WSMeter: A Performance Evaluation Methodology for Google's Production Warehouse-Scale Computers

Jaewon Lee[†], Changkyu Kim[‡], Kun Lin[‡], Liqun Cheng[‡], Rama Govindaraju[‡], Jangwoo Kim[†]

[†] Electrical and Computer Engineering, Seoul National University [‡] Platforms Engineering, Google

Abstract

Evaluating the comprehensive performance of a warehouse-scale computer (WSC) has been a long-standing challenge. Traditional load-testing benchmarks become ineffective because they cannot accurately reproduce the behavior of thousands of distinct jobs co-located on a WSC. We therefore evaluate WSCs using actual job behaviors in live production environments. From our experience of developing multiple generations of WSCs, we identify two major challenges of this approach: 1) the lack of a holistic metric that incorporates thousands of jobs and summarizes the performance, and 2) the high costs and risks of conducting an evaluation in a live environment.

To address these challenges, we propose *WSMeter*, a cost-effective methodology to accurately evaluate a WSC's performance using a live production environment. We first define a new metric which accurately represents a WSC's overall performance, taking a wide variety of unevenly distributed jobs into account. We then propose a model to statistically embrace the performance variance inherent in WSCs, to conduct an evaluation with minimal costs and risks.

We present three real-world use cases to prove the effectiveness of WSMeter. In the first two cases, WSMeter accurately discerns 7% and 1% performance improvements from WSC upgrades using only 0.9% and 6.6% of the machines in the WSCs, respectively. We emphasize that naive statistical comparisons incur much higher evaluation costs (> 4×) and sometimes even fail to distinguish subtle differences. The third case shows that a cloud customer hosting two services on our WSC quantifies the performance benefits of software optimization (+9.3%) with minimal overheads (2.3% of the service capacity).

CCS Concepts • **General and reference → Metrics**; **Evaluation**; **Performance**; • **Information systems → Data centers**; • **Computing methodologies → Model development and analysis**;

Keywords warehouse-scale computer, performance evaluation, statistical modeling

ACM Reference format:
Jaewon Lee[†], Changkyu Kim[‡], Kun Lin[‡], Liqun Cheng[‡], Rama Govindaraju[‡], Jangwoo Kim[†] . 2018. WSMeter: A Performance Evaluation Methodology for Google's Production Warehouse-Scale Computers. In *Proceedings of ASPLOS '18: Architectural Support for Programming Languages and Operating Systems, Williamsburg, VA, USA, March 24–28, 2018 (ASPLOS '18),* 15 pages.
https://doi.org/10.1145/3173162.3173196

ASPLOS '18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN 978-1-4503-4911-6/18/03.
https://doi.org/10.1145/3173162.3173196

1 Introduction

Warehouse-Scale Computer (WSC) [3, 28, 41], a large collection of tightly networked computers, has become the standard platform to host modern internet services at global scale. For WSCs hosting a single business-critical job (e.g. HPC applications, websearch [4, 6]), the performance evaluation becomes a straightforward task of measuring *the job*'s performance. For many other production WSCs hosting *thousands* of distinct jobs [49, 54], evaluating and improving the performance has been a long-standing challenge. While a large volume of studies propose novel features to improve the jobs and WSCs [9, 10, 18–21, 27, 29, 31, 36–40, 42, 44, 52, 53, 56], the adoption of such features is significantly hindered by the lack of *a standard WSC evaluation methodology*.

For example, WSC providers often face critical purchasing or upgrading decisions which cost *hundreds of millions of dollars*. While Total Cost of Ownership (TCO) estimation of a new feature (e.g., hardware upgrades or software improvements [35]) is relatively clear, there is *no systematic methodology* to accurately *quantify* its comprehensive performance impact. Consequently, WSC providers often perform expensive WSC resizing and restructuring after a feature deployment to balance the performance and costs.

Due to the limited scope of jobs and deterministic behaviors, conventional performance evaluation practices using load-testing benchmarks [5, 14, 24, 32, 55, 58] become ineffective for accurate WSC performance evaluation. The job instances [1] in our production WSCs dynamically share the computing resources (e.g., CPU, RAM, I/O) and exhibit non-deterministic performance variances. It is practically impossible to develop a benchmark which accurately models and reproduces such behaviors for thousands of different jobs.

[1] Instances of a job are identical binaries performing identical task on different pieces of data (i.e., shards).

Excluding the traditional load-testing benchmarks, we naturally evaluate a WSC with actual job behaviors in a live environment. From our experience of developing and operating multiple generations of WSCs, we identify two fundamental challenges of this live-environment evaluation.

Lack of a holistic WSC performance metric. A WSC typically hosts thousands of jobs with distinct performance characteristics and metrics. Moreover, different WSCs often host different sets of jobs. It is challenging to summarize the performance as a whole and draw a conclusion accommodating all these factors.

High costs and risks of a live-environment evaluation. Performance evaluation on a live environment incurs significant costs and risks. We often purchase new hardware in a warehouse scale, drain and reboot a large number of machines to apply a feature, and carefully perform these steps over a long period of time to minimize the service availability losses. In addition, such features are often under development and therefore are not very stable. There is a high risk of introducing performance or even functional bugs to a live production WSC impacting millions of users.

To address these challenges, we propose **W**arehouse-**S**cale performance **Meter** (WSMeter), a methodology to efficiently and accurately evaluate a WSC's performance using a live production environment.

We first study the attributes of thousands of production jobs running on a live WSC to propose the definition of a *holistic* WSC performance metric. We define the metric as the quota-weighted [2] sum of per-job performance to factor in the peak resource usage and maximize the overall performance per costs. Further analysis on the job characteristics shows that this metric can be accurately derived from a small subset of heavyweight (i.e. large-quota) jobs, facilitating a cost-effective performance evaluation.

Next, we develop a statistical model to efficiently embrace the performance variance inherent in WSCs and accurately estimate the performance metric discussed above. We design a highly efficient WSC performance model which utilizes the characteristics of the production jobs. Our model estimates the performance with tight error margins while minimizing the evaluation costs and risks.

WSMeter benefits both WSC providers (e.g., WSC administrators such as Google) and customers (e.g., cloud-based service providers such as Spotify and Evernote [45]). The providers can estimate the potentials of their new WSC-improving hardware and software upgrades, and the customers can measure the impact of service optimizations.

[2] Quota represents the maximum amount of resources (e.g., CPU, RAM, storage, etc) that a job may use. In our environment, it also determines the costs to run a job.

We validate WSMeter with two WSC provider's use cases and one WSC customer's use case. For the first category, we 1) upgrade machine CPU and quantify the per-core performance improvements and 2) apply a new DVFS policy and measure its performance impacts. WSMeter accurately distinguishes 7% and 1% improvements from the two cases using only 0.9% and 6.6% of the WSC's capacity, respectively. We emphasize that a naive statistical approach incurs much larger overheads and sometimes fails to detect improvements. The WSC customer's case shows that WSMeter discerns 9.3% performance improvement from a software optimization using only 2.3% of the service instances. These results demonstrate that WSMeter enables an accurate and cost-effective WSC performance evaluation.

In summary, we make the following contributions:

- **Problem identification.** Through an analysis on live production environments constituting global internet services, we explain the challenges of WSC performance evaluation and show that conventional practices become ineffective.
- **Holistic performace metric for WSCs.** To quantify the performance benefits of new features, we develop a metric which accurately summarizes the performance of a production WSC running thousands of jobs.
- **Statistical performance model for WSCs.** To efficiently derive the WSC performance metric, we develop a statistical model based on the performance characteristics of production jobs. The model allows us to discern subtle performance differences using only a small fraction of a WSC.
- **Wide applicability.** Both WSC providers and customers may use WSMeter to effectively analyze and quantify the performance of their services, jobs, and platforms.
- **Real-world validation.** We validate WSMeter against two WSC improvement cases (i.e., upgrading CPUs, applying a new DVFS policy) and one service optimization case (i.e., software upgrade) and successfully demonstrate the effectiveness of our approach. We are actively using WSMeter to examine other novel WSC enhancement schemes.

This paper is organized as follows. Section 2 motivates the need for systematic WSC performance evaluation. Section 3 explains the process of developing the WSC performance metric. Section 4 describes our statistical approach to accurately and efficiently derive the metric. Section 5 illustrates implementation considerations of WSMeter. Section 6 shows the evaluation results from three case studies. Related work is presented in Section 7. We conclude the paper in Section 8.

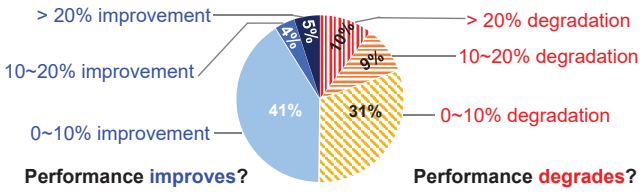


Figure 1. Performance impact of a new DVFS policy on 1000+ production jobs. The pies represent the portion of the jobs falling into the category.

2 Background and Motivation

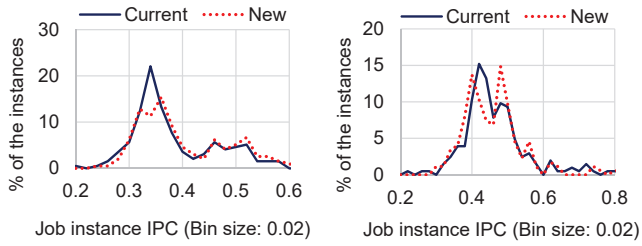
In this section, we introduce the need for a systematic WSC performance evaluation methodology, explain its key challenges with a detailed analysis, and set the design goals to motivate WSMeter.

Analysis methodology. We observe our live production WSC running thousands of jobs which constitute global-scale internet services (e.g., databases [8, 12], data analytics [16], websearch [4, 6], infrastructure support [7]). The WSC consists of several thousand machines – we subdivide it into a *current* and a *new* WSC following the validation methodology in Section 6.1, to measure the performance impact of a new feature (i.e., new DVFS policy). The machines in the WSC have identical hardware and software setup (other than the feature) to eliminate the performance variation from platform heterogeneity. We monitor the job performance for 24 hours and report the averages to minimize the effect of the variations over time.

2.1 Need for a Systematic Performance Evaluation Methodology

To improve a WSC's performance and cost-effectiveness, a large volume of studies discuss resource- and performance-aware job scheduling [18–21, 36–38, 44, 49, 52–54, 56], adaptive power management [29, 31, 39, 40, 42], novel architectures [9, 10, 27], and systematic performance investigation methods [2, 17, 22, 25, 30, 33, 47, 48, 50].

Unfortunately, to the best of our knowledge, there is *no standard methodology* to evaluate the *holistic performance* of a WSC running thousands of distinct jobs. As a result, WSC providers and customers often struggle to heuristically maintain the balance between conflicting performance demands. Figure 1 shows such an example where we need to evaluate whether a new DVFS policy is beneficial for the overall performance. We observe that half of the jobs experience performance boost while the other half suffer from degradations. It is difficult to finalize the decision because the jobs have conflicting performance behaviors and there is no metric to summarize the overall performance impact. This example well illustrates the need for a systematically defined performance evaluation methodology.



(a) Job 1: Performance improves (The new WSC has +3.9% IPC).

(b) Job 2: Performance degrades (The new WSC has -1.2% IPC).

Figure 2. Performance distribution (histogram) of two selected jobs on the current and new WSC. Both jobs have 200+ instances.

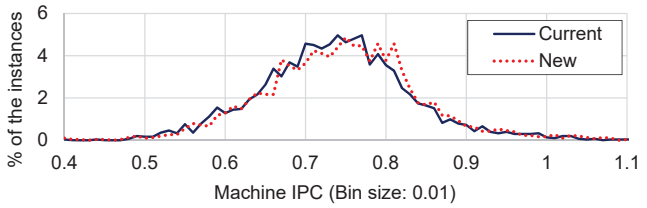


Figure 3. Performance distribution (histogram) of the machines in the current and new WSC. The new WSC has +1% IPC.

2.2 Analysis on WSC Performance Characteristics

We analyze the characteristics of a WSC to identify why performance evaluation becomes challenging for WSCs.

First, we analyze WSC job performance characteristics. We pick one performance-increasing job and one performance-degrading job from Figure 1, and observe the performance distributions of their instances (Figure 2). While the average job performance clearly increases or decreases in the new WSC, the individual instance performance shows a wide range of behaviors and it is difficult to distinguish the average differences even from the distributions. The performance variances come from input data variance and resource interference, which are the inevitable nature of distributed jobs running on a multi-tenant WSC. This indicates that it is impossible to accurately estimate the jobs' performance from few naively selected instances.

Figure 3 shows the performance distribution of the machines in the current and new WSCs. As the machines host a variety of jobs and the jobs again have a wide range of performance behaviors, the machines also exhibit a wide range of performance despite the homogeneity of hardware and software setup.

In summary, the jobs and machines in WSCs show diverse behaviors at large scale, making it difficult to accurately quantify the performance from few observations.

2.3 Challenges of the Performance Evaluation

We now discuss how the inherent WSC characteristics incur performance evaluation challenges.

Ineffective load-testing benchmarks. Traditional load-testing benchmarks aim to faithfully model and reproduce realistic job behaviors to stress and evaluate a system. However, a WSC's job behaviors are too diverse and complex to understand and emulate via simple benchmarks. Even if one manages to record all the job and performance characteristics, it is practically infeasible to reproduce such a large-scale behavior, nor is there a system (other than a WSC) to run the load and measure performance results. Therefore, load-testing approach becomes ineffective for evaluating WSCs.

Lack of a WSC performance metric. Conventionally, the performance of a system is reported as the user-level performance metrics (e.g., wall-clock execution time, operations per second) from well-understood load-testing benchmarks such as SPEC [13] and TPC's server benchmarks [43]. As they become unsuitable for WSC evaluation, we lose the performance metrics as well.

As illustrated in Figure 1, defining and comparing WSC performance is challenging because the jobs often show conflicting performance behaviors. The lack of WSC performance metric hinders making critical decisions to improve the overall performance in a data driven manner.

High costs and risks of a live-environment evaluation. As we drop load-testing benchmarks from our options, we now deploy a new feature to a live production environment to understand the performance impacts. Unfortunately, such an evaluation incurs prohibitive costs and risks.

While a feature deployment itself incurs considerable costs (e.g., purchasing hardware, affecting production jobs), the large performance variance forces us to deploy the feature to a large number of machines (or jobs) to accurately measure the new performance. In addition, deploying an experimental feature with possible bugs runs the risk of disrupting live production services. These challenges discourage WSC providers and customers from aggressively evaluating novel features to improve their performance.

2.4 Design goals of WSMeter

To address the challenges discussed in the previous section, we set the design goals of WSMeter as follows.

Develop a WSC performance metric. We aim to develop a WSC performance metric to facilitate performance evaluation and comparison. The metric should represent the performance of hundreds of thousands of jobs with conflicting behaviors as a whole, to confirm whether a new feature improves or degrades the overall performance.

Propose a low-cost and low-risk performance evaluation method. We aim to reduce the overheads of live WSC performance evaluation to encourage WSC providers and customers to aggressively evaluate and deploy new features. We emphasize that even a small improvement may translate into considerable cost reductions at such a scale. The efficient evaluation method should reduce the costs and risks while accurately delivering the WSC performance metric discussed above.

3 Holistic Performance Metric for Warehouse-Scale Computers

In this section, we discuss our approach to define a holistic performance metric for WSCs. We introduce our metric development process and discuss the challenges of deriving the metric along with our strategy to overcome them.

3.1 Defining the Performance Metric

Intuitively, the overall WSC performance should be an aggregation of per-job performance. We therefore start from the basic form below:

$$WSC\ performance\ metic = \sum Weight_i \times Performance\ metric_i$$

where i denotes a unique job ID. The equation raises three questions: 1) *how is the weight determined?*, 2) *which performance metric should we use?*, and 3) *which jobs should we include?*. We explain our observations and choices.

Weight. Weight determines how much a job's performance affects the overall performance. By setting it to the factor which we value (e.g., cost, criticality), we can prioritize the performance of more valuable jobs (i.e., higher cost or criticality). With this setting, maximizing the performance metric eventually maximizes the performance per the value as well.

For our environment, we use a job's *quota* to determine the weight (i.e., $Weight_i$ = Job i's quota / Total quota). Since quota is a measure of resources a job is allowed to use, our definition encourages to improve the performance of resource-intensive jobs to ultimately improve the overall resource utilization of a WSC. In addition, quota also determines the costs to execute a job in our system; improving the performance metric therefore best increases the cost efficiency as well.

As a practical approach, we use *CPU quota* among the resources that define a quota. While some resources only guarantee the running space for a job (e.g., RAM capacity, storage capacity), CPU quota determines *how much time a job is active* and therefore better represents the overall job activity. We note that previous work [30] also profiled the CPU cycle consumption (i.e., time) to successfully derive meaningful insights from WSC performance behaviors.

Performance metric. Defining a unified performance metric is difficult because a WSC hosts a wide variety of jobs with different metrics (e.g., wall clock execution time, operations per second). If all the jobs in a WSC (or all the services

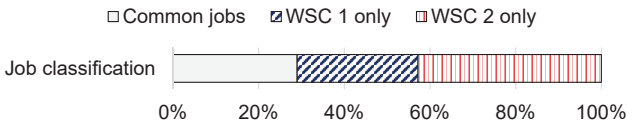
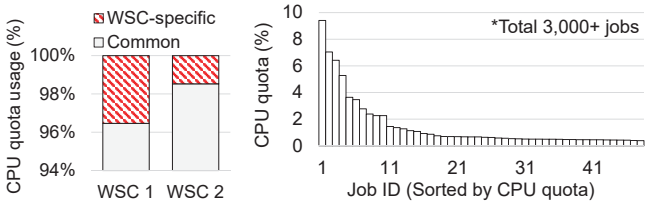


Figure 4. Job existence on two different WSCs. Only 30% of the jobs run on both WSCs.



(a) CPU quota breakdown. **(b)** CPU quota of the jobs in WSC 1.

Figure 5. CPU quota analysis for the jobs in the two WSCs.

of a WSC customer) share the same user-level performance metric, we recommend using it as it best represents the true job performance. Our WSC customer use case (Section 6.5) is such an example.

In other cases, we use IPC (or MIPS) as the performance metric for the following reasons. First, it is well-correlated to the user-level performance metrics for our production jobs. Thanks to the highly optimized binaries (e.g., minimal spinlocks), our jobs have a stable instruction footprint per request/load and the performance impacts from various resources (e.g., CPU, memory, I/O) all appear as IPC changes. In addition, our WSC schedulers dynamically adapt resource allocations to minimize the idle resources while guaranteeing the QoS (e.g., latency-sensitive jobs). This further ensures the IPC to be proportional to the throughput by eliminating the low-IPC high-throughput jobs (i.e., scheduler takes such jobs' wasted resources away to increase the IPC and utilization). Our previous studies [53, 57] also confirm these observations. Second, since it is a hardware-based metric which all the jobs have, we can easily aggregate the performance of thousands of different jobs.

Job selection. Ideally, the WSC performance metric should be derived from: 1) identical set of jobs as we want to measure the performance under identical (or at least similar) WSC behavior, and 2) all the jobs running on a WSC as they all contribute to the overall performance.

However, we note that it is impossible to achieve both properties as WSCs run vastly different sets of jobs. In the following sections, we further discuss the challenges of job selection and introduce our solution.

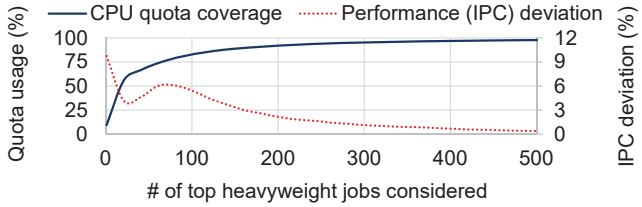


Figure 6. CPU quota coverage and performance (IPC) deviation for varying number of top heavyweight jobs considered.

3.2 Challenge I: Uneven Distribution of Jobs

We observe that different WSCs host very different sets of jobs, making fair and accurate WSC performance comparisons difficult. Figure 4 illustrates an example from our WSCs. As a retrospective study, we attempt to compare the performance of two full-scale WSCs of similar sizes (i.e., WSC 1 and 2). However, among all the jobs that appear on the two WSCs, only 30% are common on both WSCs. With such an uneven distribution of jobs, it is difficult to quantify the performance differences without introducing a bias.

To improve fairness, we may use only the common jobs of the WSCs while sacrificing the representativeness of the evaluation. We further analyze the CPU quotas of the WSCs to identify how much bias we would introduce by excluding the WSC-specific jobs. Surprisingly, the results (Figure 5a) show that a significant portion (95%+) of the quota is consumed by the common jobs that appear on both WSCs. Further analysis (Figure 5b) shows that there exist heavyweight jobs which occupy a large portion of the total quota. We find that many of these heavyweight jobs are basic infrastructure jobs (e.g., databases, caches, distributed locks) that the other higher-level jobs utilize; they are therefore common across different WSCs. Based on this observation, we decide to use the common jobs of the WSCs to derive the performance and perform fair comparisons.

3.3 Challenge II: Large Number of Jobs

In the previous section, we show that the common heavyweight jobs constitute the first-order performance characteristics. However, the absolute number of the jobs can be still very large, making the performance measurement difficult.

We therefore attempt to further reduce the number of jobs to include in the performance evaluation. Specifically, we vary the number of jobs to include in the performance calculation and observe how much the performance deviates from the all-job-based performance. We test for the top 1 to 500 heavyweight jobs in WSC 1 (Figure 4).

Figure 6 shows the CPU quota coverage (i.e., quota of the selected jobs / quota of all jobs) and the performance (IPC) deviation. As expected, including more jobs reduces the performance deviation and improves the representativeness. For this WSC, considering 10%~20% of the common jobs (in Figure 4) reduces the deviation to few percent.

Summary of the job selection. We decide to use a small number of top heavyweight jobs to accurately and efficiently derive the performance metric. Note that the appropriate number of heavyweight jobs may differ for other WSCs as they may have different job characteristics. We emphasize that the heavyweight jobs are still much more diverse compared to the jobs in typical load-testing benchmark suites (e.g., about a dozen), highlighting the need for an evaluation using live environments.

4 Statistical Performance Model for Warehouse-Scale Computers

In this section, we describe our statistical approaches to reduce the live performance evaluation costs while accurately delivering the WSC performance metric.

4.1 Embracing the Variance with Statistics

To minimize the evaluation costs while maximizing the performance estimation fidelity, we use statistical tools to systematically embrace the performance variance. We consider a (performance) metric with variance as a *random variable* following a certain probability distribution. The Central Limit Theorem (CLT) [3] then suggests that if we make sufficient observations, we can accurately deduce the *average* statistic [23]. In particular, if we make N observations (i.e., take a sample with size N) from an arbitrary distribution following (μ, σ^2), the mean of the N observations (i.e., sample mean, m) should follow a t-distribution with $(\mu, \sigma^2/N)$ [4]. We may estimate the population mean (μ) as:

$$\bar{m} - t \times s/\sqrt{N} < \mu < \bar{m} + t \times s/\sqrt{N}$$

where $\bar{m}$ and s are the mean and standard deviation from N observations (i.e., the sample) and t decides the confidence level. A common practice is to set $t \approx 2$ to achieve confidence level $\approx 95\%$ [5]. For a given confidence level, we control the fidelity of the estimation with the sample size (N). Taking a larger sample will make the confidence interval narrower (or reduce the margin of error, $t \times s/\sqrt{N}$) and improve the estimation fidelity.

We demonstrate the effectiveness of the theory by performing samplings on a performance metric (i.e., machine performance in Figure 3). Figure 7 shows the sample mean's (m) distribution from 1,000 sampling trials, for two sample sizes. Increasing the sample size successfully improves the fidelity of the estimation. The larger sample (N=100) has the estimations heavily clustered around the true average value while the smaller sample (N=1) does not. This shows that CLT allows us to estimate the average of highly variable metrics with minimal costs.

[3] We safely apply CLT as WSC performance metrics' variances are finite.
[4] We use t-distribution as we infer the population variance from sample.
[5] 95% confidence indicates that if we repeat estimating the mean, 95 out of 100 intervals will correctly include the population mean μ.

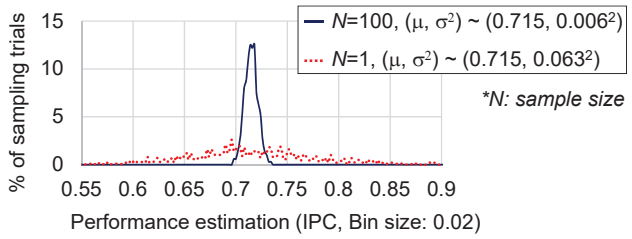


Figure 7. Estimating WSC machine performance with different sample sizes. We perform 1,000 trials and present the distribution (histogram).

4.2 Baseline Performance Model: Machine Granularity

A straightforward way to apply the statistical model is to consider the machine performance as a random variable as we did for Figure 7. Applying the CLT, we can estimate the average WSC machine performance (μ) as $\bar{m} \pm t \times s/\sqrt{N}$, where $\bar{m}$ and s are the sample's mean and standard deviation, and N is the sample size (i.e., number of machines observed).

While this baseline model accurately estimates the average WSC machine performance, it has the following drawbacks. First, it does not follow our WSC performance metric definition (discussed in Section 3). Accordingly, we cannot emphasize per-job importance using weights, use user-level performance metrics, or ensure that we are deriving the performance from a carefully selected set of jobs. Second, its evaluation costs are generally higher than WSMeter's job granularity model (introduced in Section 4.3). As the baseline model tries to make estimations from the *sum* of highly variable job behaviors, it achieves lower efficiency compared to the model which precisely utilizes per-job behaviors. We further discuss its inefficiencies in Section 6.

4.3 WSMeter's Performance Model: Job Granularity

To overcome the limitations of the baseline model, we propose a *job granularity* performance model for WSMeter. In this model, we estimate each job's performance and aggregate multiple job performance estimations to construct the overall performance. The model therefore faithfully follows the WSC performance definition (from Section 3) and leverages its benefits. In addition, WSC customers hosting multiple services may also use this model by substituting the job performance with their service performance and the job weight with their own definition of service weight.

We first assume job i's performance has mean and variance (μ_i, σ_i^2). Observing N_i job instances, its sample mean (m_i) would follow a t-distribution with $(\mu_i, \sigma_i^2/N_i)$. The comprehensive WSC performance is the weighted sum of the per-job performance:

$$WSC\ Performance = \sum(w_i \times m_i)$$

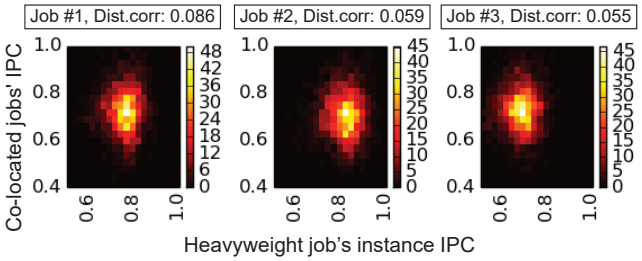
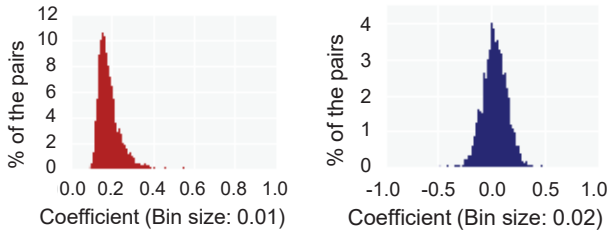


Figure 8. Heavyweight job performance and the other co-located jobs' performance, represented as heatmaps. Distance correlation coefficients are also provided.



(a) Distance correlation. **(b)** Pearson's correlation.

Figure 9. Performance correlation between the heavyweight jobs. The job pairs are formed from the top 100 heavyweights. We present the distributions (histograms) of the coefficients.

where w_i denotes the weight of the job i (discussed in Section 3.1). The summation runs over all the jobs we consider and the sum of the weights equals 1 (i.e., 100%). As the summation of t-distribution random variables does not follow a t-distribution (or other well-known distributions), we approximate m_i to follow a normal distribution. We justify this approach in Section 5.2.1.

Under the normality assumption, the weighted sum of the per-job performance (m_{Sum}, i.e., $\sum(w_i \times m_i)$) follows a normal distribution with the following mean and variance:

$$\mu_{Sum} = \sum(w_i \times \mu_i)$$
$$\sigma^2_{Sum} = \sum(w_i \times \sigma_i/\sqrt{N_i})^2 + 2\sum Covariance(m_i, m_j)$$

We examine the correlations between the job performances because they determine the covariance term of the σ^2_{Sum} (which we try to minimize). First, we analyze the performance relationship between the heavyweight jobs and the other co-located jobs (Figure 8). Specifically, for each of the top three heavyweight jobs, we plot the job's instance IPCs with the machine IPCs calculated *excluding* the job's instance IPCs (i.e., the co-located jobs' IPCs). From the shape of the distributions, we notice that the two variables lack a linear relationship. We also calculate the distance correlation coefficients [51] of the two variables. Unlike conventionally used Pearson's correlation coefficient, (near-)zero distance correlation coefficient implies that the variables are statistically independent. From the small coefficients, we conclude that the two IPCs are independent.

Second, we investigate the performance correlation between the heavyweight jobs. We pick a pair of heavyweight jobs, calculate the distance/Pearson's correlation using the two jobs' IPCs obtained from the same racks (e.g., job A's IPC from rack 1~5 and job B's IPC from rack 1~5), and show the histograms of the coefficients in Figure 9. Majority of the job pairs have a small distance correlation coefficient, indicating that they are almost independent. In addition, the Pearson's coefficients are clustered around zero; since the covariance is a scaled sum of the Pearson's coefficients, it is very likely to have a (near-)zero value.

Therefore, we approximate the WSC performance as a normal random variable whose mean is the weighted sum of the per-job performance estimations and the variance is the weight-squared sum of the per-job variances (i.e., ignore the covariance term).

We now control the performance estimation's variance (σ^2_{Sum}, i.e., evaluation fidelity) by adjusting the job performance estimations' variances (σ^2_i/N_i). Note that the variance is determined by the number of observations (N_i). Our goal is to satisfy the variance requirement ($\sigma^2_{Sum} < \sigma^2_{Target}$) while minimizing the evaluation costs ($\sum N_i \times Cost\ of\ observing\ Job\ i's\ instance$). In our case, the cost of observing a job instance is set proportional to the job instance's CPU quota.

We formulate a discrete optimization problem to find the optimal combination of N_is using a solver as follows:

Minimize the cost function:

$$\sum N_i \times Cost\ of\ observing\ Job\ i's\ instance$$

While satisfying the following constraints:

$$min(N_i) < N_i < MAX(N_i)\ for\ all\ i$$
$$\sigma^2_{Sum} < \sigma^2_{Target}$$

$min(N_i)$ is set to satisfy the normality assumption (details in Section 5.2.1) and $MAX(N_i)$ is the total number of job i's instances in a WSC.

By observing the N_i job instances that a solver suggests, we can estimate the WSC performance metric (i.e., estimate μ_{Sum} as $\bar{m}_{Sum} \pm t \times s_{Sum}$) while meeting the target evaluation fidelity. We demonstrate its effectiveness in Sections 5 and 6.

5 Implementation

In this section, we elaborate on the implementation details of WSMeter. We first provide an overview of the performance evaluation process. We then discuss practical implementation challenges and our design decisions to address them. Lastly, we describe our automated WSMeter framework.

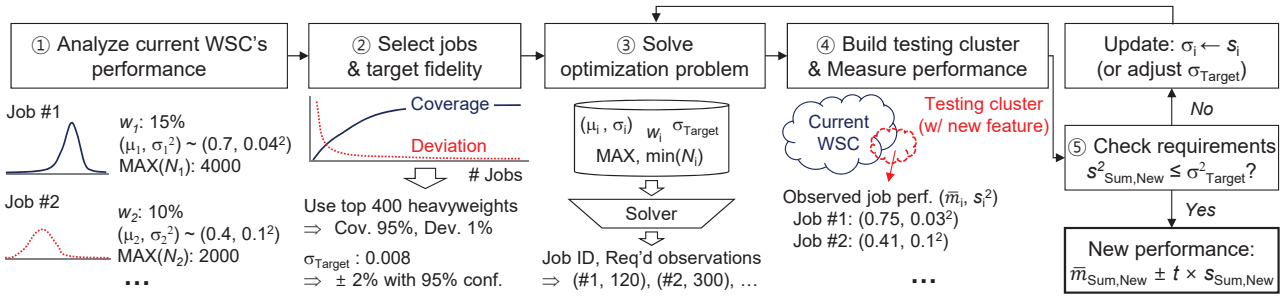


Figure 10. Overview of WSMeter's performance evaluation process.

5.1 Overview of the WSMeter Process

Figure 10 illustrates WSMeter's performance evaluation steps. The example assumes the WSMeter's job granularity model but the baseline model follows a similar process.

Step 1. Analyze the current WSC's performance. First, we analyze the job performance characteristics of the current WSC. We measure the jobs' weight (w_i), performance distribution (μ_i and σ_i), and the number of instances ($MAX(N_i)$). Note that we investigate *all* the job instances in the WSC to calculate population performance statistics (i.e., μ_i and σ_i), not sample statistics (i.e., $\bar{m}_i$ and s_i). As a result, we get the exact performance of the current WSC which we later compare with the new performance. The per-job performance characteristics are utilized again in the later steps to determine the evaluation costs for a given target fidelity.

Step 2. Select the jobs and target evaluation fidelity. Next, we select the jobs to include in the performance evaluation and decide the target evaluation fidelity. We investigate the tradeoff between job diversity and weight coverage / performance deviation (e.g., Figure 6) to select the jobs. Including more jobs improves the representativeness (i.e., quota coverage) and reduces the deviation, but has a potential of increasing the evaluation costs. Note that the job selection also affects the weights of the jobs (w_i) because we always normalize the sum of the weights to be 100%.

For the target evaluation fidelity, we set the upper bound of the performance variance (σ^2_{Target}). We can also interpret this as setting the margin of error (i.e., confidence interval width represented as the % of the average value). Lowering the bound reduces the margin of error and better estimates the performance at higher costs. In our example, we set σ_{Target} = 0.008 / ±2% margin of errors, which is suitable for detecting >4% changes [6].

[6] The confidence interval of the new performance estimation *should not overlap* with the current WSC performance to claim significant differences, because the true (i.e., population) new performance may lie *anywhere* in that interval. In general, $\pm x\%$ confidence interval robustly distinguishes $2x\%$ performance difference because in the worst case, the true new performance would be on one boundary (e.g., $+x\%$) and the current WSC performance would be on the other boundary (e.g., $-x\%$, hence $2x\%$ away from the true new performance).

Step 3. Solve an optimization problem. Using the information from the first two steps, we solve the optimization problem discussed in Section 4.3. The solver generates the number of job instances to observe (N_i) to satisfy the target fidelity (i.e., $\sigma^2_{Sum,New} \leq \sigma^2_{Target}$), while minimizing the evaluation costs.

Step 4. Build a testing cluster and measure the performance. The fourth step is to build a testing cluster with the new feature and measure the new job performance. The cluster is constructed by either replacing the existing machines in the current WSC or adding new machines along with the current WSC. We discuss how we monitor the N_i job instances from the testing cluster in Section 5.2.2.

From this step, we calculate the new job performance characteristics ($\bar{m}_i, s_i$) as well as the new overall performance characteristics ($\bar{m}_{Sum,New}$, $s_{Sum,New}$) following the model in Section 4.3. Note that we use *sample statistics* as we do not have the population (i.e., a full WSC with the new feature).

Step 5. Check the fidelity requirements. Lastly, we ensure that the new performance variance meets the fidelity requirements (i.e., $s^2_{Sum,New} \leq \sigma^2_{Target}$). Often, the new feature *increases* the performance variance of the jobs (i.e., $\sigma_{i,New} > \sigma_{i,Current}$). In such a case, N_is become *insufficient* because they are derived based on the smaller variance. We therefore re-calculate the N_is by updating the variances in the optimization problem and perform another round of measurement, until the requirement is satisfied [7]. Alternatively, we may relax the fidelity requirements to simply accept less precise results. Upon passing the variance checking step, we safely accept the estimate of the new performance.

5.2 Challenges and Design Decisions

We now discuss the practical challenges of implementing WSMeter and our design decisions to handle them.

[7] We use sample statistics ($s_{i,New}$) to *estimate* the new performane variance ($\sigma_{i,New}$). To embrace estimation errors and prevent excessive iterations, we recommend observerving more job instances than required (i.e., > N_i) to create headroom in the evaluation fidelity.

Figure 11. Standardized sample mean distribution for the top four heavyweight jobs, for two different sample sizes (20 and 4). 100 trials are performed for each job and sample size.

5.2.1 Normality of the sample statistics

When developing the performance model, we use Central Limit Theorem (CLT) to assume the normality of the sample mean (m_i). While CLT requires a sufficient number of observations to work (i.e., $N_i > \min(N_i)$), the minimum number varies depending on the underlying distribution. We thus examine how many observations are sufficient.

We first select the top four heavyweight jobs and check whether their performance estimations (m_i) follow a normal distribution. Figure 11 shows the standardized cumulative distribution functions (CDF) of the job performance estimations and the ideal normal distribution. The results show that sample size 20 (i.e., 20 observations) is sufficient to ensure normality for all four jobs. Size 4 incurs slight deviation for some cases (e.g., Job 1) but the distributions are sharper than the standard normal distribution.

We also examine the effect of minimum sample size on the evaluation costs and fidelity (Figure 12). For two minimum sample sizes (i.e., 20 and 4) and three different job selections (i.e., top 50, 100, and 200 heavyweights), we solve the optimization problem targeting ±1% margin of error to obtain the number of required observations (N_i). From the N_is, we get the corresponding margin of error (which should be $\leq \pm 1\%$) and the total machine costs to host the job instances.

When $\min(N_i) = 4$, we achieve the exact target fidelity (i.e., margin of error). Considering more heavyweight jobs does *not* necessarily increase the costs, because newly included jobs 1) occasionally have smaller variance than the existing jobs and 2) take away the weights of the existing jobs to eventually help reduce the overall variance. For $\min(N_i) = 20$, the costs increase as the number of jobs increases because we always end up selecting the minimum number of instances for all the jobs (i.e., $N_i = 20$ for all i's satisfies the fidelity requirements for all three cases).

Based on these empirical analyses, we set $\min(N_i)$ to 4 to strike the balance among coverage, fidelity and cost.

5.2.2 Constructing the testing cluster in a live WSC

One approach to build a testing cluster is to compactly pack the N_i job instances on the testing cluster, which allows us

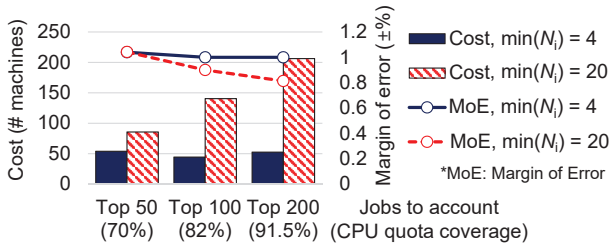


Figure 12. Evaluation costs and fidelity for different job selections and $\min(N_i)$s. The costs and fidelity are derived from the optimization problems.

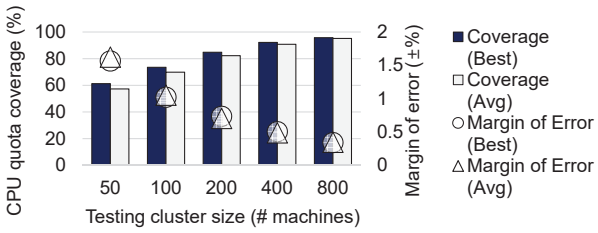


Figure 13. Coverage and performance estimation fidelity for various testing cluster sizes.

to achieve the minimal evaluation costs that a solver calculates. However, this also creates an unnatural environment that exacerbates the contention on shared resources; the machines in a live production WSC usually do not experience such intense job co-locations. Accordingly, the performance evaluation results would be no longer realistic. Moreover, packing the instances into a few racks and power domains increase the risk of failure and impact the availability of production services.

To preserve all the scheduling rules and observe the instances in a natural co-location scenario, we use a set of machines that *already have* the required job instances to construct a testing cluster. For the ease of implementation, instead of solving an optimization problem and trying to find a set of machines which have $\geq N_i$ instances for the jobs we consider, we 1) randomly pick a fixed number of machines to form a testing cluster, 2) select the jobs which satisfy $N_i > \min(N_i)$ to ensure normality, and 3) estimate the performance from the selected jobs. If the selected jobs provide too low coverage (i.e., too many jobs get excluded due to $N_i < \min(N_i)$), we may re-select the machines until the coverage becomes satisfactory. In case we add new machines to build a testing cluster, we ask the scheduler to migrate the selected jobs (in the randomly-selected existing machines) to the new machines.

To illustrate the effectiveness of our cluster construction method, we analyze the coverage and performance estimation fidelity for various testing cluster sizes (Figure 13). For each size, we perform 100 cluster constructions and report the best and average values. We note that both the best

Figure 14. WSMeter framework and its operations.

and average achieve high fidelity and reasonable coverage at affordable costs. Further analysis on the testing clusters tells that heavyweight jobs' instances are quite prevalent so our randomly generated testing clusters often have enough instances to achieve high coverage and low margin of error.

5.3 Framework Implementation

We implement WSMeter as an automated framework to facilitate evaluating new features. Figure 14 shows its operations.

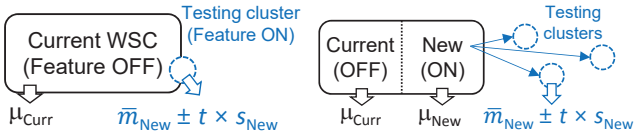
First, WSMeter periodically (e.g., every 24 hours) gathers per-job instance performance (e.g., IPC, user-level performance metrics) to analyze the current performance behavior. The information is stored on a performance database.

Second, WSMeter uses the information from the first step and a solver to calculate the observation requirements (N_i). The framework then applies the new feature to job instances. For per-job features, WSMeter randomly selects N_i instances and applies the feature. For per-machine features, WSMeter randomly selects a predefined number of machines, applies the feature, and checks which job instances are affected.

Third, WSMeter observes the new job performance and checks the fidelity requirements. If the requirements are met, the user can get the performance estimation satisfying the targeted fidelity. If not, the user ends up with less accurate results (i.e., larger margin of error than expected). In this case, the framework automatically adjusts the observation requirements and deploys the feature accordingly so that the next iteration produces the results meeting the target fidelity.

6 Evaluation

We now provide evaluation results to show the effectiveness of WSMeter. We first compare the cost and fidelity of the statistical performance models (discussed in Section 4.2). Next, we introduce two case studies of applying new features to our WSCs and measuring the performance impacts – *upgrading machine CPUs* and *improving the power management policy*. Lastly, we provide a case where a WSC customer uses our framework to identify the benefits of *job instance optimizations*.



(a) Setup for an actual usage **(b)** Setup for a validation

Figure 15. Illustration of testing cluster setups.

6.1 Experimental Setup

WSC setup. The WSCs we use consist of several thousand machines with identical hardware/software configurations within each WSC. Our evaluation results are from thousands of live production jobs such as databases [8, 12], data analytics [16], websearch [4, 6], and infrastructure support [7], which constitute the frontend and the backbone of *global internet services*. We report the average performance metrics from 24-hour periods to minimize the effect of the variations over time. The case studies may show different performance characteristics as we use different WSCs.

Validation methodology. When using WSMeter for an actual performance evaluation scenario, we would use a setup illustrated in Figure 15a. In this case, we *cannot* validate the accuracy of our performance estimation because we do not have a WSC with the new feature and thus do not know the true performance improvements.

We therefore use the setup in Figure 15b to validate the quality of our performance estimation. We construct a WSC where a half of the machines have the new feature applied. This allows us to calculate the *reference performances* (μ_{Curr} and μ_{New}), which show the true benefit of the feature. Note that we confirm the positive effects of the feature *a priori* and make such an aggressive deployment only for validation purposes.

We then draw a testing cluster from the new WSC and estimate the performance with error margins ($\bar{m}_{New} \pm t \times s_{New}$). For an estimation to be successful, it must 1) *exclude the current WSC performance* (μ_{Curr}) to claim a meaningful difference, and 2) *include the new WSC performance* (μ_{New}) to be accurate. We perform multiple testing cluster draws (e.g., 100 times) and measure how many estimations satisfy these two criteria to evaluate the robustness of our method.

6.2 Cost and Fidelity Analysis.

We first compare the cost and fidelity of the baseline statistical model and WSMeter's model (Figure 16). The ideal fidelity-costs from an optimization solver is also presented.

The baseline has low efficiency because of its coarse-grain definition of performance (i.e., machine granularity). A machine performance is an addition of highly variable job behaviors, where the job combination differs machine by machine. In contrast, our model describes the fidelity requirements for each job. This preciseness in the requirement description

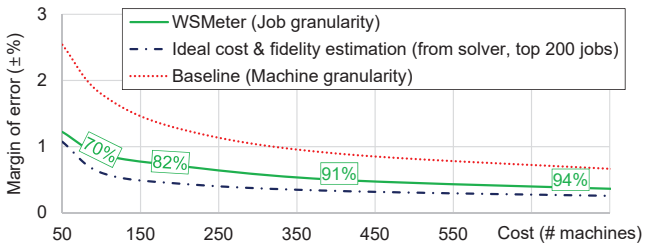
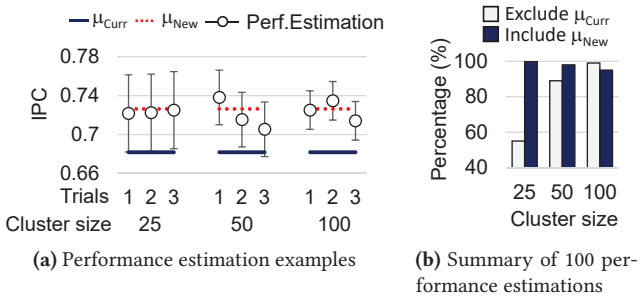


Figure 16. Evaluation costs and fidelity of the statistical performance models. WSMeter has a varying coverage due to the nature of our testing cluster construction (Section 5.2.2). The ideal estimation achieves 91.5% coverage.



(a) Performance estimation examples

(b) Summary of 100 performance estimations

Figure 17. CPU upgrade: Estimating the per-core performance improvement using the baseline model.

allows us to achieve small overall performance variance and high efficiency.

6.3 Case study I: Machine CPU Upgrade

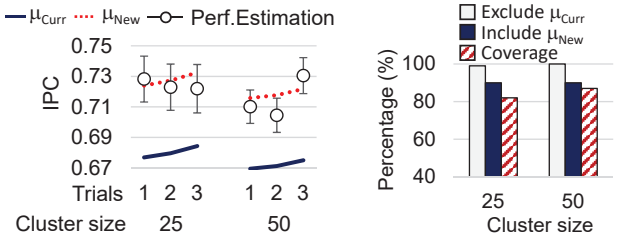
In the first case study, we measure the performance improvements from upgrading machine CPUs. We describe how the baseline and WSMeter detect the benefits.

Baseline. Using the baseline model, we would first measure the current WSC's performance distribution to determine the testing cluster size. In this case, the standard deviation is about 15% of the average performance (μ_{Curr}). This tells that deploying 25~100 machines would provide estimation with ±3%~±6% margin of error (95% confidence).

Next, we estimate the performance from the testing cluster. We try three different cluster sizes (i.e., 25, 50, and 100 machines). Figure 17a shows the performance estimations from three trials (i.e., testing cluster selections) for each size, along with the reference performances; the reference improvement is 6.5%. 25 machines have too wide margins to claim statistically significant difference (i.e., exclude μ_{Curr}) while 100 machines are enough to tell the difference. This is natural considering that their margin of errors are ±6% and ±3%, respectively; usually the performance difference should be larger than the margin width to be robustly distinguished. Figure 17b shows the performance estimations from 100 trials for each size. Again, small clusters often fail to exclude the current performance to claim meaningful improvement. We note that all sizes correctly include the new performance (μ_{New}) for more than 95% as we set the confidence level to 95%.

Summary – The baseline detects 6.5% performance improvement using 100 machines (1.7% of WSC), with ±3% margin of error.

WSMeter. To further reduce the evaluation costs from the baseline, we use WSMeter's model. We first calculate the ideal costs to roughly estimate the testing cluster size. We solve optimization problems which consider top 200 heavyweight jobs and target ±3% and ±1% fidelities. Table 1 shows



(a) Performance estimation examples

(b) Summary of 100 performance estimations

Figure 18. CPU upgrade: Estimating the performance improvement using WSMeter.

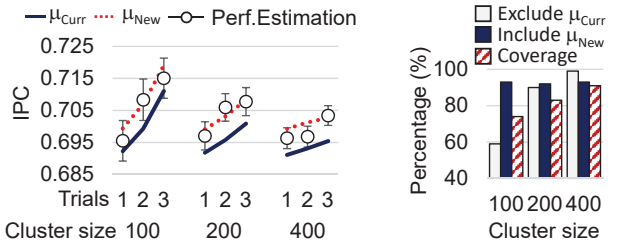
that 40~43 machines would provide about ±1%~±1.5% margin of error. Note that for ±3% case, the solver estimates the costs as 40 machines to satisfy the minimum observation constraint (i.e., $min(N_i) = 4$).

As WSMeter's model has slightly lower efficiency than the ideal cost estimation (Figure 16), we deploy 25-/50-machine cluster expecting to achieve >±1.5% margin of error. Figure 18a illustrates the performance estimations from three trials for each size. The reference performances and the margin of error slightly change for each trial because the job selection and the corresponding job weights change for different testing clusters (see Section 5.2.2 for details). We emphasize that the relative performance improvements are quite consistent (≈ 7%) and they are close to the improvement calculated from all the jobs (i.e., 100% coverage, 6% improvement [8]). Both 25 and 50 machine cluster show small margin of error

[8] The improvement is different from the baseline/machine granularity model (6.5%) because we use the job granularity model.

Table 1. Ideal testing cluster size and fidelity (Top 200 jobs).

Target fidelity	Cluster size (# machines)
±3%	40 (actually achieves ±1.5%)
±1%	43



(a) Performance estimation examples

(b) Summary of 100 performance estimations

Figure 19. New DVFS policy: Estimating the performance improvement using WSMeter.

(≈ ±2.3% and 1.6% respectively) and correctly exclude the current WSC performance to claim statistically significant performance improvements.

Figure 18b summarizes the results from 100 trials. We note that the estimations successfully exclude the current performance but often fails to include the new performance (i.e., 90% inclusion, while it should be ≥95%). We believe assuming zero covariance (Section 4.3) results in underestimating the variance and corresponding margin of error. We recommend to accurately consider correlation factors and the corresponding increase in variance (if necessary) or simply make a conservative estimation by increasing the margin. In our case, expanding the margin of error by ~20% makes the interval to include the new performance for >95% of the trials. Lastly, we confirm that the CPU quota coverage is high enough to ignore the performance deviation from the full-coverage results.

Summary – WSMeter detects ≈7% performance improvement using 25~50 machines (0.4%~0.9% of WSC) with ±2.8%~±1.9% margin of error (20% increased for a conservative statement), while covering 82%~87% of the total CPU quota. This is > 4× higher efficiency compared to the baseline, assuming the same evaluation fidelity.

6.4 Case Study II: New DVFS Policy

As the second case study, we measure the performance impact of improving the power management (i.e., DVFS) policy. It is critical to optimize the policy because a well-tuned policy can improve the performance per TCO without introducing extra software or hardware overheads.

Baseline. Similar to the first case study, we start by checking the variance of the current WSC to determine the testing cluster size. In this case, the standard deviation is 13% of the average performance and 100 machines would provide ±2.6% fidelity. However, we notice that the estimated performance from the testing cluster almost always includes the current performance (i.e., no significant difference exists). Further investigation on the reference performance reveals that the improvement is small (1.2%) and the 100-machine cluster is not enough to distinguish the benefits.

Summary – Theoretically, we can detect the improvement by setting the target fidelity to ±0.6% and using 1,900 machines. Considering the impractical costs, we declare a failure to detect the improvements.

WSMeter. For WSMeter, we try 100, 200, and 400 machine clusters. We omit the details of ideal cost analysis for brevity.

Figure 19a shows the performance estimations and the reference performance. The reference improvement for partial coverage (i.e., testing clusters) and full coverage (i.e., all jobs) are both ≈1%. Similar to the first case, our method achieves high evaluation fidelity (≈ ±0.92%, ±0.63%, ±0.45% for 100, 200, and 400 machines, respectively). Figure 19b shows that 400-machine cluster robustly identifies the improvement while also achieving high coverage (91%). The evaluation is relatively expensive (6.6% of WSC) but worth it because the feature is relatively easy to apply (i.e., hot patch) and we can achieve improvements without introducing new hardware or software costs.

This second case study further emphasizes the importance of high fidelity and low cost performance evaluation. Despite small improvements, the long term benefit of such a feature is enormous considering the scale and operation costs of WSCs.

Summary – WSMeter distinguishes ≈1% performance difference using 400 machines (6.6% of WSC) with 0.45% margin of error, while covering 91% of the total CPU quota. We emphasize that the baseline fails to detect the improvement with reasonable costs.

6.5 Case Study III: Optimizing Service Instances

This case study shows how a WSC customer who hosts internet services on our cloud may use WSMeter to evaluate the comprehensive service performance. We assume a scenario where the customer applies software optimizations (i.e., compiler optimizations and software updates) to the service instances. The customer operates one compute-intensive service and one network-intensive service of equal importance and cost (i.e., weight). For each job, 1,500+ instances are deployed on the cloud [46] as virtual machines.

Table 2 summarizes the performance characteristics of the two services. We use normalized performance metrics (i.e., throughput) for performance evaluation. Our validation methodology (Section 6.1) measures the reference performance improvements as 6.6% and 12%, respectively. The reference improvement in the overall performance metric is therefore 9.3%.

We note that the performance improvement of the compute-intensive job measured on an empty local testing cluster is 11%, which is very different from the WSC results (6.6%). This discrepancy *strongly encourages* the WSC customers,

Table 2. Performance characteristics of a WSC customer's compute- and network-intensive services.

Name	Compute	Network
μ_{Curr}	100	100
σ_{Curr}	7.4	17.5
$\mu_{New,WSC}$	106.6	112
$\mu_{New,Local}$	111	N/A
Weight	50%	50%
Cost	1	1

service developers, and researchers to *evaluate new features on live WSCs* to measure the *realistic* performance impacts.

Feeding the information in Table 2 into WSMeter framework with ±3% target fidelity, it suggests to deploy 21 compute-intensive service instances and 49 network-intensive service instances. We construct the testing cluster as suggested and observe the results (Table 3). The error margin of each job is larger than the targeted value (±3%), but as we apply the weighted sum (following the job-level model definition in Section 4.3), the margin of the overall performance metric satisfies the requirements. As our performance estimation always includes the reference performance improvement (9.3%), we validate the effectiveness of our framework for a WSC customer's use case.

7 Related work

Benchmarks targeting WSC environments. Many studies aim to characterize and reproduce the job behaviors of WSCs and other large-scale computing environments [5, 14, 17, 22, 24, 32, 55, 58]. While they provide valuable insights regarding the inherent characteristics of various jobs, the analyses assume a single job running in a controlled environment and therefore lack realistic variances shown in WSCs (discussed in Section 2). Some benchmark suites [14, 17] reproduce resource sharing behaviors to better model job co-location scenarios. However, their job diversity and resource sharing patterns are too simple and deterministic to faithfully reproduce WSC-like behaviors.

Accounting performance variances. Previous work [1, 11, 15, 26, 34] discusses various sources of performance variability (e.g., nondeterminism of multi-threaded applications, dynamics of Java environment) and statistical methods to embrace them. Some work [1, 15, 26] uses the CLT and t-statistic as we do. Other work suggests hierarchical performance testing [11] or resampling method [34] which reliably discern performances with relaxed requirements on the population distribution (e.g., sample mean does not need to follow a normal distribution).

The major difference between the previous work and our method is that we propose a much more holistic model and methodology developed based on the nature of WSCs to efficiently embrace the variance (e.g., job granularity performance model). The previous work deals with repeatable single-job benchmarks which are much simpler to model and apply statistical techniques. Note that we develop our model based on parametric statistics (e.g., CLT and t-statistic) because it works well for our scenario. The other types of testing (suggested in [11, 34]) can be orthogonally applied with appropriate modeling changes.

Table 3. Calculating the comprehensive performance of a WSC customer's two cloud-based services.

Testing cluster (Trial #)	Compute	Network	Overall
1	105.8±3.2	110.5±5	108.2±3
2	106.4±3.2	111.4±5	108.9±3
3	108.2±3.2	115.8±5	112.0±3

8 Conclusion

This paper proposes WSMeter, a cost-effective methodology to accurately evaluate Google's WSCs in live production environments. We first proposed a performance metric which accurately accounts for thousands of production jobs with complex behaviors. The metric facilitated quantifying the performance changes from new features and WSC design changes. We then proposed a statistical model to efficiently embrace the performance variance in multi-tenant WSCs. The model allowed us to accurately derive the performance metric with minimal costs and risks. Our three case studies from real-world WSCs showed that WSMeter successfully distinguished performance improvements using only a small subset of a full WSC.

Acknowledgments

We thank Bob Cypher for guiding us to investigate this critical problem. We thank Parthasarathy Ranganathan, Robert Hundt, David Lo, Sundar Dev, Lai Nguyen, Matt Brown, Peter Dahl, and others in Technical Infrastructure team for helping us carefully run experiments and providing invaluable insights to improve the paper. This work was also partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIT) (NRF-2015M3C4A7065647, NRF-2017R1A2B3011038), and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korean Government (MSIT) (No. R0190-15-2012).

References

[1] Alaa R. Alameldeen and David A. Wood. 2003. Variability in Architectural Simulations of Multi-Threaded Workloads. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA '03)*. IEEE Computer Society, Washington, DC, USA, 7–. http://dl.acm.org/citation.cfm?id=822080.822813

[2] Paul Barham, Rebecca Isaacs, and Dushyanth Narayanan. 2003. Magpie: online modelling and performance-aware systems. In *9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*. USENIX.

[3] Luis Andre Barroso, Jimmy Clidaras, and Urs Hoelzle. 2013. *The Datacenter as a Computer:An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool. 154– pages. https://doi.org/10.2200/S00516ED2V01Y201306CAC024

[4] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. 2003. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro* 23, 2 (March 2003), 22–28. https://doi.org/10.1109/MM.2003.1196112

[5] PerfKit Benchmarker. 2017. PerfKit Benchmarker. (2017). http://googlecloudplatform.github.io/PerfKitBenchmarker/

[6] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst.* 30, 1-7 (April 1998), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X

[7] Mike Burrows. 2006. The Chubby Lock Service for Loosely-coupled Distributed Systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 335–350. http://dl.acm.org/citation.cfm?id=1298455.1298487

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages. https://doi.org/10.1145/1365815.1365816

[9] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. ACM, New York, NY, USA, 17–32. https://doi.org/10.1145/3037697.3037700

[10] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 681–696. https://doi.org/10.1145/2872362.2872368

[11] Tianshi Chen, Qi Guo, Olivier Temam, Yue Wu, Yungang Bao, Zhiwei Xu, and Yunji Chen. 2015. Statistical Performance Comparisons of Computers. *IEEE Trans. Comput.* 64, 5 (May 2015), 1442–1455. https://doi.org/10.1109/TC.2014.2315614

[12] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's Globally-distributed Database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 251–264. http://dl.acm.org/citation.cfm?id=2387880.2387905

[13] Standard Performance Evaluation Corporation. 2017. SPEC. (2017). https://www.spec.org

[14] Standard Performance Evaluation Corporation. 2017. SPEC virt_sc 2013. (2017). https://www.spec.org/virt_sc2013

[15] Charlie Curtsinger and Emery D. Berger. 2013. STABILIZER: Statistically Sound Performance Evaluation. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. ACM, New York, NY, USA, 219–228. https://doi.org/10.1145/2451116.2451141

[16] Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: A Flexible Data Processing Tool. *Commun. ACM* 53, 1 (Jan. 2010), 72–77. https://doi.org/10.1145/1629175.1629198

[17] Christina Delimitrou and Christos Kozyrakis. 2013. iBench: Quantifying interference for datacenter applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*. 23–33. https://doi.org/10.1109/IISWC.2013.6704667

[18] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. ACM, New York, NY, USA, 77–88. https://doi.org/10.1145/2451116.2451125

[19] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 127–144. https://doi.org/10.1145/2541940.2541941

[20] Christina Delimitrou and Christos Kozyrakis. 2016. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 473–488. https://doi.org/10.1145/2872362.2872365

[21] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. 2015. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*. ACM, New York, NY, USA, 97–110. https://doi.org/10.1145/2806777.2806779

[22] Christina Delimitrou, Sriram Sankar, Kushagra Vaid, and Christos Kozyrakis. 2011. Decoupling datacenter studies from access to large-scale applications: A modeling approach for storage workloads. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*. 51–60. https://doi.org/10.1109/IISWC.2011.6114196

[23] William Feller. 1968. *An introduction to probability theory and its applications: volume I*. Vol. 3. John Wiley & Sons New York.

[24] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*. ACM, New York, NY, USA, 37–48. https://doi.org/10.1145/2150976.2150982

[25] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. 2007. X-trace: A Pervasive Network Tracing Framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation (NSDI'07)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=1973430.1973450

[26] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications (OOPSLA '07)*. ACM, New York, NY, USA, 57–76. https://doi.org/10.1145/1297027.1297033

[27] Johann Hauswald, Yiping Kang, Michael A. Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G. Dreslinski, Jason Mars, and Lingjia Tang. 2015. DjiNN and Tonic: DNN As a Service and Its Implications for Future Warehouse Scale Computers. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 27–40. https://doi.org/10.1145/2749469.2749472

[28] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture, Fifth Edition: A Quantitative Approach* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[29] Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenzano, David Meisner, Thomas F. Wenisch, Jason Mars, Lingjia Tang, and Ronald G. Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 271–282. https://doi.org/10.1109/HPCA.2015.7056039

[30] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-scale Computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 158–169. https://doi.org/10.1145/2749469.2750392

[31] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. 2014. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 31–40. https://doi.org/10.1109/IISWC.2014.6983037

[32] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. 1–10. https://doi.org/10.1109/IISWC.2016.7581261

[33] Christos Kozyrakis, Aman Kansal, Sriram Sankar, and Kushagra Vaid. 2010. Server Engineering Insights for Large-Scale Online Services. *IEEE Micro* 30, 4 (July 2010), 8–19. https://doi.org/10.1109/MM.2010.73

[34] Bin Li, Shaoming Chen, and Lu Peng. 2015. Precise computer comparisons via statistical resampling methods. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 83–92. https://doi.org/10.1109/ISPASS.2015.7095787

[35] David Xinliang Li, Raksit Ashok, and Robert Hundt. 2010. Lightweight Feedback-directed Cross-module Optimization. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '10)*. ACM, New York, NY, USA, 53–61. https://doi.org/10.1145/1772954.1772964

[36] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 450–462. https://doi.org/10.1145/2749469.2749475

[37] Jason Mars and Lingjia Tang. 2013. Whare-map: Heterogeneity in "Homogeneous" Warehouse-scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 619–630. https://doi.org/10.1145/2485922.2485975

[38] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, USA, 248–259. https://doi.org/10.1145/2155620.2155650

[39] David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV)*. ACM, New York, NY, USA, 205–216. https://doi.org/10.1145/1508244.1508269

[40] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. Power Management of Online Data-intensive Services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 319–330. https://doi.org/10.1145/2000064.2000103

[41] David A. Patterson. 2008. Technical Perspective: The Data Center is the Computer. *Commun. ACM* 51, 1 (Jan. 2008), 105–105. https://doi.org/10.1145/1327452.1327491

[42] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F. Wenisch, and Jack Underwood. 2010. Power Routing: Dynamic Power Provisioning in the Data Center. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS XV)*. ACM, New York, NY, USA, 231–242. https://doi.org/10.1145/1736020.1736047

[43] Transaction Processing performance Council. 2017. TPC-Homepage. (2017). http://www.tpc.org

[44] Vinicius Petrucci, Michael A. Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. Octopus-Man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 246–258. https://doi.org/10.1109/HPCA.2015.7056037

[45] Google Cloud Platform. 2017. Customer Success. (2017). https://cloud.google.com/customers

[46] Google Cloud Platform. 2017. Google Cloud Computing, Hosting Services & APIs. (2017). https://cloud.google.com

[47] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. 2010. Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers. *IEEE Micro* 30, 4 (July 2010), 65–79. https://doi.org/10.1109/MM.2010.68

[48] Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. 2006. Pip: Detecting the Unexpected in Distributed Systems. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3 (NSDI'06)*. USENIX Association, Berkeley, CA, USA, 9–9. http://dl.acm.org/citation.cfm?id=1267680.1267689

[49] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. ACM, New York, NY, USA, 351–364. https://doi.org/10.1145/2465351.2465386

[50] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a large-scale distributed systems tracing infrastructure*. Technical Report. Technical report, Google.

[51] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. 2007. Measuring and testing dependence by correlation of distances. *The annals of statistics* 35, 6 (2007), 2769–2794.

[52] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2011. The Impact of Memory Subsystem Resource Sharing on Datacenter Applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 283–294. https://doi.org/10.1145/2000064.2000099

[53] Lingjia Tang, Jason Mars, Xiao Zhang, Robert Hagmann, Robert Hundt, and Eric Tune. 2013. Optimizing Google's warehouse scale computers: The NUMA experience. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 188–197. https://doi.org/10.1109/HPCA.2013.6522318

[54] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. ACM, New York, NY, USA, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[55] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. 2014. BigDataBench: A big data benchmark suite from internet services. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 488–499. https://doi.org/10.1109/HPCA.2014.6835958

[56] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 607–618. https://doi.org/10.1145/2485922.2485974

[57] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. ACM, New York, NY, USA, 379–391. https://doi.org/10.1145/2465351.2465388

[58] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. 2016. Treadmill: Attributing the Source of Tail Latency Through Precise Load Testing and Statistical Inference. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 456–468. https://doi.org/10.1109/ISCA.2016.47