On the Island of Misfit Toys

Aleksander Maricq Dmitry Duplyakin Ivo Jimenez Carlos Maltzahn
Ryan Stutsman Robert Ricci

University of Utah, University of California Santa Cruz
{amaricq, dmdu, stutsman, ricci}@cs.utah.edu {ivo, carlosm}@cs.ucsc.edu

The performance of compute hardware varies: running the same software repeatedly on the same system, on the same system at a later date, or on supposedly-identical systems produces performance results that differ. This fact has important implications for the reproducibility of systems research and the ability to quantitatively compare the performance of different systems. It also has implications for commercial computing, such as in the quantification and measurement of SLAs.

Over a period of 10 months, we conducted a large-scale study capturing performance samples from 835 servers, comprising nearly 900,000 data points. We examine this data from two perspectives: from that of a service provider wishing to offer a consistent environment, and that of a systems researcher user who must understand how variability necessarily affects their results. From this examination, we draw a number of lessons about the types and magnitudes of performance variability, and explore 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. We have made the full dataset and analysis tools publicly available, and have built a system to interactively explore the data and make recommendations for experiment parameters based on rigorous statistics.

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 [**?**]. In the commercial space, understanding and controlling performance variability is critical to providing good user experience [**?**] and to plan resource provisioning [**?**].

Large systems have many sources of performance variability (hereafer referred to as simply "variability"), but the most fundamental is variability in the performance of hardware. For the purposes of this paper, we consider two types of variability: variability of the *same physical system over time*, and variance between different physical systems that are supposedly *identical*. Hardware can exhibit variance due to temperature [4], remapped storage blocks [9] or memory cells [12], variance in manufacture [15], "fail-slow" hardware [5], and many more causes.

Our goal in this paper is to 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 which sources of variability are under the provider's control and which are not. On the user side, we consider what sources of variability cannot be controlled by the provider, and thus must be dealt with by the user; Understanding these issues is important to all IaaS users who wish to measure the performance of their systems, and we make specific recommendations for researchers who are drawing scientific conclusions based on their measurements of system performance. While the data used for this paper was collected on an IaaS provider that constitutes research infrastructure (a "testbed"), these lessons also apply to clouds, datacenters, etc. that seek to offer their customers guaranteed levels of performance through Service Level Agreements (SLAs).

We have performed a large-scale study capturing performance samples from over 800 servers over a period of ten months. These servers are part of CloudLab [**?**], a platform for systems research that provides exclusive-access "raw" access to compute and storage resources, and which places an emphasis on repeatable research. The CloudLab administrators enabled us to run a set of our own benchmarks on on servers that were "free": not allocated to other users at the time. This enables us to report performance numbers that users could reasonably expect to see in their own applications, and this number is not directly affected by other simultaneous users.

In this paper, we:

- Provide background material about statistical methods used to assess confidence in performance results (§2) and the impact of variability on the experimentation process
- Describe our testing framework, the machines we

tested, and the resulting dataset (§3), all of which we have made publicly available

- Analyze this dataset (§4) to understand the amount of variability of each source. We devise methods for service providers to identify servers with unrepresentative behavior, and draw lessons for users about how to account for the inherent variability that still remains. We present a tool that we have created, CONFIRM, which aids experimenters in gathering statistically significant results.

Throughout, we identify specific findings (◊) that we hope will help service providers provide more consistent facilities, and for users to understand and cope with the variability inherent in computer systems.

2 The Statistics of Performance Variability

The fundamental way in which 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 from them. If we run a given experiment just once, we have no way to determine whether the result from that run is representative of the system's performance or not: it may represent a perfectly "average" run, it may be an outlier, or it may be somewhere in between. As we run more *repetitions* of the same experiment, we get a better sense for how much the results vary, and can use the statistical tool of *confidence intervals* to represent our level of confidence in our results.

Confidence intervals are expressed using a *width* r and a *confidence level* α. r defines a range that we are fairly certain the true mean (the mean we would compute if we could do an infinite number of repetitions) lies in. The confidence level quantifies how "fairly certain" we are that the true mean lies in this range. For example, if we compute a mean of 10.0, with a confidence interval of $9.9 - 10.1$ at 95% confidence, this indicates that we are 95% confident that the true mean lies within 1% of our measured value of 10.0. This is important for our general confidence that our experiment design has been rigorous enough, and specifically applies to our ability to confidently state that one system performs better, worse, or the same as another. If we compare two performance results that have different means, we can only make a strong statement that one is higher or lower if their confidence intervals (which many systems papers do not compute) do not overlap. The smaller the effect an experiment is analyzing (for example, a 5% performance improvement), the more the width of the confidence interval matters.

◊ **Finding 1:** ***Perform enough repetitions to achieve tight confidence intervals***
Techniques from statistics provide robust foundations for making strong statements about performance differences between systems. Because these techniques typically start with an estimate of variance, having data from the underlying platform can help "seed" them.

Using the equations for computing confidence intervals [**?**], for a given desired width and confidence level, we can compute the number of experiments that are likely to be required to reach our target. All we need is an initial estimate of the *variance* of the experiments. The data that we have collected and analyzed for this paper can serve as such an initial estimate, and our CONFIRM tool (§4.4) can compute these estimates. A critical aspect is that the higher the variance of a set of experiments, the wider the range is for a given confidence level for a given set of experimental results. Put another way, when variance is small, only a small number of runs are required to gain high confidence that the true mean is close to the measured mean; when variance is large, a larger—potentially *much* larger—set of experiments must be run to achieve a similar confidence.

To illustrate this point, we use $E(r, \alpha, X)$ to represent the number of experiments that must be run to get an interval within $r\%$ of the computed mean with confidence level α for experiments whose values comprise the series X. The value of E can vary widely depending on the deviation of the systems whose performance we are measuring (σ_X). To provide a consistent standard to compare against, we adopt $E(1\%, 95\%, X)$, denoted as $\check{E}(X)$, in this paper, though depending on the circumstances, different values of r and α might be warranted for other experiments.

To get an intuition for how σ_X affects $\check{E}(X)$, consider two datasets, X_1 and X_2 that have the same mean but different standard deviations. With $\sigma_{X_1} = 0.01 \cdot \bar{X}_1$ (the standard deviation is 1% of the mean), $\check{E}(X_1)$ is only 4. With $\sigma_{X_2} = 0.05 \cdot \bar{X}_2$ (the standard deviation is 5% of the mean), $\check{E}(X_2)$ is 96: it takes $24x$ as many experiments to get the same confidence in experiments X_2 as X_1. This has strong implications for experiment design: if we take X_1 and X_2 to be the same software, run on two different hardware systems with different underlying performance variability, we can get high confidence in our results from far fewer repetitions on the lower-variability hardware. If we had run X_2 only 4 times, our 95% confidence interval would be $X_2 \mp 0.49X_2$: in other words, our confidence interval would be almost 10% wide, far too wide to have confidence in a measured effect of, say, 5%.

◊ **Finding 2:** ***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 the chance that the conclusions are incorrect.

2.1 Nonparametric Confidence Intervals

We find that many of the hardware benchmarks that we study in this paper do not follow the normal distribution (analyzed in more detail in §**??**). Intuitively, a system such as RAM has a maximum bandwidth that we cannot exceed (except via measurement error), and typical results are close to, but not quite at, that maximum. Thus, there is limited room for measurements to vary on the "high" side of the median, but there is a much longer tail on the "low" side, creating a skewed distribution. Rather than having to fit each type of measurement to a—potentially different—distribution, we use nonparametric statistics.

Statistical methods can be broadly classified into two classes: parametric and nonparametric techniques. The former class relies on the assumption that the analyzed data stems from known probability distributions (such as the Normal/Gaussian distribution). A variety of closed-form expressions for statistical quantities of interest enable powerful parametric analysis. In contrast, nonparametric techniques are used when the probability distributions are unknown. In parametric cases, often the data or the measurement noise present in the data is assumed to be normally distributed. In other words, the data or the noise is viewed as samples from the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ characterized with the mean μ and the standard deviation σ. Not only can the empirical estimates of μ and σ be obtained, but also the linear regression or Analysis of Variance (ANOVA) can be used for statistical modeling. However, many studies suggest that the normality assumption does not hold for the data obtained in computer experiments, especially when the data includes measurements of performance. Due to the inherent nondeterminism in computer systems, performance variations observed in practice are severe and unpredictable, both on a single machine [7] and in parallel programs running on supercomputers [16]. In [6] and [3], the authors provide recommendations for statistically sound performance analysis and argue for applying robust nonparametric techniques.

To Write: Rob ►*Consider turning the following into a more step-by-step stlye*◄ For distributions that are non-normal, *nonparametric* confidence intervals are a powerful tool. Specifically, these numbers $\check{E}_n(X) = E_n(1\%, 95\%, X)$, where the subscript denotes the nonparametric case, are calculated as follows. As described in [8], for n samples in X, lower and upper bounds of the confidence interval for the median can be estimated as the $\left\lfloor \frac{n - z\sqrt{n}}{2} \right\rfloor$-th and $\left\lceil 1 + \frac{n + z\sqrt{n}}{2} \right\rceil$-th values in the sorted list of X values, respectively (these brackets denote rounded-down and rounded-up index values, respectively). Calculated as the *standard score* or *z-score* [**?**] using the normal distribution, $z = 1.96$ for $\alpha = 95\%$. After finding these bounds, we compare them against the selected $r\%$ error bounds around the median: $[(1 - \frac{r}{100}) \times \text{median}\{X\}, (1 + \frac{r}{100}) \times \text{median}\{X\}]$. For a growing set of measurements, i.e. sets of samples X with increasing n, we define $\check{E}_n(X)$ as the smallest n for which the confidence interval lies completely within these error bounds.

3 Methodology

Over a period of 10 months, from [**FIXME**: *May 17, 2017*] to April 1, 2018, we collected extensive performance measurements on servers that are part of the three CloudLab [**?**] 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. We adopted a "least-recently-measured-first" strategy in our testing framework, preferring to measure servers that we had not recently measured. As a side effect of the way that the CloudLab allocation policies and usage patterns work, servers were not sampled uniformly: some servers may have been unavailable for up to months at a time, as they were part of long-running experiments, and in general, and in general, the more popular the type of server, the more sparsely sampled it is.

3.1 Testing Framework

Our testing framework is built with "geni-lib" [1], a Python library for interacting with GENI-compatible testbeds such as CloudLab. We wrote an orchestration script to run at a fixed interval for each CloudLab site and select In order to avoid consuming excessive resources on CloudLab, this interval is [**FIXME**: *how many*] hours, and each iteration tests three to five nodes, depending on the size of the cluster. Servers are selected by fetching a list of the target site'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 prevent them from always ending up at the head of the list. Once our test servers are provisioned, we wait for the provisioning process to be completed, SSH into the server, and automatically clone

and run our test script/benchmarks. A test run can take from [**FIXME**: *min*] minutes to [**FIXME**: *max*] hours; the majority of this time is spent running disk tests.

3.2 Benchmarks

We selected a set of benchmarks to cover three key resources: memory, storage, and networking. In our selection of benchmarks, we had to strike a balance between observing the performance of the hardware when pushed to the limit—so that we can detect degraded performance—and what might be seen in a more typical application—so that users can use the results to understand "typical" behavior of the hardware they are using. 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 I/O devices. We also required benchmarks that were portable across different architectures, due to the presence of both x86 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 [**FIXME**: *when?*] months in.

[**FIXME**: *Do we feel comfortable saying that CPU performance "problems" would appear in the memory benchmarks?*]

3.2.1 Memory

We use two different benchmark suites for our memory tests. First, STREAM [**?**] (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. All tests use many GB of memory to minimize caching effects. While we made no modifications to the STREAM tests themselves, we did modify the code to run both the single-threaded and multi-threaded versions of STREAM in sequence, as well as provide more complete reporting of statistics at the end of the run. Second, we use a suite of micro-benchmarks by A.W. Reese [**?**] for **Rob** ►*We need to either say how they are different, or if we will not discuss them, just don't mention*◄ slightly different simple tests and some non-portable x86-specific tests. In the case of Intel processors, we run tests both with a standard frequency-scaling configuration and with a configuration 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` and 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.

3.2.2 Storage

To test storage, we use `fio` [**?**] 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, we issue a TRIM to the device using `blkdiscard` before we run any write workload. This is so that the device doesn't have to worry about the contents of blocks when writing to them [**?**]. We install `fio` from the Ubuntu package repository.

3.2.3 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` [**?**] with TCP and take measurements bidirectionally. Some of servers we test are rack-local with the destination server, and require multiple layer-2 hops. Since CloudLab makes its topology public, we know that, for all nodes we are testing, all non-local nodes are two Ethernet hops away, and we record the path taken with each test. We install `iperf3` from the Ubuntu package repository, and `ping` is already included bundled with the base operating system.

3.3 Software Parameters

While we focused mainly on hardware-based variance rather than software, we recognize that software differences can have a major impact on performance. To this end, we made certain to track the version information of the kernel used, some installed packages, and the revision of our repository containing our test script and memory benchmark sources. All machines tested ran the CloudLab default Ubuntu 16.04 image, and while this image underwent a number of revisions, we believe we have collected enough information to be able to check for changes that may be more immediately impactful.

A number of versions stayed constant: The linux kernel release (4.4.0-75-generic), ping (iputils-s20121221), and iperf3 (3.0.11) utilities. Additionally, while our testing repository was updated a number of times over the testing period, no modifications were made to any timed portions 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 seem to have used slightly earlier versions of both gcc and fio, so to maintain software consistency we excluded them while performing our analysis.

One major example of the impact of software on performance is the recently released mitigations for spectre and meltdown mitigations. While we realize that these mitigations will most likely significantly affect our benchmarks, we have not included any such results in this study. The CloudLab release of default images with these mitigations built in arrived towards the end of our data collection period for this paper. We are, however, continuing to collect data so that we can analyze the impact of these changes in the future.

3.4 Machines Tested

To Write: Aleks ►*Mention we used all "high quantity" nodes available to us at the time.*◄ We gathered our results from CloudLab's three primary clusters: Utah, Wisconsin, and Clemson. Severs at each site ate divided into a small number of distinct homogeneous *types*; no sites currently have overlapping types. All servers we tested are interconnected via a 10Gbps experimental network. 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 dozens of disks; we did not test these types to avoid consuming CloudLab's scarcest resources. For more detailed information regarding the experimented-upon server types such as specific component models, refer to the Hardware section of the CloudLab docs [14] or the CloudLab Hardware page [13].

3.4.1 CloudLab Utah

From CloudLab Utah, we used 315 HPE Proliant **m400** AArch64 APM X-GENE Server-on-a-Chip cartridges, each with eight 64-bit ARMv8 (Atlas/A57) cores and a SATA III/M.2 SSD. We also used 270 HPE ProLiant **m510** server cartridges, each with an 8-core Intel Xeon D-1548 and an NVMe SSD [14]. CloudLab Utah was built off of the low-power HP Moonshot platform, so all server are contained within a total of thirteen 45-node chassis.

3.4.2 CloudLab Wisconsin

From CloudLab Wisconsin, we used 90 `c220g1` servers and 163 `c220g2` servers, all of which are dual-socket Cisco c220m4 1U servers with a SATA III SSD and Two SAS-2 HDDs. The c220g1 and c220g2 servers differ slightly in CPU and RAM: the c220g1s have two Intel E5-2630v3 8-core CPUs, and the c220g2s have two Intel E5-2660v3 10-core CPUs with two more populated DIMM slots [14].

3.4.3 CloudLab Clemson

From CloudLab Clemson, we used 84 Dell PowerEdge `c6320` sleds with two Intel E5-2683 v3 14-core CPUs, and 96 Dell PowerEdge `c8220` sleds with two Intel E5-2660 v2 10-core CPUs. Each node type additionally contains two SATA II HDDs [14].

3.5 Software Variance

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 the version information of the kernel for each test, 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 was 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 a number of times over the testing period, no modifications were made to any timed portions 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 seem to have 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.

3.6 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

Table 1: Coverage of the resulting dataset

CloudLab Site	Hardware Type	Tested/ Total Nodes	Total Runs	Mean/ Median Runs
Clemson	c8220	96/96	1742	18/12
	c6320	82/84	741	9/8
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

hardware type can be found in table 1. 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. Since we are only able to test nodes that are available for allocation, we have inconsistent sampling of nodes, especially during periods of high testbed utilization.

We have made our base dataset, orchestration script, and benchmark script publicly available at `TODO`. Also included are various Jupyter notebooks [**?**] that we used to analyze the data and produce all of the figures in this paper.

4 Understanding and Compensating for Variability

Using our dataset, we now perform a series of analyses that seek to understand the sources and properties of variability, and to develop tools that will aid service providers and users in working with this variability.

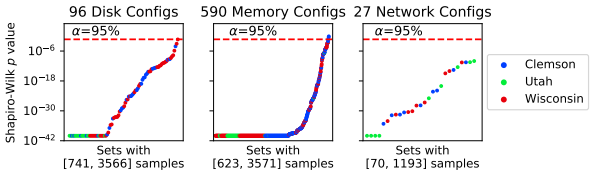
4.1 Major Sources of Variability

[**FIXME**: *Probably either Dmitry or Aleks could write this section*]

Rob ▶*What I am looking for in this section is some basic reporting on what varies a lot, what doesn't, etc. Does disk vary more than RAM? HDD more than SSD? Are the types and amounts of variability roughly consistent across nodes? I want to see lots of dense graphs here.*◀

◊ **Finding 3:** ***Some amount of variation is unavoidable***
Some degree of varation in hardware performance is unavoidable, no matter what steps the facility provider takes to provide consistent hardware. These varations inevitably "climb up" the software stack, meaning that experimenters must take them into account in experiment designs.

Figure 1: Testing normality of the collected data.



[**FIXME**: *This finding may be superseded by "Use low-variance hardware whenever possible"*]

◊ **Finding 4:** ***Use the simplest platform suited to your experiment***
More complex sytems (SSDs, NUMA, network filesystems, etc.) tend to increase performance variation and offer more pitfalls; avoid them unless they are necessary for your experiment.

[**FIXME**: *We cannot conclusively say whether or not the first part of this is the case. The second part may warrant its own finding and accompany a discussion of this "oscillatory" time-independent effect we noticed on the Wisconsin SSDs.*]

◊ **Finding 5:** ***SSDs provide less repeatable results than HDDs***
The level of complexity within SSDs makes them much less preictable than HDDs. They also exhibit greater "memory effects" from previous usage.

4.2 Evaluating Normality

[**FIXME**: *Consider moving nonparametric finding here*]

The statistics commonly used to analyze the performance of computer systems [**?**] tend to assume that measurements are normally distributed. We use the Shapiro-Wilk [**?**] to test for normality in the data that we have collected, and find that the results of these benchmarks are **not** normally distributed. We apply this test to all analyzed disk and memory configurations and show our results in Fig. 1. Each point, shown in the order of increasing p-values obtained from the test for each site, characterizes samples we collected for a specific configuration we explored. For points above the threshold $1 - \frac{\alpha}{100}$, where $\alpha = 95\%$ is the selected confidence level, we cannot reject the *null hypothesis* (stating that the samples come from populations which have normal distributions), whereas for the points below the threshold we reject the null hypothesis at this confidence level.

Our analysis shows that we should reject the null hypothesis for over 99% of explored configurations (710 out of 713) based on the collected data. Considering the large number of samples in the analyzed sets, from 70 in the smallest up to 3,571 in the largest, we reject the normality hypothesis without a smallest doubt and continue with nonparametric analysis for all configurations.

> ◊ **Finding 6:** ***Use nonparametric statistics when analyzing performance results with skewed distributions***
> Many computer systems performance results have skewed distributions (longer tails on one side); non-parametric confidence intervals are fairly simple to compute, and work for these distributions in addition to normally-distributed results.

[**FIXME**: *The following graph is now kind of orphaned*]

4.3 Detecting Unrepresentative Servers

To Write: Dmitry ►*Dmitry owns this section*◄

> ◊ **Finding 7:** ***Provide indistinguishable resources***
> When servers—even those that are supposedly identical—exhibit performance differences that can be detected reliably through statistical tests, repeatable experimentation is more difficult, and SLOs are difficult to meet.

4.4 CONFIRM: How Many Measurements Is Enough?

To Write: Rob ►*Give procedure for determining how many experiments to run.*◄

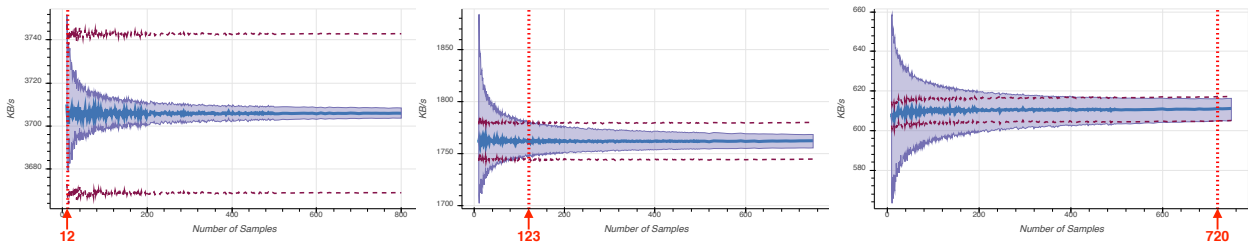
In a truly "online" experimentation setting, an experimenter would use the described procedure iteratively, after every experiment or a small batch of experiments, and decide whether to continue or not. In contrast, we aim to continually collect performance measurements aiming to understand both long-term performance trends and also rare abnormalities. Therefore, we do not stop collecting benchmarking data and propose an alternative, "offline" approach that allows us to evaluate sets of measurements obtained so far or before any particular moment in time.

We refer to this approach as *resampling* and implement it as follows. 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 confidence interval for the median as described above. We shuffle X, select another subset of s values, and obtain new estimates of the confidence interval. After we repeat this process c times, we calculate the means of the lower and upper confidence interval 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 confidence interval observed across many trials. Thus, the results presented in the rest of the paper are obtained using $c = 200$. To estimate the recommended number of measurements $\check{E}_n(X)$, we start at $s = 10$, assuming that smaller subsets are insufficient to estimate nonparametric confidence intervals reliably and should not be considered. Then, we increase s until $s = n$ or the mean confidence intervals fit within the error bounds, as stated earlier. 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 $\check{E}_n(X) = s$ measurements according to the selected allowed error and confidence level.

Our experimentation with the described analysis led to the development of a service we call CONFIRM or CONFIdence-based Repetition Meter. This dashboard, implemented in Bokeh [2], imports our benchmarking datasets and facilities interactive nonparametric analysis of confidence intervals for measurements collected from individual nodes, groups of nodes, and entire hardware types available on CloudLab. CONFIRM is available at: `http://www.confirm.fyi` [1]

Using CONFIRM, we perform two types of analysis: compare different hardware types – specifically, performance of two different HDD models – and investigate how one node's low performance can severely impact experimentation performed on a set of 10 seemingly identical nodes. Information obtained in the former analysis can help users intelligently select the configurations that yield the most repeatable results. In the latter case, the analysis focuses on experimentation that uses a fairly small subset of available nodes. In such sets, due to the individual node variability, the task of obtaining statistically sound results may be significantly more or less difficult, depending on the particular node selection. Both types of analysis can provide insights that are also valuable to system administrators and cloud service providers.

[1] During the period of double-blind review, accesses to this service will not be logged.



(a) 88 HDDs at Wisconsin (`c220g1`) *random reads, high iodepth* (4096)

(b) 82 HDDs at Clemson (`c6320`) *random reads, high iodepth* (4096)

(c) 82 HDDs at Clemson (`c6320`) *random reads, low iodepth* (1)

Figure 2: Visualizations of nonparametric confidence intervals produced by CONFIRM. As the number of samples $10 \leq s \leq n$ grows, 95% confidence intervals (shown as filled areas) for the medians (thick blue lines) shrink and eventually fit within the 1% error bounds around the median (dashed lines). This stopping condition is depicted with red lines and annotated with the numbers of recommended measurements for each configuration.

Table 2: Recommended number of measurements $\check{E}_n(X)$ for 9- and 10-node sets. Estimates are produced using CONFIRM for Wisconsin `c220g2` nodes.

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

4.5 Effects on Full Systems

4.6 Wisconsin Memory Bandwidth

To Write: Aleks ►*Aleks or Ryan - Decide on ordering. This is temporary organization*◄

4.6.1 Background

During benchmarking, unexpected differences arose in the memory bandwidth measurements between the two node types at CloudLab Wisconsin. Both c220g1 and c220g2 nodes use the same server chassis, but c220g2 nodes are a modest but strict upgrade to c220g1 nodes: They have more CPU cores, higher CPU and memory clock speeds, and more populated DIMM slots [14]. Thus, one would reasonably expect benchmark performance to be roughly similar between the two node types, with c220g2 nodes edging out c220g1 nodes.

Contrary to our expectations, experiments showed c220g1 nodes vastly out-performed c220g2 nodes in multi-threaded memory bandwidth benchmarks. Effective memory bandwidth under multi-threaded tests on c220g2 nodes was consistently one-third of what it was on c220g1 nodes, and it was similar to the performance of single-threaded experiments on c220g2 nodes. No other node type across the three CloudLab sites showed this behavior, so we explored this phenomenon further.

Eventually the problem was linked to the extra DIMMs in the c220g2 nodes. Every c220g1 node has exactly one DIMM per memory channel on each socket, but each c220g2 node has an extra DIMM its first memory channel. To confirm our theory, we requested that the extra DIMMs be removed from a c220g2 node allocated to us. Removing the extra DIMMs immediately solved the problem; its multi-threaded memory bandwidth exceeded that of c220g1 nodes as one would expect.

4.6.2 Potential Impact on Real Experiments

Exploring more, we discovered this hardware misconfiguration can have subtle effects on experiments. Each time after using the `mctop` [**?**] tool to profile node inter-CPU topology and latencies, these unbalanced c220g2 nodes would recover some memory bandwidth performance – purely as a side effect of running the software, with no changes to the hardware configuration. This created two modes for unbalanced c220g2 nodes. A slow "before" memory bandwidth, and a faster "after" memory bandwidth. After running `mctop` the bandwidth remained improved until the machine was rebooted.

While `mctop` is primarily a tool for profiling CPU topologies, it is important to point out that it also contains a section at the end that measures memory bandwidth using large NUMA-aware memory allocations. Prematurely halting execution of `mctop` before it can test memory bandwidth does not induce any recovery in multi-threaded memory bandwidth. This suggests that specific application patterns, in this case large memory allocations, can affect future performance of other applications. We believe that these patterns redistribute which physical page frames (and, hence, memory channels)

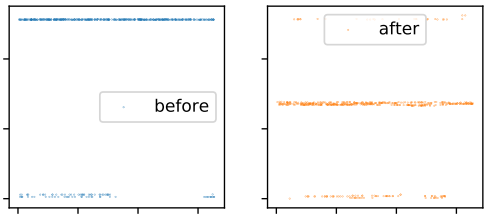


Figure 3: Visualization of physical page frame numbers (pfn) used by stream before and after a run of mctop. The y-axis is the floor of $pfn/8192$, and the x-axis is $pfn \mod 8192$.

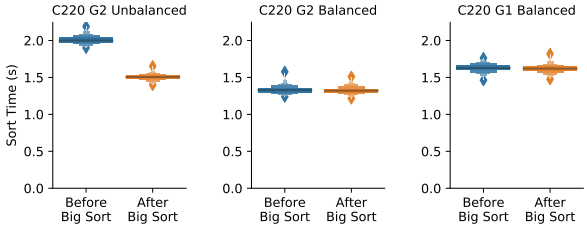


Figure 4: Spark 2 GB sort time before and after a 116 GB sort on two platforms. Sort times improve by 25% on c220g2s after a large sort when memory channels are imbalanced.

would be used for future "big" applications, changing their observed performance until the next reboot.

We looked to profile this behavior by using a tool [**?**] to parse the system-exposed mappings between the virtual address space and physical page frames used by STREAM both before and after `mctop`. Figure 3 plots which physical pages are used by STREAM before and after a single run of `mctop`; the two runs of STREAM clearly rely on different regions of physical memory.

Our original observations were made with the STREAM and `mctop` benchmarks and only affected our artificial metrics collecting, but we immediately wondered if the same effect could impact realistic experiments and change conclusions. To test this, we tried to replicate the issue using simple sorting workloads on Spark [17]. Figure 4 shows that, indeed, experimental results can significantly depend on what workloads and applications were run before the experiment. The right two graphs show the distribution of sort completion time for 2 GB of small records both before and after a larger, 116 GB sort. The c220g1 and c220g2 nodes with DIMMs in a balanced configuration perform the same despite the large intervening sort. The sort-completion time on the unbalanced c220g2s improves by 25% after the big sort.

◊ **Finding 8:** ***Randomize experiment ordering***
Performance effects can "leak" between seemingly unrelated experiments; running experiments in a randomized orders can help expose and control for these effects.

◊ **Finding 9:** ***Cross check results against similar configurations***
Reduce the risk of pathological hardware-software interactions by cross-checking results against similar, but different hardware when possible.

5 Related Work

In [11] the authors describe a suite of tests composed 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 [10] 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).

In [5], 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.

6 Discussion

To Write: Rob ►*Rob owns this section*◄

References

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

[2] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2014.

[3] A. B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney. Why you should care about quantile regression. In *ACM SIGPLAN Notices*, volume 48, pages 207–218. ACM, 2013.

[4] 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.

[5] 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)*, pages 1–14, Oakland, CA, 2018. USENIX Association.

[6] 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.

[7] 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 2005)*, pages 484–490, 2005.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] The CloudLab Team. Cloudlab hardware. `https://www.cloudlab.us/hardware.php`, 2017.

[14] The CloudLab Team. Hardware. `http://docs.cloudlab.us/hardware.html`, November 2017.

[15] 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.

[16] 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.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX.