, the effects of underestimating q (observed in Figure 9) are more significant. When capacities are larger, GPA results in less capacity violations than the requested thresholds. This is the impact of the last-opened bin which, with high probability, is underloaded; for larger capacities this last bin is able to absorb more tasks during the rebalancing phase. Figures 11 (b) and (d), where we measure q for algorithms without rebalancing, confirms this explanation.



(a) av 2, GPA 0.001, rebalancing
(b) av 2, GPA 0.001, no rebalancing
(c) av 1.25, GPA 0.1, rebalancing
(d) av 1.25, GPA 0.1,no rebalancing

Figure 11: q for av and GPA by different machine capacities $c \in \{0.5, 0.8, 1.0, 2.0, 5.0\}$.



(a) GPA 0.01
(b) av 1.25
(c) GPA 0.001
(d) av 2.0

Figure 12: q for GPA and av as a function of different clairvoyance levels.1 is full clairvoyance; 0.001 corresponds to 10 observations; 0.01 to 100, etc. $c = 1$.

In contrast, for av, q differs significantly when capacities change. This result demonstrates that using fixed thresholds for av estimation on heterogeneous resources results in unpredictable frequency of capacity violations: thresholds have to be calibrated by trial and error, and a threshold achieving a certain QoS for a certain machine capacity results in a different QoS for a different machine capacity.

5.5 Clairvoyance

Next, we analyze how the algorithms are affected by reduced quality of input data. We vary the clairvoyance level, i.e., the fraction of samples belonging to the observation set, in $\{0.001, 0.01, 0.1, 0.3, 0.5, 0.8, 1.0\}$. For instance, for clairvoyance level 0.001, the estimators have $0.001 \cdot 10\,000$, or just 10 *inst* observations to build the tasks' statistical model; to compute the empirical QoS q, the produced packing is then evaluated on the remaining 9 990 observations (the only difference is clairvoyance 1.0, for which the estimation and the evaluation sets both consisted of all $R = 10\,000$ samples). Figure 12 summarizes the results for av (which we treat as a baseline) and GPA estimators. We omit results for other av; we also omit results for GPA with other thresholds ϱ, as they were similar to $\varrho = 0.01$. Figures for GPA and av have different Y scales: our goal is to compare the relative differences between algorithms, rather than the absolute values (which we do in Section 5.2).

Just 100 observations are sufficient to achieve a similar empirical QoS level q as the fully-clairvoyant variant for both GPA and av. although, comparing results for levels 0.01 and 0.1, GPA has a slightly higher mean than av. As we demonstrated in Section 5.2, GPA underestimates rare events; hiding data only magnifies this effect. Furthermore, as the model build by GPA is more complex (it estimates both the average and the standard deviation), for smaller clairvoyance levels, we expected GPA to have relatively worse q. However, with the exception of the smallest threshold $\varrho = 0.001$, the relative degeneration of GPA and of av is similar.

5.6 Impact of Rebalancing on Frequency of Capacity Violations

Finally, we measure how much the rebalancing reduces the frequency of capacity violations, compared to the results of the on-line bin packing algorithm. Figure 13 shows the *relative* gains achieved by rebalancing (normalized by q of the base algorithm; we omit *perc* and algorithms for which the base algorithm had zero q). Rebalancing uses the unused capacity of the last-opened machine, which is usually severely underloaded, to move some of the tasks from other machines; thus leading to less capacity violations. The mean relative decrease in capacity violations for both av and GPA is around 7%, which shows that rebalancing modestly improves QoS. The median absolute number of machines m_{abs} used by GPA is between 35 (GPA 0.1) and 47 (GPA 0.001). Thus, the last machine represents at most roughly 2%-3% of the overall capacity (in case the machine is almost empty). As shown in Figure 8, the relationship between the capacity and the frequency of capacity violations q is

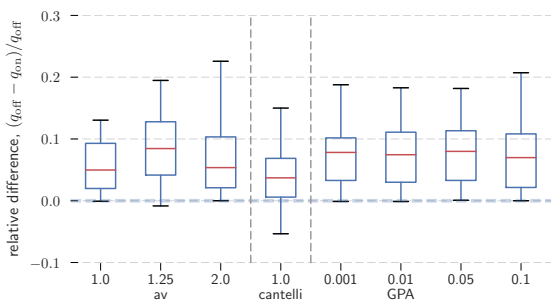


Figure 13: Relative decrease in the measured frequency of capacity violations from rebalancing. q_{on} denotes q with rebalancing; while q_{off} without.

exponential: small capacity increases result in larger decreases of capacity violations.

6 RELATED WORK

This paper has two principal contributions: the analysis of a new data set of the instantaneous CPU usage; and GPA, a new method of allocating tasks onto machines.

In our data analysis of the Google cluster trace [23, 31] (Section 3) we focused on the new information, the instantaneous CPU usage. [11, 22] analyze the rest of the trace; and [5] analyzes a related trace (longer and covering more clusters).

Data center and cloud resource management is an active research area in both systems and theory. A recent survey concentrating on virtual machine placement and on theoretical/simulation approaches is [21]. Our paper modeled a problem stemming from placement decisions of a data center scheduler, such as Borg [29]; we did not consider many elements, including handling IO [8, 14] or optimization towards specific workloads, such as data-parallel/map-reduce computations [2, 10].

We concentrated on bin packing, as our goal was to study how to maintain an SLO when tasks' resource requirements change. If some tasks can be deferred, there is also a scheduling problem; if the tasks arrive and depart over time, but cannot be deferred, the resulting problem is dynamic load balancing [9, 20]. We considered simple bin packing as the core sub-problem of these more complex models, as one eventually has to solve packing as a sub-problem. Moreover, scheduling decisions in particular are based on complex policies which are not reflected in the trace and thus hard to model accurately.

Our method, GPA, uses a standard bin packing algorithm, but changes the fitting criterion. Bin packing and its variants have been extensively used as a model of data center resource allocation. For instance, [27] uses a dynamic bin packing model (items have arrival and departure times) with items having known sizes. [25] studies relaxed, on-line bin packing: they permit migrations of items between bins when bins become overloaded. Our focus was to model uncertainty of tasks' resource requirements through stochastic bin packing. Theoretical approaches to stochastic bin packing usually solve the problem for jobs having certain distribution. [3, 30] consider bin packing with Gaussian items; [33] additionally takes into account bandwidth allocation. [13] considers load balancing, knapsack and bin packing with Poisson, exponential and Bernoulli items. [18] for bin packing shows an approximation algorithm for Bernoulli items. [7] solves the general problem by deterministic bin packing of items; item's size is derived from the item's stochastic distribution (essentially, the machine capacity is divided by the number of items having this distribution that fit according to a given SLO) and correlation with other items. According to their experimental evaluation (on a different, not publicly-available trace, and using 15-minute usage averages), this method overestimates the QoS (for target $\rho = 0.05$, they achieve $q = 0.02$); they report the number of machines 10% smaller than *perc 95* (although the later result is in on-line setting: usage is estimated from the previous period).

GPA estimates machine's CPU usage by a Gaussian distribution following the central limit theorem (CLT), perhaps the simplest possible probabilistic model. The CLT has been applied to related problems, including estimation of the completion time of jobs composed of many tasks executed across several virtual machines [32]. The stochastic bin packing algorithms that assume Gaussian items proposed for bandwidth consolidation [3, 30] can be also interpreted as a variant of CLT if we drop the Gaussian assumption.

[16] addresses stochastic bin packing assuming items' means and standard deviations are known. It essentially proposes to rescale each item's size according to Cantelli inequality (item's mean plus 4.4 or 1.7 times the standard deviation, see Section 5.1). Our experimental analysis in Section 5.2 shows that such rescaling overestimates the necessary resources, resulting in allocations using 2.5-5 times more machines than the lower bound. Consequently, for the target 95% SLO, Cantelli produces QoS of 99.9998%.

To model resource heterogeneity, bin packing is extended to vector packing: an item's size is a vector with dimensions corresponding to requirements on individual resources (CPU, memory, disk or network bandwidth) [19, 26]. Our method can be naturally extended to multiple resource type: for each type, we construct a separate GPA; and a task fits into a machine only if it fits in all resource types. This baseline scenario should be extended to balancing the usage of different kinds of resources [19], so that tasks with complementary requirements are allocated to a single machine.

Our method estimates tasks' mean and standard deviation from tasks' observed instantaneous usage. [17] combines scheduling and two dimensional bin packing to optimize tasks' cumulative waiting time and machines' utilization. The method uses machine learning to predict tasks' peak CPU and memory consumption based on observing first 24 hours of task's resource usage. If machine's capacity is exceeded, a task is evicted and rescheduled. While our results are not directly comparable to theirs (as we do not consider scheduling, and thus evictions), we are able to get sufficiently accurate estimates using simpler methods and by observing just 100 samples (Section 5.5). While to gather these 100 samples we need roughly 42 trace hours, a monitoring system should be able to take a sufficient number of samples in just a few initial minutes. [4] uses an artificial neural network (ANN) in a combined scheduling and bin packing problem. The network is trained on 695 hours of the Google trace to predict machines' performance in the subsequent hour. Compared to ANN, our model is much simpler, and therefore easier to interpret; we also need less data for training. [12] analyzes resource sharing for streams of tasks to be processed by virtual

machines. Sequential and parallel task streams are considered in two scenarios. When there are sufficient resources to run all tasks, optimality conditions are formulated. When the resources are insufficient, fair scheduling policies are proposed. [1] uses statistics of the past CPU demand of tasks (CDF, autocorrelation, periodograms) to predict the demand in the next period; then they use bin packing to minimize the number of used bins subject to a constraint on the probability of overloading servers. Our result is that a normal distribution is sufficient for an accurate prediction of a high percentile of the *total* CPU usage of a group of tasks (in contrast to individual task's).

7 CONCLUSIONS

We analyze a new version of the Google cluster trace that samples tasks' instantaneous CPU requirements in addition to 5-minute averages reported in the previous versions. We demonstrate that changes in tasks' CPU requirements are significantly higher than the changes reported by 5-minute averages. Moreover, the distributions of CPU requirements vary significantly across tasks. Yet, if ten or more tasks are colocated on a machine, high percentiles of their total CPU requirements can be approximated reasonably well by a Gaussian distribution derived from the tasks' means and standard deviations. However, 99th and 99.9th percentiles tend to be underestimated by this method. We use this observation to construct the Gaussian Percentile Approximation estimator for stochastic bin packing. In simulations, GPA constructed colocations with the observed frequency of machines' capacity violations similar to the requested SLO. Nevertheless, because of using the Gaussian model, GPA underestimates rare events: e.g., for a SLO of 0.0010, GPA achieves frequency of 0.0016. Thus, for such SLOs, GPA should be invoked with lower goal tresholds. Compared to a recently-proposed method based on Cantelli inequality [16], for 95% SLO, GPA reduces the number of machines between 1.9 (when Cantelli assumes Gaussian items) and 3.7 (for general items) times. GPA also turned out to work well with machines with different capacities. Moreover, as input data it requires only the mean and the standard deviation of each task's CPU requirement — in contrast to the complete distribution. We also demonstrated that these parameters can be adequately estimated from just 100 observations. Apart from the rebalancing step, our algorithms are on-line: once a task is placed on a machine, it is not moved. Thus, the algorithms can be applied to add a single new task to an existing load (if all tasks are released at the same time).

Using the Gaussian distribution is a remarkably simple approach—we achieve satisfying QoS without relying on machine learning [17] or artificial neural networks [4]. We claim that this proves how important high-frequency data is for allocation algorithms.

Our analysis can be expanded in a few directions. We deliberately focused on a minimal algorithmic problem with a significant research interest in the theoretical field — stochastic bin packing — but using realistic data. We plan to extend our experiments to stochastic processes (raw data from the trace, rather than stationary distributions generated from them) to validate whether the algorithms still work as expected. We also plan to drop assumptions we used in this early work: to extend packing algorithms to multiple dimensions; to measure and then cope with correlations between tasks; or to pack pools of machines with different capacities. An orthogonal research direction is a requirement estimator more robust than GPA, as GPA systematically underestimates rare events (small machines or SLOs of 99th or 99.9th percentile).

ACKNOWLEDGMENTS

We thank Jarek Kuśmierek and Krzysztof Grygiel from Google for helpful discussions; Krzysztof Pszeniczny for his help on statistical tests; and anonymous reviewers and our shepherd, John Wilkes, for their helpful feedback. We used computers provided by ICM, the Interdisciplinary Center for Mathematical and Computational Modeling, University of Warsaw. The work is supported by Google under Grant No.: 2014-R2-722 (PI: Krzysztof Rzadca).

REFERENCES

[1] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. 2007. Dynamic placement of virtual machines for managing SLA violations. In *IM*. IEEE.
[2] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *OSDI*. USENNIX.
[3] David Breitgand and Amir Epstein. 2012. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *INFOCOM*. IEEE.
[4] Faruk Caglar and Aniruddha Gokhale. 2014. iOverbook: intelligent resource-overbooking to support soft real-time applications in the cloud. In *IEEE Cloud*.
[5] Marcus Carvalho, Walfredo Cirne, Francisco Brasileiro, and John Wilkes. 2014. Long-term SLOs for reclaimed cloud computing resources. In *SoCC*. ACM.
[6] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby. 1999. Benchmarks and standards for the evaluation of parallel job schedulers. In *JSSPP*. Springer.
[7] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. 2011. Effective VM sizing in virtualized data centers. In *IM*. IEEE.
[8] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient coflow scheduling without prior knowledge. In *SIGCOMM*. ACM.
[9] Edward G. Coffman, Michael R. Garey, and David S. Johnson. 1983. Dynamic bin packing. *SIAM J. Comput.* 12, 2 (1983).
[10] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. 2016. Job-aware scheduling in Eagle: Divide and stick to your probes. In *SoCC*. ACM.
[11] Sheng Di, Derrick Kondo, and Cappello Franck. 2013. Characterizing cloud applications on a Google data center. In *ICPP*. IEEE.
[12] Sheng Di, Derrick Kondo, and Cho-Li Wang. 2015. Optimization of Composite Cloud Service Processing with Virtual Machines. *IEEE Trans. on Computers* (2015).
[13] Ashish Goel and Piotr Indyk. 1999. Stochastic load balancing and related problems. In *FOCS, Procs*. IEEE.
[14] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert NM Watson, and Steven Hand. 2016. Firmament: fast, centralized cluster scheduling at scale. In *OSDI*. USENIX.
[15] John A Hartigan. 1975. *Clustering algorithms*. Wiley.
[16] Inkwon Hwang and Massoud Pedram. 2016. Hierarchical, Portfolio Theory-Based Virtual Machine Consolidation in a Compute Cloud. *IEEE Trans. on Services Computing* PP, 99 (2016).
[17] Jesus Omana Iglesias, Liam Murphy Lero, Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. 2014. A methodology for online consolidation of tasks through more accurate resource estimations. In *IEEE/ACM UCC*.
[18] Jon Kleinberg, Yuval Rabani, and Éva Tardos. 2000. Allocating bandwidth for bursty connections. *SIAM J. Comput.* 30, 1 (2000), 191–217.
[19] Sangmin Lee, Rina Panigrahy, Vijayan Prabhakaran, Venugopalan Ramasubramanian, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. 2011. *Validating heuristics for virtual machines consolidation*. Technical Report.
[20] Yusen Li, Xueyan Tang, and Wentong Cai. 2014. On dynamic bin packing for resource allocation in the cloud. In *SPAA*. ACM.
[21] Ilia Pietri and Rizos Sakellariou. 2016. Mapping virtual machines onto physical machines in cloud computing: A survey. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 49.
[22] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM SoCC*.
[23] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. 2011. *Google cluster-usage traces: format + schema*. Technical Report. Google Inc., Mountain View, CA,

USA. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[24] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: flexible, scalable schedulers for large compute clusters. In *EuroSyS*. ACM.

[25] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. 2014. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Trans. on Computers* 63, 11 (2014).

[26] M. Stillwell, F. Vivien, and H. Casanova. 2012. Virtual Machine Resource Allocation for Service Hosting on Heterogeneous Distributed Platforms. In *IPDPS Procs.* IEEE.

[27] Xueyan Tang, Yusen Li, Runtian Ren, and Wentong Cai. 2016. On First Fit Bin Packing for Online Cloud Server Allocation. In *IPDPS*.

[28] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. 2009. Server workload analysis for power minimization using consolidation. In *USENIX*.

[29] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *EuroSys*. ACM.

[30] Meng Wang, Xiaoqiao Meng, and Li Zhang. 2011. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *InfoCom*. IEEE.

[31] John Wilkes. 2011. More Google cluster data. Google research blog. (Nov. 2011). Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[32] Sungkap Yeo and Hsien-Hsin Lee. 2011. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *IEEE Computer* 44, 8 (2011), 55–62.

[33] J. Zhang, Z. He, H. Huang, X. Wang, C. Gu, and L. Zhang. 2014. SLA aware cost efficient virtual machines placement in cloud computing. In *IPCCC*.