Taming Performance Variability

Aleksander Maricq and Dmitry Duplyakin, ***University of Utah;***
Ivo Jimenez and Carlos Maltzahn, ***University of California Santa Cruz;***
Ryan Stutsman and Robert Ricci, ***University of Utah***

https://www.usenix.org/conference/osdi18/presentation/maricq

This paper is included in the Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18).

October 8–10, 2018 • Carlsbad, CA, USA

ISBN 978-1-939133-08-3

Open access to the Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation is sponsored by USENIX.

Taming Performance Variability

Aleksander Maricq[*] Dmitry Duplyakin[*] Ivo Jimenez[†] Carlos Maltzahn[†]
Ryan Stutsman[*] Robert Ricci[*]

[*]*University of Utah,* [†]*University of California Santa Cruz*

The performance of compute hardware varies: software run repeatedly on the same server (or a different server with supposedly identical parts) can produce performance results that differ with each execution. This variation has important effects on the reproducibility of systems research and ability to quantitatively compare the performance of different systems. It also has implications for commercial computing, where agreements are often made conditioned on meeting specific performance targets.

Over a period of 10 months, we conducted a large-scale study capturing nearly 900,000 data points from 835 servers. We examine this data from two perspectives: that of a service provider wishing to offer a consistent environment, and that of a systems researcher who must understand how variability impacts experimental results. From this examination, we draw a number of lessons about the types and magnitudes of performance variability and the effects on confidence in experiment results. We also create a statistical model that can be used to understand how representative an individual server is of the general population. The full dataset and our analysis tools are publicly available, and we have built a system to interactively explore the data and make recommendations for experiment parameters based on statistical analysis of historical data.

1 Introduction

Variability is an unavoidable aspect of computer systems performance. In the research community, rigorous comparison of systems requires understanding, analysis, and control of system variability [45, 21, 12, 27]. In the commercial space, understanding and controlling performance variability is critical to providing good user experience [14, 23] and to plan resource provisioning [1].

Large systems have many sources of performance variability (hereafter referred to as simply "variability"), but one that cannot be avoided is the variability of hardware. For this paper, we consider two types of variability: variability of the *same physical system under repeated experiments*, and variance between different physical systems that are supposedly *identical*. Hardware can exhibit variability due to temperature [17], variations in timings and orderings, remapped storage blocks [44] or memory cells [52], variance in manufacture [65], "fail-slow" hardware [25], and many more causes.

We present findings and recommend best practices from two different perspectives: infrastructure-as-a-service (IaaS) *providers* and their *users*. On the provider side, we consider the amount of variability that can reasonably be controlled by factoring out unrepresentative servers, and how to reliably detect such devices. On the user side, we consider the variability that remains, how to cope with it when running experiments, and how to avoid certain pitfalls. Our intention is to make experimentation in the face of variability easier by demystifying its sources and quantities and by making concrete recommendations.

We collected data from servers in CloudLab [60], a platform for systems research that provides exclusive "raw" access to compute and storage resources. CloudLab allocates an entire server to one user at a time; we ran our benchmarks on servers when they were not allocated to any other user. This enables us to report performance numbers that users could reasonably expect to see in their own applications, unaffected by other simultaneous users. This data was collected on an IaaS provider that constitutes research infrastructure (a "testbed"), but we believe these lessons also apply to other settings in which there is an agreement between providers and users to supply a specific, measurable level of performance, such as clouds and datacenters.

In this paper, we:

- Provide a refresher of statistical methods used to assess confidence in performance results (§2) and the impact of variability on experimentation
- Describe our testing framework, the servers we tested, and the resulting dataset (§3)
- Analyze this dataset (§4) to understand the sources and quantities of variability
- Present a new method for estimating how many repetitions of an experiment to run (§5) and CONFIRM, our tool to aid experimenters in gathering statistically significant results
- Devise methods for service providers to identify servers with unrepresentative behavior (§6).
- Cover defensive practices (§7) that help avoid pitfalls with respect to variability

Throughout, we identify specific findings (identified with ◊) aimed at helping service providers provide more

consistent facilities and assisting users in understanding and coping with the variability inherent in computer systems. We close with related work and future directions.

2 The Statistics of Performance Variability

The fundamental way variability impacts systems research is that it affects our *confidence* in the statistical power of our results and the correctness of conclusions that we draw. When we run experiments and calculate statistics (mean, median, etc.) we are producing *empirical* statistics from a sample (a finite number) of a notional *population* (an infinite number) of executions. As we run more *repetitions* of an experiment, we can be more confident that our empirical distributions are close to the population distributions, and for key statistics such as the mean and median, we can compute *confidence intervals* (CIs).

For a chosen *confidence level* α, a CI defines a range in which we are α% sure that the population mean lies. For example, a sample mean of 10.0, with a CI of $9.9-10.1$ at 95% confidence indicates a 95% confidence that the true mean lies within $r = 1\%$ of our estimate 10.0. In order to make a strong statement that one sample mean is higher than another, their CIs should not overlap [31]; if they do, it is possible that the true population means have the reverse relationship. When an experiment is analyzing a small effect (for example, a 5% performance improvement), a wide CI may invalidate the conclusion.

> ◊ ***Perform enough repetitions to achieve tight confidence intervals***
> Techniques from statistics provide robust foundations for making strong statements about performance differences between systems.

Statistical methods fit into two broad classes: parametric and nonparametric techniques. The former class, which is more well-known, relies on the assumption that the analyzed data stems from known probability distributions, typically the Normal/Gaussian distribution. A variety of closed-form expressions for statistics of interest enable powerful parametric analysis. In contrast, nonparametric techniques are used when the probability distributions are *unknown*, and require fewer assumptions. Nonparametric methods, which have fewer closed-form equations, involve less powerful counterparts of popular parametric techniques, e.g. the Kruskal-Wallis test [40] instead of ANOVA. In nonparametric analysis, empirical mean and standard deviation can be computed, but their interpretation is different compared to the parametric case: rather then using them to fit distribution curves, they reveal only high-level information about the shapes of population distributions. The two most common metrics of interest, the median and CI for the median, can be used to compare pairs or sets of sampled nonparametric distributions.

Many studies suggest that the normality assumption does not hold for the data obtained in computer systems experiments, especially when the data includes measurements of performance. This applies both on a single machine [34] and in parallel programs running on supercomputers [67]. Indeed, as we document in §4.3, most data in our dataset does not follow the normal distribution. Thus, we adopt nonparametric statistics for the remainder of this paper, and recommend that, for performance experiments, these methods be used unless normality can be demonstrated. In [27] and [13], the authors provide advice for statistically sound performance analysis and argue for applying robust nonparametric techniques.

In nonparametric analysis, one uses the median, rather than the mean, as the measure of central tendency. To get CIs for a set of measurements X, one first sorts X. Then (as described in [41]), compute $\left\lfloor \frac{n-z\sqrt{n}}{2} \right\rfloor$ and $\left\lceil 1+\frac{n+z\sqrt{n}}{2} \right\rceil$, where n is the number of elements in X, and z is the *standard score* (or *z-score*) [31]. z depends only on the desired confidence level, and is 1.96 for the commonly-used level of $\alpha = 95\%$. These two numbers are then used as indexes into the sorted X: the values at those locations are the top and the bottom bounds of the CI. Note that one of these numbers will be larger than the median (at index $\lfloor \frac{n}{2} \rfloor$) and the other will be smaller, and they will not necessarily be symmetric around the median. These bounds tend to get tighter—to approach the sample mean—with more repetitions. Typically, we are concerned with the relative difference r% between the CI bounds and the mean.

A natural question is how many repetitions of an experiment are likely to be needed to achieve a sufficiently narrow CI (e.g., indicating that the empirical median differs from the true median by no more than $r = 1\%$) for a given confidence level α (e.g. 95%): we want to be sure to run enough repetitions to be confident in our results, but don't want to waste time running more than necessary. We use $E(r,\alpha,X)$ to represent this value for a set of experiment results X. The value of E can vary widely depending on the data in X; intuitively, the more variation between measurements in X, the more runs it will tend to take to narrow the CI to the target of r%. So that we can compare values of E to each other, for the remainder of this paper we adopt $E(1\%,95\%,X)$ as our standard target and denote it as $\check{E}(X)$. It is important to note that this is an *estimate* of what is required to get the desired confidence: empirical CIs must still be computed from the data gathered.

Finding $\check{E}(X)$ for parametric models is straightforward, as most such models have a closed-form equation that uses an estimate of the variance of X, obtained by running

a handful of exploratory experiments. In the nonparametric case, this number is harder to find, since we cannot make any assumptions about the distribution and there is therefore no equation we can use. One of the major contributions of this paper, covered in §5 is a resampling-based technique for estimating $\check{E}(X)$ for nonparametric models, and a tool we have built that makes it easy for experimenters to get these estimates.

3 Methodology

Over a period of 10 months, from May 20, 2017 to April 1, 2018, we collected performance measurements on servers that are part of the three CloudLab [60] clusters. Our experiments were run while servers were *not* allocated to other users, meaning that they did not affect, nor were they affected by, other users of the facility.

3.1 Testing Framework

Our testing framework is built with `geni-lib` [5], a Python library for interacting with GENI-compatible testbeds such as CloudLab. We wrote an orchestration script which selects free servers, runs benchmarks, and collects the results. In order to avoid consuming excessive resources on CloudLab, this script runs at a fixed interval every six to eight hours on each CloudLab cluster. Three to five servers (depending on the size of the cluster) are selected by fetching a list of the target cluster's available servers, checking them against our database of previous runs, and prioritizing never-tested servers, followed by least recently tested servers. Servers that have had a recent failure are not re-tested for a week to avoid having them remain at the highest priority.

Once the test servers are provisioned, the orchestration script waits for the provisioning process to be completed, logs into the server, and automatically runs the tests (described below). A single run can take between 30 minutes and 5 hours; the majority of this time is spent running disk tests.

As a side effect of the way that the CloudLab allocation policies and usage patterns work, servers were not sampled uniformly: some servers were unavailable for up to months at a time, as they were part of long-running experiments. In general, the more popular the type of server, the more sparsely sampled it is. Times of heavy testbed utilization, such as major deadlines, are also sparsely sampled. This requires us to use analyses that are robust with respect to different sample sizes.

3.2 Benchmarks

We selected a set of benchmarks to cover three key resources: memory, storage, and networking. In our selection of benchmarks, we struck a balance between observing the performance of the hardware when pushed to the limit (to detect degraded performance), and what might be seen in a more typical application (to understand "typical" behavior of the hardware). Hyper-optimized benchmarks can often come at the expense of practicality, and often make use of instructions, settings, and "tricks" that are limited to specific processors or I/O devices. We also required benchmarks that were portable across different architectures, due to the presence of both `x86-64` and `ARM` machines in CloudLab. Our primary benchmarks follow both principles, and we have some supplementary x86-specific benchmarks that use intrinsics to maximize performance. Memory and storage results have been collected since the beginning of our study, and we started collecting network benchmarks about 6 months later.

Memory We use two different benchmark suites for our memory tests. First, STREAM [43] (a standard benchmark for HPC machines) gathers a simple set of single-threaded and multi-threaded micro-benchmarks that perform basic operations such as memory copies and simple mathematical manipulation of memory contents. Second, we use a suite of micro-benchmarks by Alex W. Reece [51, 50] for supplemental non-portable tests utilizing Intel x86 intrinsics such as SSE and AVX. We found that, while absolute numbers differed, these other benchmarks did not alter our conclusions, so we discuss only the STREAM benchmarks in this paper. All tests use sufficient memory to minimize caching effects.

While we made no modifications to any timed portions of the benchmarks, we did modify both benchmarks to provide more complete reporting of statistics at the end of their runs. In addition, we altered the overall STREAM workflow to run a single-threaded test followed by a multi-threaded test. In the case of Intel processors, we run tests both with a standard frequency-scaling setting and with a setting that disables turbo boost and sets the performance governor to "performance." In the case of multi-socket machines, we test on each socket independently using `numactl` to avoid bottlenecks with QPI. Both memory benchmarks are built from source during each run using `gcc` with exactly the same compile flags every time; this helps with our multi-architecture environment, and means that `gcc` applies the optimizations appropriate to that environment.

Storage We test storage by using `fio` [3] to issue direct 4KB asynchronous I/O requests to target raw block devices. For the boot device, we run `fio` on the partition of that device containing the remaining empty space. Otherwise, we run `fio` on the entire device. We test both sequential and random reads and writes independently, and each workload is run both with a high and low number of I/Os issued to the device at any given time. A low I/O depth (we use 1) is more sensitive to device latency, whereas a high one (we use 4096) is more sensitive to bandwidth and internal parallelism. In the case of SSDs,

Type	#	Model	Processor	S	C	RAM	Boot Disk	Other Disks
m400	315	HPE m400	ARM64 X-Gene	1	8	64 GB (8x4)	SATA III SSD	None
m510	270	HPE m510	Xeon D-1548	1	8	64 GB (4x8)	NVMe SSD	None
c220g1	90	Cisco c220m4	Xeon E5-2630v3	2	16	128 GB (8x8)	SAS-2 HDD	SAS-2 HDD &
c220g2	163	Cisco c220m4	Xeon E5-2660v3	2	20	160 GB (8x10)	SAS-2 HDD	SATA III SSD
c8220	96	Dell C8220	Xeon E5-2660v2	2	20	256 GB (16x16)	SATA II HDD	SATA II HDD
c6320	84	Dell C6320	Xeon E5-2683v3	2	28	256 GB (16x16)	SATA II HDD	SATA II HDD

Table 1: Server configurations. "S" is the number of sockets, and "C" is the total core count (across all sockets). RAM is described as "(DIMM size x # DIMMs)". SAS-2 HDDs are all 10k RPM, and SATA II HDDs are all 7.2k RPM.

we issue a TRIM to the device using `blkdiscard` before we run any write workload. This clears certain block state, allowing for more efficient write operations [26]. We install `fio` from the Ubuntu package repository.

Network For each site, we set a fixed destination server that every server runs network tests against over a shared VLAN. For latency tests, we use a simple ICMP `ping` in flood mode. For Bandwidth tests, we use `iperf3` [30] with TCP and take measurements bidirectionally. Some of servers we test are rack-local with the destination server, and others require multiple layer-2 hops. Since CloudLab makes its topology public, we know that all non-local servers we are testing are three to four Ethernet hops away, and we record switch-path information along with each test. We install `iperf3` from the Ubuntu package repository, and `ping` is already bundled with the base operating system.

3.3 Servers Tested

We gathered our results from CloudLab's three primary clusters: Utah, Wisconsin, and Clemson. Servers at each site are divided into a small number of distinct homogeneous *types*; no sites currently have overlapping types. All servers we tested are interconnected via a 10 Gbps "experiment" network within each site. At the time of our tests, each of these sites had two "dominant" types consisting of tens to hundreds of servers. Some sites have types with only a few instances containing specialized hardware such as GPUs or many disks; we did not test these types to avoid consuming CloudLab's scarcest resources.

A summary of the server types we tested can be found in Table 1. The two Utah types are designed on the low-power and high-density Moonshot platform from HPE, with 45 servers in each 4U chassis. The two Clemson types are somewhat less dense, with four to eight servers per 2U chassis, while the Wisconsin servers are in independent 1U chassis. The Wisconsin servers have the most disks, with each server having a boot HDD, plus one "extra" HDD and SSD each. More detailed information regarding the experimented-upon server types, such as specific component models, can be found on the CloudLab Hardware documentation pages [61, 59].

3.4 Software Consistency

While we focus on hardware-based variance in this paper, we recognize that software differences can have a major impact on performance. To this end, our testing framework tracks, for each test, the version information of the kernel, versions of key packages (such as the compiler), and the revision of our repository containing our test script and memory benchmark sources. The key software remained at the same version throughout this test: the operating system release (Ubuntu 16.04, standard CloudLab image), the Linux kernel release (`4.4.0-75-generic`), `ping` (`iputils-s20121221`), and `iperf3` (`3.0.11`). While our testing repository was updated several times over the testing period, no modifications were made to any timed areas of our memory benchmarks. Finally, almost all runs utilized the same `gcc` version (`5.4.0`) and `fio` version (`2.2.10`). A very small percentage ($< 1\%$) of our runs used slightly earlier versions of both `gcc` and `fio`, so to maintain software consistency we excluded them while performing our analysis.

CloudLab released disk images with mitigations for Spectre and Meltdown (which are known to affect performance) on April 2, 2018; we intentionally use data through April 1 so that we can focus on hardware variance in this paper. We are continuing to collect data, and expect variance due to system software to be an interesting topic for study in its own right.

3.5 Resulting Dataset

From the period of May 20th 2017 to April 1st 2018, we collected 10,400 total runs from 835 total machines. A complete breakdown of machines tested and runs by hardware type can be found in Table 2. Since each run involved execution of a multitude of benchmarks in different configurations, we ended up with a total of 892,964 distinct data points over this period.

We use the term "configuration" to refer to the combination of hardware type, configuration, and benchmark settings. For example, the possible memory configurations come from varying hardware type, socket number, single- or multi-threaded operation, frequency scaling, and type of memory operation; this results in 590 possible configurations for memory. Similarly, there are

CloudLab Site	Hardware Type	Tested/ Total Servers	Total Runs	Mean/ Median Runs
Utah	m400	223/315	3583	16/8
	m510	221/270	2007	9/7
Wisconsin	c220g1	88/90	800	9/7
	c220g2	125/163	1527	12/8
Clemson	c8220	96/96	1742	18/12
	c6320	82/84	741	9/8
Total		835/1,018	10,400	12/8

Table 2: Coverage of our dataset

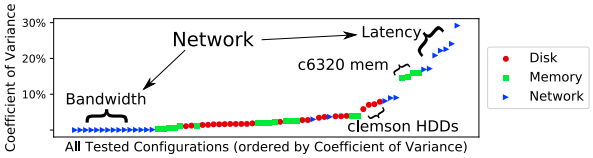


Figure 1: CoV for a variety of configurations.

96 possible configurations for storage, and 27 possible configurations for network tests. Each data point in the dataset comes from executing one configuration.

4 Understanding Variability

We begin our analysis of the dataset with an exploration of some of the key statistics computed from it. All data used in this section has had measurements from servers that are outliers removed, so it represents the unavoidable variation that experimenters must cope with even when the service provider does its best to provide consistently-performing servers. The procedure that we developed for removing unrepresentative servers is described in §6.

4.1 Unavoidable Variability

We first use our dataset to answer questions of importance to experimenters and other users who need consistency from the platforms they use. These stem from the basic question "How much variability must I account for in my experiments?"

Aiming to perform fair high-level assessment, we select a subset of 70 benchmark × hardware combinations with relatively even distribution: 24 disk (all for boot devices), 19 memory (variants of *copy* benchmark), and 27 network (both latency and bandwidth) configurations being tested. We use *coefficient of variance* (CoV), the ratio of the standard deviation to the mean, to compare these configurations; absolute *standard deviation* cannot be used here due to the difference in the scales and units of compared measurements. Displayed in Figure 1 and ordered by CoV, the analyzed configurations reveal the following insights:

Networking Both top and the bottom of the list are dominated by the network tests: primarily, latency tests are at the top with CoV in the range $[16.9\%, 29.2\%]$, while the bandwidth tests are at the bottom with CoV $< 0.1\%$. For the configuration with the largest CoV, we notice that the standard deviation is $7.7\mu s$ is quite small in absolute terms. However, it is a significant fraction of the empirical mean for the latency at $26.3\mu s$. This seems to stem from a couple of sources: First, we are using standard, unoptimized, tools to measure latency, and the timescales are small enough that effects within the kernel networking stack are noticeable (even loopback `ping` displays some variation). Second, the granularity of timestamps reported by `ping` is sufficiently coarse ($1\mu s$) that measurements group into discrete bands instead of being continuously distributed. In contrast, the 3.3×10^5 standard deviation on most bandwidth tests corresponds to only 330kbps out of the median of 9.4Gbps. We note that CloudLab allocates network bandwidth in such a way as to attempt to guarantee each experiment the full bandwidth it has requested, free of interference from other users. The low CoV in bandwidth tests suggests that it is effective in doing so. Not all datacenters have similar bandwidth allocation policies, and this result may vary in a different setting.

c6320 Memory A block of memory tests for the `c6320` servers stands out for having higher CoVs than other memory tests: they are in the range $[14.5\%, 16.0\%]$. There is no clear cause for this variability, but it is remarkable for being the only set of configurations for which a particular type of server is tightly grouped. The lesson we take from this is that it it underscores the need to test all configurations rather than assuming that similar types of resources exhibit similar variability.

Clemson HDDs Clemson servers stood out in another way as well: the HDDs on both Clemson types show moderately high CoV for high-iodepth random I/O for both reads and writes. These disks (which are the same model on both types) are the only 7.2k RPM HDDs in CloudLab, as well as the only SATA HDDs.

Bulk of the Tests The remaining block of 44 tests consists of intermingled disk and memory configurations. CoVs for this set of tests are in the range $[0.3\%, 9.0\%]$. While this is a fairly wide range, there is no clear pattern within it; for example, the data does not support the hypothesis that disk bandwidth consistently exhibits more variability than memory or vice versa. Also, unlike the results for `c6320`, individual server types show no grouping, leading us to the conclusion that, on the whole, there is little correlation between server type and CoV. Based on this analysis, we expect CoV for hardware performance metrics observed in practice to be roughly in this range most of the time and, in rare cases, exceed the 10% mark.

HDDs@c8220	HDDs@c220g1	SSDs@c220g1
6.85% (rr, H)	5.66% (r, L)	9.86% (rr, L)
6.42% (rw, H)	3.68% (rr, H)	5.38% (r, L)
6.08% (rr, L)	1.93% (r, H)	4.65% (rw, L)
5.82% (r, L)	1.90% (w, H)	3.95% (w, L)
5.32% (rw, L)	0.99% (rw, L)	1.00% (w, H)
4.96% (w, L)	0.93% (rr, H)	0.68% (r, H)
1.27% (w, H)	0.58% (rr, L)	0.53% (rw, H)
1.20% (r, H)	0.14% (w, L)	0.09% (rr, H)

Table 3: Coefficient of Variance. Values are annotated with the type of test and iodepth: *r*ead, *w*rite, *r*and*r*ead, *r*and*w*rite. "L" and "H" denote iodepth 1 and 4096.

> ♢ ***Some amount of variation is unavoidable***
> Some degree of variation in hardware performance is unavoidable, no matter what steps the facility provider takes to provide consistent hardware. Coefficients of Variance of up to 10% may be attributed to hardware variability and considered expected, while higher values may indicate room for improvement from the measurement standpoint.

For the aforementioned CoV range, we determine that the configuration with CoV $= 0.3\%$ is likely to require only $\check{E}(X) = 10$ experiments in order to make the corresponding CI sufficiently small. In contrast, this number significantly increases, up to $\check{E}(X) = 240$, for the configuration with CoV $= 9.0\%$. This demonstrates the need for careful experiment design that takes into account variation of the specific resources that are exercised, and we present further analysis of the relationship between the two metrics in §5.

4.2 Disk I/O

SSDs are well-known to have complex performance profiles [26] due to the write characteristics of flash and their internal Flash Translation Layers (FTLs). In addition, different types of HDDs have performance characteristics based on their rotational latency, attachment protocol, density, etc. We wanted to answer the question "Are SSDs more consistent (lower CoV) than HDDs?", so we look at the variability for two different types of HDDs (one model at Wisconsin and one at Clemson), and for SSDs at Wisconsin. The devices at Wisconsin are higher-end: 10k RPM SAS-2 HDDs and enterprise Intel SSDs, while the HDDs at Clemson are 7.2k RPM SATA III devices.

As shown in Table 3, the answer to our question depends on the level of parallelism (iodepth) and the type of HDD. With high iodepth, SSDs use their internal parallelism and demonstrate both much higher performance and more consistency. The SSDs we tested are 2.3–2.4

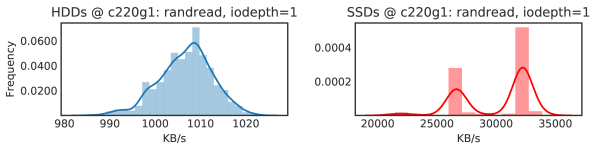


Figure 2: Histogram of iodepth=1 randread on c220g1.

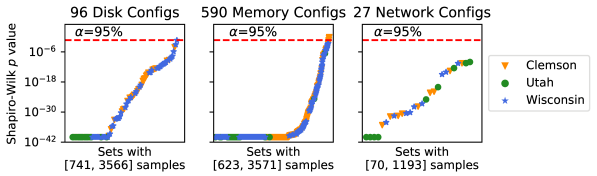


Figure 3: Testing normality of the collected data.

times faster on sequential tests than HDDs, and from 82.5 up to 262.3 times faster on random reads and writes. CoVs for these tests were in the range $[0.09\%, 1.0\%]$ for SSDs, lower than most HDD CoVs.

On HDDs, unsurprisingly, iodepth is not strongly correlated with CoV: these devices have less internal parallelism, and it is harder to exploit due to the lack of an abstraction layer as complex as the FTL. Because SSDs have such high CoV on low-iodepth tests, some HDDs are competitive in terms of CoV (if not absolute performance). The reason for this can be seen in Figure 2, which examines the case of random reads. HDDs have a performance curve that is fairly compact: it is dominated by seek time and rotational delay, and roughly bounded by the maximum values of those two variables. This curve is more compact for the higher-RPM SAS dives at Wisconsin, which have lower CoV for most low-iodepth test than the SSDs. The SSDs that we tested, on the other hand, exhibit a bimodal pattern; the exact underlying cause is difficult to ascertain because of the opaque nature of the vendor's FTL, but the effect on experiments is clear and dramatic. The lower-RPM SATA HDDs at Clemson are less competitive against the SDDs in terms of CoV; this is likely due, in part, to their higher rotational latency.

4.3 Testing For Normality

The statistics commonly used to analyze the performance of computer systems [31] tend to assume that measurements are normally distributed. We use the Shapiro-Wilk test [55] to test for normality in our dataset, and find that benchmarks of individual configurations *across different servers* are **not** normally distributed. We apply this test to all configurations and show our results in Figure 3. Each point, shown in the order of increasing p-values, characterizes samples for a specific configuration. For points above the threshold, we cannot reject the *null hypothesis* (stating that the samples come from populations which have normal distributions). For points below this

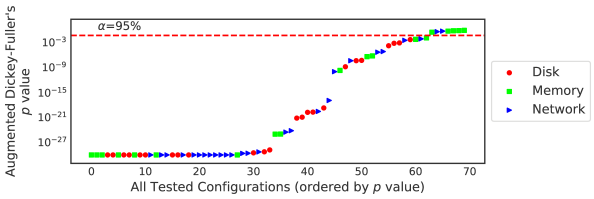


Figure 4: Testing stationarity of the collected data.

threshold, we reject the null hypothesis at this confidence level (95% in this figure), and assume non-normality. Our analysis shows that we should reject the null hypothesis for over 99% of the configurations (710 out of 713). Intuitively (and confirmed by inspecting the underlying data), when we measure maximum bandwidth of a device, there is a practical maximum that cannot be exceeded except by measurement error, and most measurements lie near this maximum. On the other hand, some measurements are significantly lower than the maximum, leading to a skewed distribution with a compressed range above the median and a much larger range below it. The situation is reversed for latency tests. Considering the large number of samples in the analyzed configurations, from 70 in the smallest up to 3,571 in the largest, we reject the normality hypothesis for tests across servers; hence, our focus on nonparametric analyses in this paper.

◊ ***Use nonparametric confidence intervals to avoid assumptions of normality.***
Many computer systems performance results have skewed distributions (longer tails on one side); nonparametric confidence intervals are simple to compute, and work for these distributions (as well as normally-distributed results).

We also test normality for sets of data points that are all drawn from the *same server*. We filter data by selecting servers with at least 20 data points coming from memory tests (this number coming from [55]). Given the way we schedule tests, many servers have not executed more than 20 tests and thus this subset corresponds to 42,680 data points. After applying Shapiro-Wilk to this subset, roughly half of the points (26,695) can be considered to be coming from a normal distribution. Intuitively, we can assume normality in this subset because data points are obtained by running a configuration on the same machine, that is, the hardware and software are the same for all points. This suggests that experimenters should proceed with caution when analyzing results from a single server: data *may* be normally-distributed and thus suitable for analyses that assume normality, but a test such as Shapiro-Wilk should be run to confirm or deny this assumption.

◊ ***For some configurations, single-server tests can be assumed to be from a normal distribution.***
Evaluating normality for tests run on a single server can simplify the analysis since parametric statistics can be employed for these single-server results.

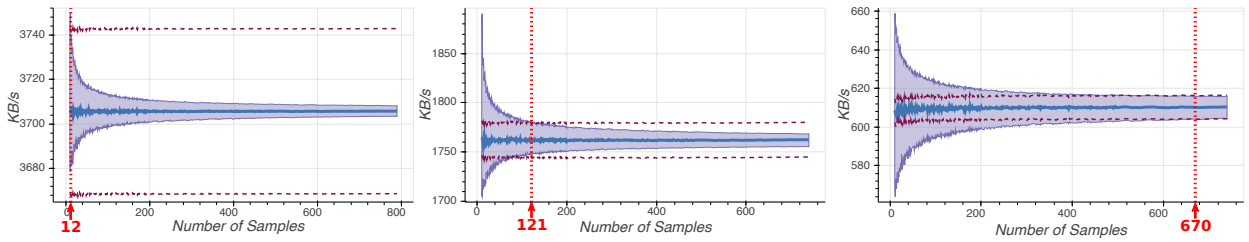
4.4 Checking Stationarity

Most statistical tests—including confidence intervals—assume *stationarity*: that is, that the properties of the underlying distribution (such as median and variance) do not change over time. In addition to affecting data analysis, non-stationary distributions would harm reproducibility: if performance is not stable over time, future experiments cannot reliably be compared to past ones. We use the Augmented Dickey–Fuller (ADF) [15] test to check for stationarity in our data.

For all 70 configurations shown in Figure 1, we run ADF and get a range of p values allowing us to accept or reject the non-stationarity null hypothesis in each case. These values, shown in Figure 4, indicate that nearly all of the analyzed datasets present strong evidence for stationarity: we can reject the hypothesis that they are non-stationary with the confidence level $\alpha = 95\%$ for all points below the line. Among the handful of non-stationary cases (above the line), we find several memory (*copy* benchmark run on `c220g1`) and network bandwidth (also run on `c220g1`) tests. Among the evaluated disk tests there is more tendency towards non-stationarity in the tests with iodepth $= 1$. Recall that our measurements are not sampled from servers uniformly, as described in §3. This appears to be a cause of some of the non-stationary patterns we observe: during some periods, certain servers are over-sampled, and, as they are slightly outside the mean for the whole population, this produces a temporary shift in the mean. These effects could be visible to CloudLab's users, since during periods of heavy utilization, users frequently creating and terminating experiments could see the same set of servers repeatedly. Our remedy to this, detailed in §6, is to find and remove servers that have significant statistical departures from the rest of the population.

5 CONFIRM: How Many Measurements Are Enough?

Given that some amount of variability is inevitable, we turn to a perennial question for experimenters: "How many repetitions do I need to run in order to be confident in my results?" As described in §2, given a set of measurements and a desired confidence level (such as 95%), we can compute a confidence interval (CI) for the mean or median. A standard procedure is to "invert" this calculation, and for a given desired confidence level and CI



(a) 88 HDDs at Wisconsin (c220g1), random reads, iodepth=4096

(b) 82 HDDs at Clemson (c6320), random reads, iodepth=4096

(c) 82 HDDs at Clemson (c6320), random reads, iodepth=1

Figure 5: Nonparametric confidence intervals produced by CONFIRM. As the number of samples grows, 95% CIs (filled areas) for the medians (thick blue lines) shrink and fit within the 1% error bounds (dashed lines). This stopping condition is depicted with red lines and annotated with the numbers of recommended measurements $\check{E}(X)$.

width, estimate how many repetitions are likely necessary to achieve the desired confidence.

When assuming normality, there is a closed-form equation to calculate this estimate [31]; the main input to this equation is an estimate of variance, typically obtained by running a small number of trial runs. In the nonparametric space, there is no closed-form equation, so producing such an estimate requires a more complex technique. We have developed such a technique using *resampling*:

For a set of collected measurements X with n values, we randomly select a subset of $s \leq n$ values for which we estimate the bounds of the CI for the median as described in §2. We shuffle X, select another subset of s values, and obtain new estimates of the CI. After we repeat this process c times, we calculate the means of the lower and upper CI bounds. Obtained using *sampling without replacement*, each of these random selections or "trials" represents a hypothetical scenario where a smaller, partial subset of measurements was collected by an experimenter. The aforementioned averaging eliminates the dependence of the results on the properties of a particular subset and provides an aggregate view on the convergence of the CI observed across many trials. The results presented in the rest of the paper are obtained using $c = 200$. To estimate the recommended number of measurements $\check{E}(X)$, we start at $s = 10$, assuming that smaller subsets are insufficient to estimate nonparametric CIs reliably and should not be considered. Then, we increase s until $s = n$ or the mean CIs fit within the desired error bounds. In the former case, we conclude that these n samples are insufficient for meeting the stopping condition, while in the latter case, we note that the experimentation could have stopped after $\check{E}(X) = s$ measurements according to the selected allowed error and confidence level.

We have implemented this technique in a service we call CONFIRM or CONFIdence-based Repetition Meter. This dashboard imports our benchmarking datasets and facilities interactive nonparametric analysis of CIs for measurements collected from individual servers, groups of servers, and entire hardware types available on CloudLab. We present three analyses here to demonstrate how CONFIRM can help guide experimentation: looking at the how variability affects experiments on different types of HDDs; quantifying how much a single outlier can increase the number of repetitions that must be run; and looking at the relationship between variance and $\check{E}(X)$.

HDD Variation In the first set of experiments, we compare 88 HDDs at Wisconsin with 82 HDDs at Clemson, revisiting results from §4.2 from the perspective of variation. CONFIRM produces visualizations of the CIs and the $\check{E}(X)$ estimates that are depicted in Figure 5. Figure 5 (a)-(b) show the difference of over 10× in $\check{E}(X)$ for two disk types running the same benchmark (random reads, iodepth = 4096), with Clemson disks exhibiting higher variance and wider CIs. A similar benchmark—random reads, iodepth = 1—demonstrates an even more severe case, as shown in Figure 5 (c). For the same set of Clemson HDDs we have to use as many as 670 samples (almost all of the measurements we have collected) in order to fit the CI within the same 1% error bounds. If we were to select a set of servers based on reproducibility of disk-heavy workloads, the Wisconsin servers would be the clear choice; conversely, if our experiments must be run on the Clemson servers, we will need to be careful to run many repetitions to get statistically significant results.

Effects of Outliers In the second set of experiments, we start with a randomly selected set of 9 c220g2 servers at Wisconsin, add one more "badly" performing server of the same type (one that will be eliminated using the method in §6), and analyze CIs for memory tests with and without this outlier. We run CONFIRM on the combination of the selected servers and four variations of the *copy* memory test and record obtained $\check{E}(X)$ estimates in Table 4. We can see that inclusion of this server results in a 2.1–5.9× increase in the recommended number of repetitions. Our analysis shows that the distribution of the performance data obtained on these 10 servers is highly skewed, with the "long tail" caused by the low-

Memory test / frequency-scaling / tested socket	9 servers	10 servers (same 9 + 1 "outlier" server)
copy / no / 0	18	63
copy / no / 1	10	58
copy / yes / 0	33	68
copy / yes / 1	10	54

Table 4: Recommended number of measurements $\check{E}(X)$ for 9- and 10-server sets. Estimates are produced using CONFIRM for Wisconsin c220g2 servers.

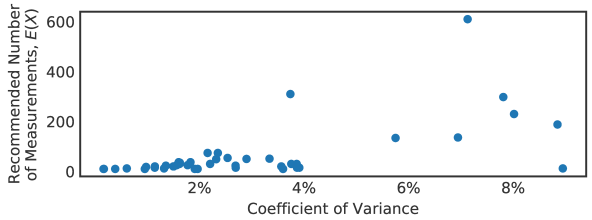


Figure 6: Relationship between CoV and $\check{E}(X)$.

performance measurements. In this and similar cases, not only we are less confident about the value of the statistic of interest—in this case, sample median—but we are likely to make poor conclusions using insufficient number of measurements. Thus, further analysis of the data shown in Table 4 confirms that if we stop after 10 measurements in the 10-server case, our reported median values will be *outside* of the 95% CIs around the medians reported after the recommended 58-68 measurements.

> ♢ ***Use low-variance hardware whenever possible***
> The higher the performance variance of the underlying hardware, the more repetitions must be run to establish statistical significance; conversely, if not enough repetitions are run, there is a greater chance that the conclusions are incorrect.

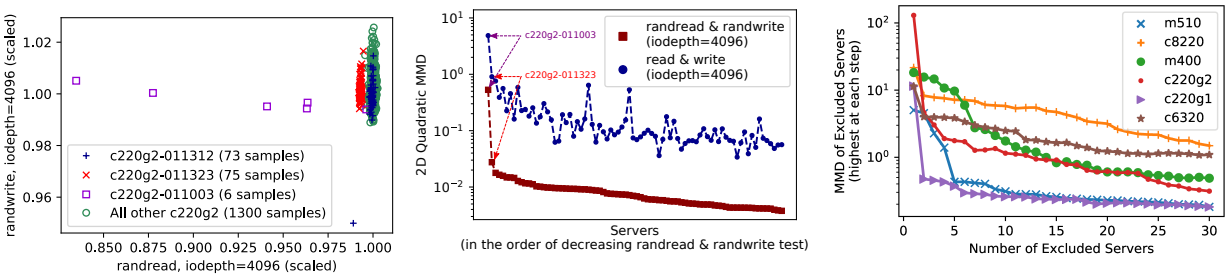
CoV vs. $\check{E}(X)$ Figure 6 shows the relationship between the CoV and the number of repetitions recommended by CONFIRM for the bulk of the configurations from §4.1. This figure is generally favorable for experimenters: most configurations up to about 4% CoV require only tens of repetitions to reach the target of $r = 1\%$ for CIs. Some configurations, however, are extreme outliers, requiring hundreds of experiments to reach this level of confidence. These outliers do not show a consistent pattern in either the type of configuration nor the relationship between CoV and $\check{E}(X)$. The reason that the CoV and $\check{E}(X)$ are not perfectly correlated is that they react differently to outliers and multi-modal distributions. Outliers can skew means and standard deviations quite a bit, but the median is less sensitive to them, and nonparametric CIs effectively take into account the *presence* of points outside the CI but not their *magnitudes*. For extreme multi-modal distributions, such as the one seen in Figure 2, the mean and standard deviation have no problem computing values "in the middle" where no points actually lie, but the median and nonparametric CIs can only pick from points actually in the dataset, making it take much longer for them to converge—or preventing them from converging at all. This figure shows the importance of a tool like CONFIRM: our intuitions about variance, confidence, and the number of repetitions are not always correct, and actual measurements are needed to inform rigorous experiment design.

> ♢ ***Base experiment design on past measurements***
> The relationship between variance and the number of repetitions required is complex; good estimates of the latter require significant prior data.

Using CONFIRM We run CONFIRM as a service at `https://confirm.fyi/` to help users of CloudLab plan their experiments. The tool itself is open-source, so it can be applied to any other facility for which similar data can be collected. We note than when using CONFIRM to estimate the number of repetitions needed for an experiment, it should be used as an *initial* estimate, by selecting the resource(s) that the performance of the experiment is most likely to depend on. Once data is collected, empirical CIs should be computed for the collected data (as described in §2) to ensure that the target allowed error range has been met; the level of variability in a higher-level system may be higher or lower than those found in the low-level benchmarks that CONFIRM uses to compute its estimates.

6 Detecting Unrepresentative Servers

We now turn our attention to the provider's perspective: given what we have seen about the effects of variance on users, what can a provider do to provide resources with consistent performance? As we have seen, some variance is unavoidable, so we pursue the goal of having a *set of servers where every server is representative of the whole*. Put another way, in a distribution drawn from all servers, if we draw samples from a particular server, we should not be able to distinguish those samples from the complete population. This is a strong analysis, as it gets directly at the goal of a testbed or service provider that it should not matter which server(s) an experiment uses: all should provide results that are statistically indistinguishable.



(a) Identifying outlier servers. All values are normalized by the medians.

(b) Ranking servers based on 2D MMD. First two – servers with consistent performance degradation shown in (a).

(c) Using 8 benchmarks (4 disk + 4 memory), we exclude unrepresentative servers for each of the hardware types.

Figure 7: MMD-based server evaluation for c220g2 (a-b) and outlier elimination for all tested hardware types (c).

> ◊ ***Provide indistinguishable resources***
> When servers—even those that are supposedly identical—exhibit performance differences that can be detected reliably through statistical tests, reproducible experimentation is more difficult.

Statistical distributions can be compared based on independent samples using the Mann-Whitney U-test [42]. Unlike its parametric counterpart, the *t*-test, the nonparametric U-test does not assume normality of the compared distributions. As reviewed in [33], many authors have focused on this problem and offered various sophisticated approaches. Appearing in the recent machine learning literature, a kernel[1] two-sample test based on *maximum mean discrepancy* (MMD) [24] offers a powerful solution that is suitable to large-scale datasets and naturally supports multivariate comparisons. This kernel-based testing can be summarized as follows:

The test compares samples $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ from distributions P and Q, where n and m do not need to be equal. No assumption is made about P and Q, and the robustness of this test with different n and m is important for our setting, since we will be using it to compare the samples for an individual server to the rest of the population. MMD provides a measure of similarity (or dissimilarity) between P and Q, expressed as a distance between their embeddings in the reproducing kernel Hilbert space (RKHS) [6]. Abstract in its formulation, this distance metric is still straightforward to use in practice. Similar to many statistical tests, the univariate values obtained using MMD can be compared against thresholds calculated for a given confidence level α and used to estimate probabilities of P and Q being the same distribution given the analyzed samples. The test comes with the quadratic-time and linear-time (w.r.t. $m+n$) estimation variants. The former is a more powerful test as it uses every measurement to the maximum effect, while the latter is more suitable to online processing where the analysis is performed as the data becomes available.

We use the quadratic test implemented in Shogun [57], an open-source machine learning library for Python. One important aspect of MMD testing is kernel selection: we chose a meaningful range of kernel parameters and found that the results of our analysis are not sensitive to particular parameters selected, so we use a common smooth kernel function, a Gaussian kernel,[2] with the bandwidth parameter $\sigma \in [5\%, 50\%]$ of the analyzed measurements. Designed to be robust to individual outliers, MMD tests can point out distribution differences, including pronounced skew and frequent outliers, that are statistically significant.

Based on the MMD statistic, we develop the following method for identifying unrepresentative servers:

Use multiple benchmarks to characterize servers of a particular type. To increase robustness to outliers and avoid bias caused by uneven magnitudes of values in different dimensions, we divide all values by the medians in each dimension prior to kernel testing. Figure 7 (a) demonstrates how such scaled data looks for two disk benchmarks (random read and write tests). In this figure, it is possible to visually identify outlier servers, but it would not be possible to eliminate the outliers cleanly using a simple threshold as the observed distributions overlap (for red and green clusters). In this case, we notice two servers that are unrepresentative—with a small consistent degradation (red) and a larger spread of outlier-like measurements (purple)—in one of the dimensions and a representative server with a single outlier (blue) in the other dimension.

Rank servers: Using the selected benchmarks, we run MMD tests that compare an individual server's samples against samples from all other servers of the same type. This statistic, which represents a measure of dissimilarity,

[1] A *kernel* or a *kernel function* in this context refers to the dot product of features of compared objects.

[2] Gaussian kernel functions facilitate comparison of non-Gaussian distributions and detect differences between multivariate clusters.

is the highest for the least representative servers. In the disk example, the unrepresentative servers end up at the top of the sorted list, as shown in Figure 7 (b). We also observe an expected yet nontrivial result: the same procedure with two different disk benchmarks (sequential tests instead of random), points at performance issues with the same two servers. The exact server ordering in the ranking that uses these sequential tests would be different, but both rankings demonstrate the same elbow-shaped decreasing pattern. At the same time, we confirm that the single-outlier server (blue in Figure 7 (a)) does not show up at the beginning of either ranking as the majority of its samples appear unquestionable.

Eliminate consistent outliers: Actionable insights provided by these dissimilarity rankings allow us to exclude the least representative servers from the pool available to users. We remove them iteratively, one at a time, starting with the least representative server; this ensures that the MMD statistics for the remaining servers are not skewed by the inclusion of the removed servers. Results obtained during such elimination are shown in Figure 7 (c). The elbow-shaped curves indicate that the largest reduction of dissimilarity comes from excluding a few servers at the beginning: from two to seven, representing only 2% of the overall population. Subsequent server elimination provides diminishing returns (note the log scale of the figure).

We have tested this elimination procedure in a variety of settings—in 2D, 4D, and 8D, with each "dimension" being a different configuration—and conclude that the described procedure helps identify the servers with nontrivial performance abnormalities for all analyzed hardware types. The MMD statistic that this test uses is abstract, and does not directly correspond to units in the original space (Gbps, μs, etc.), but this is a necessary side-effect of simultaneously testing metrics that are measured with different units and have different scales; nonetheless, the shape of the curve makes it very clear which servers are not representative. Testbed or service providers can use this procedure to investigate the most unrepresentative servers and take appropriate actions. This method can also help users understand how representative or unrepresentative the servers they use are by revealing their ranks within relevant populations.

7 Steering Clear of Pitfalls

While performing analyses, we ran into situations that resulted in surprising or counter-intuitive results. The potential set of such pitfalls is large, and we have certainly not uncovered all of them, even within the CloudLab environment. However, we can recommend defensive practices that help steer clear of them and likely others.

7.1 ♢ Randomize experiment orderings

Unexpected differences appeared in the memory bandwidth measurements on the two server types at CloudLab Wisconsin: we expected similar results, but the older `c221g1` servers outperformed the newer `c220g2` servers by a factor of nearly 3 (about 36 GB/s versus 12 GB/s) in multi-threaded benchmarks. After a long search, we traced this problem to an unbalanced DIMM configuration in the `c220g2` servers: as a result of their larger memory, the first memory channels were populated with two DIMMs, while the others all had one DIMM. When we had the extra DIMMs removed from one of the `c220g2` servers, memory performance jumped to expected levels. This imbalance appears to interact poorly with a combination of Intel's memory-striping algorithms [28], Linux's allocation of pages in sequential physical order, and the nature of the STREAM benchmark. The result is that STREAM's memory appears to reside mostly or completely on one memory channel, preventing the benchmark from using the hardware's full bandwidth.

While tracking down the cause of this behavior, we found an even more surprising effect: the order in which we ran benchmarks had a dramatic effect on STREAM's performance. In the most extreme case, running a particular benchmark would cause subsequent STREAM runs (until the server was rebooted) to increase their performance by a factor of three, "recovering" approximately the expected performance. Though the exact mechanism behind this recovery is not clear, it appears that the way one benchmark allocates memory—both the size of the allocation and the specific pattern—has an effect on the other's layout on physical channels, so the order in which we run these benchmarks matters. This is an effect that we would not have noticed had we not tried a variety of benchmarks in different orders. Trying to predict ahead of time which orderings would reveal which types of effects would be fruitless; thus a good defensive practice is to **randomize the order of experiments to expose effects that they might have on each other.** Others [48, 45] have made similar observations for other benchmarks.

7.2 ♢ Check configuration sensitivity

The experience related in the previous section also raises another important question: should it be considered a "bug" for a facility like CloudLab to have hardware with an unbalanced memory configuration? Placing blame for the behavior is complex: In the Intel platform, this configuration of DIMMs is legal, but results in fallback to a lower-performance mode that is not widely known. Linux's physical page management policy could also be blamed: FreeBSD does not allocate physical pages sequentially and we found that it exhibits full memory bandwidth performance in this hardware configuration. Our memory benchmarks could also be considered to be at

fault: while they use sufficient RAM to avoid caching issues, they do not use enough to ensure that all DIMMs get exercised. A facility like CloudLab aims to provide servers that are representative of servers in the wider world, and this is a configuration that is not unique to CloudLab.

Ultimately, we believe the primary lesson is the fact that experiments are more sensitive to small details of specific configurations than is commonly acknowledged, and that both facility and user share responsibility for being aware of this sensitivity. The service provider should aim for the highest-quality resources possible. At the same time, it cannot be aware of every interaction between hardware configuration, system software, and workload. The best defensive practice for users is to **perform sensitivity analyses with respect to the hardware configuration**: run experiments on hardware with multiple configurations to understand the extent to which results depend on a particular configuration.

7.3 ♢ Match hardware and software

When we first ran the STREAM benchmarks on the Wisconsin and Clemson servers, we discovered variance that was much higher than we expected. This was because these servers are dual-socket NUMA machines, and STREAM is not NUMA-aware. Not only did this have a deleterious effect on average performance (lowering it 20–25%), but it had an even more pronounced effect on the CoV (raising it from about 80 MB/s to 8,000 MB/s—two orders of magnitude). This problem was simple to resolve: we bind STREAM to one socket at a time, and test each socket separately.

Despite the ease of resolution, this points to a larger problem in experimentation: mismatch between the properties of the hardware and what the software was prepared to handle. Bigger and faster are not always better when it comes to running experiments, and can be worse because they typically imply greater complexity. Experimenters should **carefully consider whether they need features like NUMA, hyperthreading, complex memory hierarchies, etc. before selecting servers that have them**. Using hardware with features not supported in software runs the risk of invalidating results by affecting absolute performance and causing variability that harms the ability to make solid claims backed by statistics.

7.4 ♢ Don't assume independence: check

It is tempting to treat repeated experiments as independent: that earlier experiments do not have an effect on the outcomes of later ones. This is not always the case; one particular instance of this seen in our dataset is the performance of SSDs. Figure 8 shows performance results from a single representative SSD on a `c220g2` server over a period of several months; a clear periodic pattern is present.

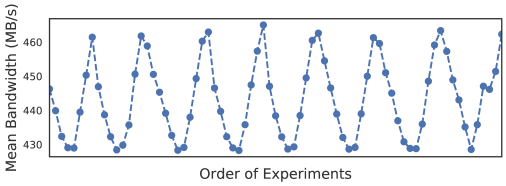


Figure 8: Periodic behavior on a `c220g2` SSD over time for sequential writes with iodepth 4096. Gaps between successive points can represent different durations of time.

Recall that we run `blkdiscard` before every one of these experiments: in theory, this should return the drive to a "clean" state. This periodic behavior seems to be present for two reasons. First, there is likely some sort of "lazy" process that does not do the work of `blkdiscard` all at once but saves part of it for later, resulting in noticeable performance artifacts. Second, this SSD does not seem to be heavily used by other experimenters (it is not the boot disk) so each time we run a new experiment, we are picking up where we left off in the disk's lifecycle.

The effect is that earlier experiments can affect later ones, such as through the quantity of data they write or where they write it, and this effect can persist many weeks later through multiple reboots. Effects may have been even worse if we had not run `blkdiscard`, since this would have left more FTL state from previous experiments. If we assumed independence between runs, we might very well come to incorrect statistical conclusions, as many techniques assume IID (Independent, Identically Distributed) results. This provides more motivation for randomizing the order of experiments, since the sets of experiments that affect one another is not the same for every run. To **test for independence**, we can compare the samples in their original order with with a shuffled version. These comparisons can be done using the Mann-Whitney test or the kernel-based MMD test, similar to the nonparametric two-sample testing we described in §6.

7.5 ♢ Be careful on shared infrastructure

Some experimenters, by choice or necessity, run experiments on virtualized resources in shared environments such as clouds. The most prominent issue with operating in a shared environment is the potential for the presence of "noisy neighbors," whose behavior can impact experimental results, and into which the experimenter has no visibility. Prior work [58, 49, 62, 8] has shown that workloads run by one tenant can affect other tenants in a shared environment. This has implications for variations on three different scales:

- Competing workloads increase variability during their runtime, affecting the *variability seen during individual experiments*.

- Competing workloads may come and go on timescales from minutes to days, causing experiments to get different results *on the same VM at different times*, or changing results during long-running experiments.
- Noisy neighbors may be more prevalent on some hosts than others, making *different VMs* perform differently.

This poses a problem for ensuring accurate experiment results—every bit of additional variance makes it harder to present results with high confidence. In some sense, the presence of a noisy neighbor is analogous to the addition of an "outlier" server as presented in Table 6. To get an intuition for how added variance can affect confidence in results, consider the data reported in Figure 5 (a): this configuration has a CoV of 1.0%, and requires 12 repetitions to achieve the desired confidence. A seemingly modest increase in CoV to 5.0% (Figure 5 (b)) results in a 10× increase in the number of repetitions required (to 121), and a further increase to 8.1% (Figure 5 (c)) requires 670 repetitions (55×). It is also important to note that these calculations assume a *stationary* distribution: that the distribution from which performance results are pulled does not change over time. Clearly, this is not the case with transient noisy neighbors, requiring even more careful experimentation techniques to detect and/or compensate for changing performance characteristics.

Studies have found high CoVs in commercial clouds [18, 29]—particularly for network and disk operations—and the long performance tails in clouds are well-known [14]. Farley et al. [18] found CoVs on EC2 from 0.35% to 25.4% for network bandwidth (average 4.4%), and from 0.5% to 40.9% (average 9.8%) for storage performance. They also found significant differences in performance (typically around 1.2×, but as high as 3.7×) from different VM instances of the same "type" (eg. `m1-small`). Compared with the CoVs found in this study—0.004% CoV for network bandwidth, and average 3.3% CoV (max. 9.86%) for disk I/O—experimenters are likely to require many more repetitions to gain high confidence.

Another issue with running on shared infrastructure is that virtualization adds a layer of abstraction. Even if there are no noisy neighbors to contend with, there is still the presence of the hypervisor. Other studies [49, 62, 56, 11] have explored the extent to which the hypervisor layer impacts the performance of various workloads, including increasing variance.

It is important to note that, as we have explored in this paper, running on non-shared, non-virtualized resources does *not* shield the user entirely from variability: even "bare" hardware has complex, opaque behavior, and the OS kernel can introduce variability just as the hypervisor does. The additional variance from shared resources does not make it *impossible* to run good experiments, but it can make it *much harder*. Earlier work has looked to address issues with running workloads in shared environments. Some solutions [46, 10, 7] focus on the perspective of the provider, and seek to manage these interference effects by varying virtual machine placement or resource allocation. Others [69] approach this from the perspective of the client and try to find the "best" type of virtual machine. The common thread between these solutions, however, is the reality that performance interference effects must be *managed* and cannot be entirely *avoided*. To achieve statistical confidence, the experimenter is likely to have to run many more experiments, and to consider sources of variation that are not stationary, which makes experiment design far more complex. Conversely, experiments run in this environment that do *not* account for increased and more complex variance run a larger risk of coming to incorrect conclusions: for a fixed number of runs, the more variance is present, the wider the confidence intervals. The wider the confidence intervals, the larger the effect that can be potentially misreported.

Our overall recommendation is to **run experiments in a shared (and therefore, likely high-variance) environment only if it is unavoidable.** If experiments must be run in such an environment, design them in ways that help compensate for variability: run many more repetitions, run on many different VMs and at different times to avoid over-measuring artifacts from particular neighbors, and ensure that the experiment design does not introduce systematic bias.

7.6 ♢ Plan experiments for uncertainty

It is not always practical to run a large number of repetitions of an experiment. This can be due to factors such as monetary costs, long execution times or both. Techniques in Active Learning [53] and Bayesian Optimization [54] help design sequences of experiments that efficiently "explore" available configurations. Generally speaking, the former class of techniques focuses on reducing the uncertainty about experiments' outcomes, while the latter helps find configurations corresponding to the maximums (or minimums) of the objective functions studied via experimentation. In contrast with classical (static) experiment design, these iterative techniques train Machine Learning models on the data available from existing experiments and use the recommendations produced by these models to run subsequent additional experiments. There is a wealth of literature describing optimizations for these techniques, including [20] and [36], as well as specific computer applications, such as [16], [22], and [4], among many other studies. While these experimentation techniques are mostly outside the scope of this study, as part of our future work, we intend to equip CONFIRM with the ability to recommend specific servers and specific

hardware and benchmark configurations for additional experiments on the basis of high performance variability and observed outliers.

8 Related Work

In [37], the authors present a profiling study of a Warehouse-Scale Computer where they analyze 12 to 36 months worth of performance counter metrics for applications running on Google data centers. The study focuses on microarchitecture-level statistics to identify hotspots in distributed applications, main memory and CPU cache latencies, among others. In contrast, we focus on coarser-grained metrics such as runtime and bandwidth of microbenchmarks with the goal of taking into account the points of view of both system administrators and users. Similar studies have focused on other cloud platforms such as Microsoft's Azure [39]. Other related profiling efforts have the goal of improving the scheduling of applications on shared infrastructure by identifying and reducing contention between applications [35, 70]. More recently, in [25], the authors present a study of the impact of slow failures (i.e. "hardware that is still running and functional but in a degraded mode, slower than its expected performance") found in large-scale cluster deployments in 12 institutions.

In [48] the authors describe a suite of tests composed of of microbenchmarks that run continuously over the entire Grid5000 infrastructure. The heuristic to decide which tests to run and where is similar to ours, but in our case we prioritize testbed coverage. In [47] a set of open questions for experimental testbeds are outlined, with respect to reproducibility of experiments. In particular, the topic of "Respective Responsibilities of Testbeds and Experimenters" poses the questions of "How far should testbeds go with providing advanced services to experimenters? What should be left as a burden for experimenters?" As part of our work, we have introduced the foundation for a new service that aids experimenters in getting a better understanding of the variability of the underlying platform with respect to the performance of basic subcomponents (CPU, memory bandwidth, network and storage).

Another two broad topics that relate to our work are anomaly detection [9, 64, 63] and straggler analysis [14, 2, 68]. In the former, runtime metrics are analyzed either offline or online in order to identify events that do not conform with the performance expectation of the operator, either at hardware or software levels. Straggler analysis deals with identifying a small proportion of subjobs that cause significant degradations on the parent job. We see our work as complementary to these two topics and envision the methodology and analysis presented here as a way of generating a baseline on which new techniques and approaches in both can be evaluated.

DCBench [32], CloudSuite [19], TailBench [38] and BigDataBench [66] are benchmarking suites whose goal is to recreate workloads that run on cloud infrastructures. In our case, our goal was to target any type of workload running on CloudLab and thus we ended up selecting a generic (and simple) workload for our study.

9 Conclusion and Future Work

In this paper, we have explored the types and magnitudes of hardware performance variation that are an inevitable part of measuring the performance of computer systems. The method we developed for finding unrepresentative resources can be used to provide more consistent environments, and the CONFIRM system can help to design better experiments. These results demonstrate valuable properties of a large, shared experimentation platform: scale is required in order to determine which servers are representative and which are not, and measurement and analysis done once can be used for many experiments.

In this study, we have deliberately focused on the set of hardware resources whose performance is of the most interest in the CloudLab testbed. Differences due to system software and libraries—kernels, compilers, memory allocators, etc. should not be discounted, and there are many more hardware metrics that are of interest. We hope to expand our study to include these factors in the future.

Code and Data

Raw Data and Analysis Code:
`doi:10.5281/zenodo.1435969`
CONFIRM: `https://gitlab.flux.utah.edu/emulab/confirm`
Benchmarks: `https://gitlab.flux.utah.edu/emulab/cloudlab-benchmarks`
Benchmark Orchestration: `https://gitlab.flux.utah.edu/emulab/cloudlab-orchestration`

Specific versions within the `git` repositories used for this paper are identified with the `osdi18` tag.

Acknowledgments

We would like to thank Jeff Phillips for suggesting the Kernel 2-sample MMD test for our evaluation, as well as providing valuable assistance in understanding it. We are grateful to the faculty and staff of the Flux Research Group for their feedback prior to submission, and to the anonymous OSDI reviewers as well as our shepherd, Justine Sherry, for their feedback and suggestions during the review and shepherding process. This work was made possible by the CloudLab testbed, supported by the National Science Foundation under Grant Nos. CNS-1419199 and CNS-1743363. This work was also partially supported by NSF Grant No. OAC-1450488 and the Center for Research in Open Source Software (`https://cross.ucsc.edu`).

References

[1] J. Allspaw. *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly Media, Inc., 2008.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI*, volume 13, pages 185–198, 2013.

[3] J. Axboe. Flexible I/O tester. `https://github.com/axboe/fio`, 2006-2018.

[4] P. Balaprakash, R. B. Gramacy, and S. M. Wild. Active-learning-based surrogate models for empirical performance tuning. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[5] Barnstormer Softworks, Ltd. Welcome to geni-lib's documentation! `http://docs.cloudlab.us/geni-lib/index.html`, 2016.

[6] A. Berlinet and C. Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[7] F. Caglar, S. Shekhar, and A. S. Gokhale. Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud. In *REACTION*, 2014.

[8] G. Casale, S. Kraft, and D. Krishnamurthy. A model of storage I/O performance interference in virtualized systems. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops*, ICDCSW '11, pages 34–39. IEEE Computer Society, 2011.

[9] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[10] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt. Cloudscope: Diagnosing and managing performance interference in multi-tenant clouds. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 164–173, Oct 2015.

[11] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, ATC '05, pages 24–24. USENIX Association, 2005.

[12] C. Curtsinger and E. D. Berger. Stabilizer: statistically sound performance evaluation. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 219–228. ACM, 2013.

[13] A. B. de Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney. Why you should care about quantile regression. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 207–218, New York, NY, USA, 2013. ACM.

[14] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[15] D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74(366a):427–431, 1979.

[16] D. Duplyakin, J. Brown, and D. Calhoun. Evaluating active learning with cost and memory awareness. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 214–223, May 2018.

[17] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature management in data centers: why some (might) like it hot. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):163–174, 2012.

[18] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing*, Oct. 2012.

[19] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. 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, pages 37–48, New York, NY, USA, 2012. ACM.

[20] W. Fu, M. Wang, S. Hao, and X. Wu. Scalable active learning by approximated error reduction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1396–1405. ACM, 2018.

[21] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 57–76, New York, NY, USA, 2007. ACM.

[22] R. B. Gramacy and H. K. Lee. Adaptive design and analysis of supercomputer experiments. *Technometrics*, 51(2):130–145, 2009.

[23] B. Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[24] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[25] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, 2018.

[26] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. The unwritten contract of solid state drives.

In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 127–144. ACM, 2017.

[27] T. Hoefler and R. Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 73. ACM, 2015.

[28] Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, 248966-040 edition, April 2018. Section 2.4.6.

[29] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Proceedings of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 104–113, 2011.

[30] iPerf3 Authors. iPerf - the ultimate speed test tool for TCP, UDP and SCTP. `https://iperf.fr/`.

[31] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, Apr. 1991.

[32] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. Characterizing data analysis workloads in data centers. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–76. IEEE, 2013.

[33] J. Jurečková, J. Kalina, et al. Nonparametric multivariate rank tests and their unbiasedness. *Bernoulli*, 18(1):229–251, 2012.

[34] T. Kalibera, L. Bulej, and P. Tuma. Benchmark precision and random initial state. In *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 484–490, 2005.

[35] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012.

[36] K. Kandasamy, J. Schneider, and B. Póczos. Bayesian active learning for posterior estimation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[37] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks. Profiling a warehouse-scale computer. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 158–169. ACM, 2015.

[38] H. Kasture and D. Sanchez. Tailbench: A benchmark suite and evaluation methodology for latency-critical applications. In *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016.

[39] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server engineering insights for large-scale online services. *IEEE micro*, (4):8–19, 2010.

[40] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[41] J.-Y. Le Boudec. *Performance evaluation of computer and communication systems*. EPFL Press, 2011.

[42] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, pages 50–60, 1947.

[43] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995.

[44] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. A large-scale study of flash memory failures in the field. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 177–190. ACM, 2015.

[45] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 265–276, New York, NY, USA, 2009. ACM.

[46] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 237–250. ACM, 2010.

[47] L. Nussbaum. Testbeds support for reproducible research. In *Proceedings of the Reproducibility Workshop*, pages 24–26. ACM, 2017.

[48] L. Nussbaum. Towards trustworthy testbeds thanks to throughout testing. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, Orlando / Buena Vista, FL, USA, May 29 - June 2, 2017*, pages 1571–1578, 2017.

[49] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao. Who is your neighbor: Net I/O performance interference in virtualized clouds. *IEEE Transactions on Services Computing*, 6(3):314–329, July 2013.

[50] A. W. Reece. Achieving maximum memory bandwidth. `http://codearcana.com/posts/2013/05/18/achieving-maximum-memory-bandwidth.html`, May 18 2013.

[51] A. W. Reece. Memory bandwidth demo. `https://github.com/awreece/memory-bandwidth-demo`, May 19 2013.

[52] B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204. ACM, 2009.

[53] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison, 2009.

[54] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[55] S. Shapiro and M. Wilk. An analysis of variance test for normality (complete samples). *Biometricka*, 52:591–611, Dec. 1965.

[56] R. Shea, F. Wang, H. Wang, and J. Liu. A deep investigation into network performance in virtual machine based cloud environments. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1285–1293, April 2014.

[57] S. Sonnenburg, H. Strathmann, S. Lisitsyn, V. Gal, F. J. I. Garcìa, W. Lin, S. De, C. Zhang, frx, tklein23, E. Andreev, JonasBehr, sploving, P. Mazumdar, C. Widmer, P. D. . Zora, G. D. Toni, S. Mahindre, A. Kislay, K. Hughes, R. Votyakov, khalednasr, S. Sharma, A. Novik, A. Panda, E. Anagnostopoulos, L. Pang, A. Binder, serialhex, and B. Esser. Shogun 6.1.0, Nov. 2017.

[58] J. Taheri, A. Y. Zomaya, and A. Kassler. vmbbprofiler: A black-box profiling approach to quantify sensitivity of virtual machines to shared cloud resources. *Computing*, 99(12):1149–1177, Dec. 2017.

[59] The CloudLab Team. CloudLab hardware. `https://www.cloudlab.us/hardware.php`, 2018.

[60] The CloudLab Team. The cloudlab testbed. `https://cloudlab.us/`, 2018.

[61] The CloudLab Team. Hardware. `http://docs.cloudlab.us/hardware.html`, 2018.

[62] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell. Modeling virtual machine performance: Challenges and approaches. *SIGMETRICS Perform. Eval. Rev.*, 37(3):55–60, Jan. 2010.

[63] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan. Online detection of utility cloud anomalies using metric distributions. In *Network Operations and Management Symposium (NOMS)*, pages 96–103. IEEE, 2010.

[64] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical techniques for online anomaly detection in data centers. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 385–392. IEEE, 2011.

[65] G. Wang, L. Zhang, and W. Xu. What can we learn from four years of data center hardware failures? In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 25–36. IEEE, 2017.

[66] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 488–499. IEEE, 2014.

[67] N. J. Wright, S. Smallen, C. M. Olschanowsky, J. Hayes, and A. Snavely. Measuring and understanding variation in benchmark performance. In *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, pages 438–443. IEEE, 2009.

[68] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC)*. ACM, 2014.

[69] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz. Selecting the best VM across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 452–465. ACM, 2017.

[70] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. CPI 2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 379–391. ACM, 2013.