Does Systems Research Measure Up?

Christopher Small, Narendra Ghosh, Hany Saleeb, Margo Seltzer, Keith Smith
Harvard University
{chris,nkghosh,saleeb,margo,keith}@eecs.harvard.edu

We surveyed more than two hundred systems research papers published in the last six years, and found that, in experiment after experiment, systems researchers measure the same things, but in the majority of cases the reported results are not reproducible, comparable, or statistically rigorous. In this paper we present data describing the state of systems experimentation and suggest guidelines for structuring commonly run experiments, so that results from work by different researchers can be compared more easily. We conclude with recommendations on how to improve the rigor of published computer systems research.

1 Introduction

Systems papers tend to either present new ideas or quantitatively report the performance of systems. While it is reasonable for papers in the former category to contain no performance measurements, it is crucial that the results described in the latter share three critical qualities:

- *reproducibility*: the paper must include enough information to reproduce the experiments for independent verification.
- *comparability*: the experiments run should be structured in such a way that they allow the behavior of the system being measured to be compared with that of other systems and those described in other papers.
- *statistical rigor*: the results must be statistically valid.

In essence, comparability and rigor follow from reproducibility. If another researcher wants to verify the experiment, a suitable description of the benchmark, hardware, software, and system conditions must be given so that the experiment can be recreated. This is the property of reproducibility. To allow the reader to understand the published results in the context of related research, commonly reported metrics should be measured in similar ways on similar systems. This is the property of comparability. In order to validate the published results, an independent researcher not only has to know the result, but also the expected range of values, the measured deviation (or confidence intervals), and the experimental error. A meaningful comparison of experimental results cannot be made without this information, which we call statistical rigor. When experiments and their results are sufficiently explained as to be reproducible, a paper's results are much more convincing since the evidence is far more credible.

We surveyed the proceedings from ten recent computer systems conferences. We set a minimum standard for reproducibility and calculated the percentage of papers that included results that were reproducible by these standards, assuming the availability of appropriate hardware and software.

We then checked the results for comparability, determining which results could be reasonably compared with the results of other research. In cases where the authors used well-known benchmarks, or widely available code, we found that the reported results usually were easily compared. However, in more than 80% of the cases we found that the authors designed their own ad hoc method for benchmarking a commonly measured quantity, be it null system call time, UDP throughput, or context switch time. In these cases, we found that it was virtually impossible to compare results reported in different papers.

The third critical quality of any result reported is statistical rigor; in more than 40% of the results, the authors did not include *any* information about the statistical validity of their results, not even the number of test runs. In 64% of the results reported, the authors did not indicate whether the result was a mean, minimum, maximum, or the only measured value, the distribution of results, or any measure of confidence or error.

We find this distressing. Given our minimal criteria for reproducibility, comparability, and rigor, we concluded that most quantitative systems papers provide inadequate information about the experimental data used to support the paper's claims.

We observed that there are a small number of commonly reported metrics, but researchers repeatedly create individualized tests to measure quantities such as system call overhead, network latency, and file system throughput.

In the following section, we argue for well-designed, common benchmarks. Section 3 discusses the conference proceedings that we surveyed and the data

that we collected; Section 4 presents the results of the survey. In Section 5 we include suggestions for designing and choosing tests. Section 6 discusses statistical rigor, and Section 7 lists guidelines for how to present experimental results. Section 8 includes a discussion of related work on constructing and evaluating benchmarks, and we conclude in Section 9.

2 The Argument for Standard Tests

Designing robust, meaningful tests is challenging. Small differences in the test code or in the test environment can cause substantial changes not only in the test results, but even in what is being measured by a test. We argue against this practice on the following grounds:

- It is difficult to write tests that actually measure what they are intended to measure. There are pitfalls associated with even the simplest benchmarks.
- However well intentioned the author, and however carefully designed, two different tests will perform different measurements. Reusing a standard test greatly increases the comparability of results.
- It is a waste of effort to re-create tests. Test code should be written once and shared. It should change slowly, if at all.
- It is difficult to statistically analyze and present results so that they are meaningful. The techniques commonly used (e.g., computing the standard deviation for normal distributions) are not applicable in all cases.

To demonstrate these challenges, we consider the difficulties involved in creating a simple benchmark program. One of the most frequently cited numbers in operating systems research literature is system call overhead. This quantity is usually calculated by measuring the time required to make a trivial or "null" system call. Even this simple test is fraught with ambiguities.

First, the candidate "null" system call is often the `getpid()` call, which should do almost no work in the kernel (typically a few cycles to copy the process ID from a structure into a return argument). However, on some systems, the value returned by `getpid()` is cached at user level, so only the first invocation actually performs a system call.

In order to use `getpid()` to measure system call overhead, a test program might measure the latency of a single call to `getpid()`. To achieve statistical significance, this program could be executed repeatedly until enough data points have been acquired. Unfortunately, on many systems the timing granularity is too coarse to accurately measure the latency of a single system call. Timing multiple, successive calls to `getpid()`, the usual technique for measuring a short-duration event with a coarse-grained timer, will not work in this case, since the result of `getpid()` is cached after the first call.

The lmbench suite [McVoy96] avoids the `getpid()` pitfall in its null system call benchmark by writing a single byte to `/dev/null`, under the assumption that it is unlikely that a system would check (at user level) to see if a write is directed to `/dev/null`, and if so, not perform the write. However, on some systems (notably Linux), the kernel determines that the target is `/dev/null` substantially earlier in the path through the file system than on others (e.g., FreeBSD). In an attempt to remove one bias from the test (caching of process IDs), this test introduces another (the path length required to identify `/dev/null`). Neither test accurately captures the sought after quantity: the cost of changing protection domains in a particular operating system.

Other problems in designing a meaningful test are not specific to measuring system call overhead. One commonly used benchmarking technique is to measure an individual call, but to issue the call repeatedly in a tight loop to determine its average duration. If the code path through the user and kernel code is sufficiently small, this technique allows the code in the tested path to remain in the cache, hence measuring only the time required to run the code when it is already in cache. In the common case it is unlikely that a short code sequence will always be found in the cache, so results gathered in this way are unrealistic. In some cases, but not all, it is acceptable to report "hot cache" results; a test should take this into account.

In addition to enhancing the scientific rigor of measuring computer systems, benchmark standardization would save many researchers a great deal of time. One recent trend is for researchers to make their test code and data available on the internet. While this is a step in the right direction, it is not a substitute for standardized benchmarks. Test code written for a single platform may not run on other platforms, or may, like the lmbench null system call test, not really measure what it is intended to measure.

Through our survey of past quantitative system papers we have identified existing benchmarks that are implemented well, qualities that experimental results often lack, and general properties that benchmarks need. In this paper we are advocating a general method for systems experimentation and presentation, not proposing a new set of benchmarks or insisting that particular ones be used. The first component of this method is the use of existing, standard benchmarks. However, this is not always possible since appropriate benchmarks may not exist for new system ideas. Therefore the second component of our method is the creation of new bench-

marks according to certain principles we describe. After presenting our survey and conclusions drawn from it, we explain our method for designing new benchmarks, interpreting experimental results, and presenting the results in a meaningful way.

3 Experimental Setup

We surveyed ten systems conference proceedings: the 13th, 14th, and 15th Symposia on Operating Systems Principles [SOSP91, SOSP93, SOSP95], the First and Second Symposia on Operating Systems Design and Implementation [OSDI94, OSDI96], the Fifth and Sixth Conferences on Architectural Support for Programming Languages and Operating Systems [ASPLOS92, ASPLOS94], and the 1994, 1995, and 1996 USENIX Technical Conferences [USENIX94, USENIX95, USENIX96]. We reviewed 235 papers in all. There were between 280 and 311 measurements reported in each conference (see Table 2).

22 of the papers (9%) did not include any measurements. The remaining 213 include the results of a total of 1163 experiments. For each experiment[1], we recorded:

- The area of the experiment.
- The test that was run or quantity that was being measured.
- The characteristics of the experimental setup.
- The attributes of the presentation of the results.

The categories that we used may not be appropriate for all systems papers. These categories were chosen after we performed an initial review of the ten conference proceedings, to describe our sample set of papers. Although not all work in systems is presented at these conferences, these papers, and the areas they represent, are a good indication of what work is being done in systems and what the systems community thinks of as its best work.

The data collected from this survey were used to derive the statistical results found in Section 4 and to determine the Common Tests listed there.

3.1 Areas and Tests

First, we organized the measurements into the test categories described in Table 1. Within each category, there were two general types of tests: standard benchmarks (or well-known programs), and ad hoc benchmarks (not in common use). When ad hoc benchmarks were used, the comparability and reproducibility of the results suffered. For example, results from two papers reporting null system call times cannot be compared if they used different techniques for measuring the null system call, such as those described in Section 2.

At this stage we did not attempt to determine the rigor or value of any particular test or benchmark, but instead generated a list of commonly used tests. Tests that were not represented in the list of commonly used tests were identified as ad hoc. The tests and ad hoc categories are listed in Table 6, in the appendix.

Area	Description	Example
general	general-purpose standard benchmarks that measure quantities from more than one area.	lmbench
cpu	cpu time, instruction count, multiprogramming workload, idle time.	dhrystone
db	database benchmarks.	TPC/B
fs	file system or disk tests.	Andrew
mem	memory (RAM) tests.	bcopy
net	network throughput and latency.	ttcp
para	parallel system and DSM tests.	splash
sys	system throughput and load.	aim

Table 1. Test Areas. After a preliminary review of all the papers, the experiments were found to be in one of these areas.

3.2 Test Details

For each test, we gathered information on how the experiment was run and how the results were presented. The test details fall into three categories: comparability, reproducibility, and statistical rigor.

3.2.1 Comparability

In conducting our survey, we found that results were most easily compared when researchers used standard benchmarks. In general, each area included a small number of well-known tests, which were rarely used, and a larger number of ad hoc benchmarks, which were frequently used. This was distressing because it was often difficult or impossible to compare results in papers that reported the same quantity (e.g., null system call time), but measured it in different ways.

In Section 4.1, we discuss the results in more detail, and in Section 5.3, we discuss how to construct a good benchmark, when no standard benchmark exists for the quantity that a researcher needs to measure.

3.2.2 Reproducibility

In evaluating for reproducibility, our goal was to determine whether a researcher in the field could, given the information found in the paper, reproduce the results presented there.

1. We group multiple runs of a test as a single experiment. For example, if a test is run twenty times, varying a parameter, we report this as a single experiment.

Trace-based simulation is often the simplest way to guarantee that an experiment is reproducible, but it is not the only way. If a paper includes enough detail about how an experiment is set up and run, the experiments can be reproduced by a knowledgeable reader.

In the case of trace-based simulation, making both the traces and the simulator itself publicly available is the simplest way to ensure reproducibility. In the absence of an available simulator (as might be the case for proprietary simulation environments), the researchers must be sure to describe the simulation algorithms in sufficient detail that another researcher could produce a comparable simulator. In the absence of available traces, the researchers must characterize the traces used in sufficient detail that another researcher could determine if the traces were suitably representative. When traces cannot be made available, stochastic simulation with a parameterized workload provides a convenient alternative.

Simulation is only one way of making an experiment reproducible. Any experiment can be made reproducible if the experimental testbed is described in sufficient detail. In systems research, this usually requires describing the hardware and software. A hardware description should usually include the type of processor, its clock speed, and the system in which the processor resides (e.g., the motherboard for a Pentium processor.) It is often important to include information about the memory system (main memory size and cache sizes) and details about the I/O subsystem as well. The software description should identify a specific version of the operating system and application (where relevant).

3.2.3 Statistical Rigor

Many authors provide only a single number as a measure of the performance of a system or a component of a system, without a detailed description of how the number was computed or measured. Without this information, readers cannot accurately interpret results. It is not our intention to cast aspersions on the validity of results presented without this information, just to point out that data collected or computed in different fashions should often be interpreted differently.

In systems research, we measure the performance of systems in the real world, and the real world is neither perfectly consistent nor perfectly predictable. A lack of statistical rigor does not necessarily lead to unbelievable numbers, but without information on the number of measurements taken, the distribution of the measured results, error bars, or variance, we, as readers, can not know how to interpret the results.

Not everything listed here is necessary for rigor, but, in general, the more detail the better. It is important to note that inclusion of detail does not require burying noteworthy results. It is not difficult to include both the important details of how a measurement was taken and a summary of the importance of the major result.

In addition, statistical rigor can highlight additional ramifications of the data collected from a system. If the authors find that the distribution of a set of measurements is exponential, where a normal (bell shaped) distribution might be expected, this is in itself an interesting result. In fact, this will frequently reveal either an error in the test or a particularly noteworthy effect.

For example, in recent work, we measured a reasonable mean (x), but a large standard deviation (nearly 100% of the mean) in a test that we were running. Although we had a (somewhat) plausible explanation for the large deviation, we went back and examined the data more closely. We discovered that the measurements formed a classic bimodal distribution, caused by a problem in the test where two *different* quantities were being measured, one with mean $x/2$, the other with mean $3x/2$. If we had not examined the data in more detail, we would not have noted the problem with our test code. Furthermore, had we published the mean, x, without noting the high standard deviation, the reader would have been none the wiser.

The characteristics that determine statistical rigor vary with the experimental setup. When the quantity being measured is *fixed* (e.g., the number of instructions an application executes on a particular input or the number of occurrences of an operation in a trace) there is little need for statistical analysis. However, when the experimental setup produces variation (e.g., you are measuring an actual system) or there is an aggregation of data, the need for statistical rigor increases.

In the presence of aggregated data, it is imperative to report the number of measurements that are being aggregated and the quantity that is being reported (e.g., mean, median, mode). Next, it is useful to present an indication of the variability of the data. Standard deviation, minimum, maximum, and distribution information are all useful tools for indicating how accurately the aggregate represents the entire data sample.

Timer resolution is closely tied to statistical rigor; coarse grain timing leads to greater margins of error. In conducting our survey, we looked for the use of high-resolution counters or timers to reduce the probable margin of error in experiments.

4 Results

In this section, we present the data and analysis from our survey. Wherever possible we attempted to err on the side of generosity, hence the results summarized here

are a best-case estimate of the comparability, reproducibility, and statistical rigor of the measurements reported.

4.1 Standard and ad hoc tests

We were astonished to discover that more than 80% of the tests run were ad hoc, i.e., did not use standard benchmark tests.

	total	asplos	osdi	sosp	usenix
number of tests	1163	280	289	311	283
standard	19%	27%	16%	13%	23%
ad hoc	81%	73%	84%	87%	77%
Test Breakdown by Area					
general	9.6%	23.6%	2.1%	1.9%	12%
cpu	4.2%	12.1%	2.4%	1.0%	1.8%
ad hoc	*3.5%*	*11.1%*	*1.0%*	*0.6%*	*1.8%*
db	3.0%	0%	6.2%	4.8%	0.7%
ad hoc	*1.5%*	*0%*	*4.5%*	*1.6%*	*0%*
filesys	25.1%	10.0%	19.7%	30.5%	39.6%
ad hoc	*22.6%*	*8.6%*	*18.3%*	*28.3%*	*34.6%*
memory	5.3%	14.6%	2.4%	4.5%	0%
ad hoc	*5.3%*	*14.6%*	*2.4%*	*4.5%*	*0%*
network	13.5%	6.4%	10.7%	12.5%	24.4%
ad hoc	*12.5%*	*6.4%*	*10.7%*	*12.2%*	*20.8%*
parallel	13.2%	9.3%	25.3%	17.4%	0.4%
ad hoc	*10.1%*	*8.2%*	*18.3%*	*13.5%*	*0%*
system	26.0%	23.9%	31.1%	27.3%	21.2%
ad hoc	*24.7%*	23.9%	28.6%	23.4%	*21.1%*

Table 2. Summary of Test Statistics, Areas. Tests from each area, broken out by conference and category. Below the percentage of tests from each area we report the percentage of tests from that area that were ad hoc. The sum of the percentage of tests in each area is 100%. The sum of the *ad hoc* rows of a column is the percentage in the **ad hoc** row of that column. Note that there are no ad hoc general tests; all general tests were standard benchmarks.

The type of test varied by conference (see Table 2), but the results were not substantially different. We found it surprising that papers published at USENIX, which is often considered to be a less prestigious conference, more often used standard benchmarks than papers published in SOSP or OSDI.

Across the four conferences (ASPLOS, OSDI, SOSP, and USENIX) 23% of the tests were ad hoc file system tests and 25% were ad hoc system tests. This distribution varied somewhat by conference. For example, as might be expected, ASPLOS had a higher percentage of ad hoc CPU and memory tests (26%).

Only two of the standard tests listed in Table 6 (in the appendix) were used for more than 2% of the reported measurements (lmbench at 2.2%, SPEC at 7.1%). USENIX accounted for the majority of the use of lmbench while ASPLOS accounted for the majority of the use of SPEC.

It was surprising how infrequently standard benchmarks were used. In the SOSP proceedings, it was more common to find ad hoc measurements of null system call time (eight times) than it was to see use of the SPEC benchmarks (five times). This is not just a reflection on the use of SPEC; it is indicative of the frequency of use of other standard benchmarks as well.

4.2 Reproducibility and Comparability

Many of the reproducibility and comparability attributes (see Section 3.2) were rarely seen. The results are outlined in Table 3.

Although 75% of the tests included information about the hardware platform used, only 39% included enough detail about the software platform to reproduce the test environment. The papers published in OSDI most often included information on the hardware platform (93%), and the papers published in USENIX most often had software information (53%).

	total	asplos	osdi	sosp	usenix
hardware	75%	76%	93%	65%	67%
software	39%	21%	42%	40%	53%
simulation	15%	22%	16%	9%	11%
trace	11%	15%	12%	7%	12%
trace reuse	5%	2%	6%	7%	4%

Table 3. Summary of Test Statistics, Reproducibility and Comparability. Details of the test results surveyed, reported as a percentage of the number of tests.

As mentioned in Section 3.2.2, simulation and trace-based tests are the most easily reproduced. We found that simulation was used most often in ASPLOS papers (22%) and least often in SOSP (9%). Traces were used in 12% or more of the measurements reported in USENIX, OSDI, and ASPLOS, but only 7% of the measurements in the SOSP papers.

Reuse of well-known traces varied from 2% (ASPLOS) to 7% (SOSP). Where 11% of the tests (138) used traces, only 5% (62) used (or reused) commonly avail-

able traces. Traces are extremely useful things; Baker et al.'s Sprite filesystem traces [Baker91] have outlived both the operating system and hardware on which they were gathered. The traces themselves have been used in several subsequent studies. Although an argument can be made that the applicability of these traces has faded with time, reuse of these traces allows studies of different file systems, different cache designs, and so on, to be directly compared.

4.3 Rigor

We were surprised by how little statistical rigor is evident in the literature (see Table 4). 24% of the measurements reported the output of fixed runs and needed little other supporting data. However, of the remaining 76% of the measurements, only 33% of the results reported the mean of multiple test runs; only five of the reports included the median, and only one included the mode. In the rest of the experiments (67% of them), a single number is presented as the result of the measurement without any additional information specifying the number of times that the test was run or how the reported result was derived from these tests.

Few of the non-fixed value results (15%) include the standard deviation, or some other measure of the variance of the measurements. Only three of the 1163 measurements included information on the type of distribution observed.

High-resolution counters are now available on several commonly used hardware platforms (e.g., Alpha, Pentium, and SuperSparc). The systems community should use them where possible, and encourage hardware developers to make them available on the platforms that have not yet adopted them.

4.4 Common Tests

In Table 5 we list the quantities most commonly measured in ad hoc tests[2]. Several of the tests we list here are found in existing benchmark sets (e.g., lmbench [McVoy96] and Ousterhout's microbenchmark suite [Ousterhout90]). Our goal here is to encourage the community to standardize on a common set of tests for these measurements, not to endorse one test suite or another.

5 How To Build Good Tests

The goal of this section (and this paper) is to provide advice about designing, selecting, analyzing, and using good benchmarks. First we discuss the types of benchmarks available and how to determine which are appropriate for the task at hand. Next we describe how to design a benchmark if the standard benchmarks are unsuitable. Once the benchmark is selected, we discuss how to properly run a benchmark and verify that the results are meaningful and accurate. Finally, we discuss how to present the experiments so that others will find the results useful and informative.

	total	asplos	osdi	sosp	usenix
fixed results	24.0%	49.3%	19.7%	13.8%	14.5%
no stats reported	42.5%	23.6%	50.5%	56.6%	37.5%
stats reported	33.5%	27.1%	29.8%	29.6%	48.1%
Stats Reported					
num runs	32.8%	57.7%	21.6%	20.1%	43.0%
mean	35.3%	30.3%	35.8%	35.4%	37.6%
std dev	14.7%	8.5%	17.2%	15.3%	15.3%

Table 4. Summary of Test Statistics, Statistical Rigor. Tests with fixed results include trace-based simulation and counts of fixed quantities (e.g., memory references), reported as a percentage of the number of tests. Of the remaining tests, fewer than one-third of the results included information on the number of runs, and less than 15% included the standard deviation. Of papers without fixed results, we break out the stats reported by each test in the bottom half of the table. For each statistic, the value reflects the percentage of the tests without fixed results that report the statistic. Due to round-off, the percentages may not sum to 100% in all cases.

5.1 Types of Benchmarks

There are two major classes of benchmarks—*micro-benchmarks* and *macro-benchmarks*. *Micro-benchmarks* are used to measure the performance of specific features of a system, such as system call overhead, RPC latency or file system throughput.

Macro-benchmarks, in contrast, are used to measure the overall performance of a system under different workloads. Macro-benchmarks can be divided into two broad classes. Some macrobenchmarks reproduce real workloads, such as a large kernel build, or the replay of a file system trace. Other macro-benchmarks such as dhrystone [Weicker84] and LADDIS [Wittle93] use synthetically generated workloads. Synthetically generated workloads strike a balance between reproducibility and relevance. Real traces or measurements of actual system activity provide good relevance, but are often difficult to precisely reproduce. Additionally, they model only a single point in the spectrum of workloads. In contrast, synthetic workloads may not be representa-

2. Note to program committee: We do not currently have an accurate measure of which ad hoc tests were most frequently run. We plan to make another pass over the surveyed proceedings to more formally generate this list of common tests for the final paper.

tive of any authentic workload, but are easily reproduced. With proper tuning and parameterization, it can also be argued that synthetic workloads can be made to model a wide range of authentic workloads.

There is a natural interplay between the use of micro- and macro-benchmarks. Micro-benchmarks are illustrative, and can be used to highlight specific differences between systems. Macro-benchmarks can then be used to evaluate the effect these differences have on overall performance. If macro-benchmarks are run first, micro-benchmarks can be used to explain the differences in the macro-benchmark results. Micro-benchmarks are also useful as diagnostic tools. A number of the tests that comprise the lmbench suite were derived from performance problems observed by customers [McVoy96].

5.2 What to Measure

The first step in benchmarking a system is determining what should be measured. A good way to decide what to measure (and how to measure it) is to consider the known differences between the systems under test. If the differences are relatively minor (such a modified file system, or an improved protocol stack), start with micro-benchmarks designed to highlight the expected differences between the systems. Then run macro-benchmarks to show the effect of these changes on real-workloads.

If systems are being compared that have substantial differences, such as entirely different operating systems, file systems, or hardware platforms, one may want to start by running macro-benchmarks. These results will suggest the key differences between the systems. These differences can then be quantified using the appropriate micro-benchmarks.

Once the author has determined what kind of benchmark to use, the next question to consider is whether to use an existing benchmark or to design a new one. Our hope is that over time the systems community will come to repeatedly use a small set of standard benchmarks. As we have seen from our survey, there are a small number of commonly measured quantities (see Section 4.4). Therefore, the need to design a new benchmark should not arise often. However, new ideas will sometimes require measuring original quantities.

5.3 Designing a Benchmark

In designing a micro-benchmark, it is vital to make sure that its results reflect the time spent in the part of the system being measured. If the goal is to measure null-RPC time, but a benchmark spends most of its time paging, then the results are not indicative of null-RPC time. Similarly, if the benchmark only spends 5% of its time actually executing the RPC, there may be a problem with the benchmark.

For macro-benchmarks, there are different concerns. Macro-benchmarks should reproduce or represent real-world workloads. This is easy to do if the macro-benchmark runs actual applications, or is driven by traces of real workloads. Unfortunately, there are few standard macro-benchmarks based on real workloads. Many researchers use ad hoc macro-benchmarks based on applications that are available or are of interest to them. Thus many kernel builds and video players are used in different publications. It would be a great service to the research community if these tests were clearly documented, and their data sets made publicly available so that these benchmarks can be reused by other researchers.

Many macro-benchmarks use artificially generated workloads. The goal of using a synthetic workload is usually to create a benchmark that is meaningful to a wider range of users than a specific application's workload. An artificial workload introduces the danger, however, that it is not representative of any user's workload, or worse, that the benchmark results will not map to the results observed by users in their applications.

The key issue in designing a synthetic macro-benchmark is the relevance of the workload that is used. Benchmarks that use a completely random mix of operations are highly suspect. A more reasonable approach is to base the distribution of various operations in the synthetic workload on observed distributions in a real workload. (The LADDIS benchmark [Wittle93] is an example of a synthetic workload based on actual data.) Better yet is a benchmark that captures the dependencies between successive operations that are observed in the real world.

Another concern when writing both micro- and macro- benchmarks is eliminating system dependencies. Consider file system benchmarks. In order to ensure that measurements reflect the performance of the file system, rather than the buffer cache, it may be desirable to explicitly flush the buffer cache to ensure that the benchmark is utilizing the file system. One way of doing this might be to read ten megabytes of "junk files" before each benchmark run. This technique introduces two assumptions about the underlying buffer cache—that it is no larger than ten megabytes, and that it uses an LRU replacement policy. These dependencies are undesirable because the benchmark will give invalid results on systems that violate these assumptions. Even worse, it may be difficult for other users of the benchmark to determine whether these assumptions hold on their systems. A better way to flush the file cache is to unmount and then remount the file system. Another sure fire way

to flush the buffer cache is to run the benchmark on a newly booted system.

A final point about designing a new benchmark is the question of how it actually performs measurements. Many benchmarks fail to factor out the cost of their measurement techniques (e.g., the overhead of calling `gettimeofday`()). This is especially important when measuring short duration events, because the measurement overhead can have a large effect on the measured times.

5.4 Running a Benchmark

There are two important issues to keep in mind when setting up the environment in which to run a benchmark—control and relevance.

When we run benchmarks, we are usually attempting to evaluate the effect of some change in system design on the performance of the system in question. By controlling the environment in which we run a benchmark, we ensure that any performance differences reported by the benchmark can be attributed to the particular design change that we are studying. Thus, all aspects of the test environment—the hardware platform, the operating system configuration, network connectivity, versions of software tools, etc.—should be the same during all of the benchmark tests. The only things that should change between benchmark runs are the parts of the test system that are being compared.

The other issue to consider when setting up a benchmarking platform is that, ideally, the benchmark environment should mimic the environment in which the test systems would actually be used. In benchmarking process switch times, for example, the easiest course is to run the system with a small number of processes belonging to the test suite. In real-world usage, however, there are often a mixture of processes in the system, some ready, others blocked. A better way to run a context switch benchmark would be in the context of a realistic number of ready and blocked processes. Of course, in order to guarantee a controlled and reproducible test environment, the processes must be created in a deterministic and reproducible manner.

5.5 Understanding Results

After running a benchmark, it is vitally important to validate the results that it gives. Initially the question will be whether the change in the system improved some facet of system performance. However, the questions cannot end there. Understanding benchmark results is a microcosm of the scientific method. In looking at the results from a benchmark, it is imperative that all of the results can be explained (e.g., "Why does this test get slower as the size of the buffer increases?"). This may require forming a new hypothesis that explains the observed behavior (e.g., "With a larger buffer size, there is worse cache locality"). Finally, test this new hypothesis to see if it does indeed explain the results (e.g., use CPU-based performance counters to determine the number of cache misses during the benchmark.)

Although it is easy to focus exclusively on the results that do not match expectations about the systems under measurement, it is important to scrutinize all measurements with equal vigor. Implicit in each "expected" result is a hypothesis about what is causing it. These hypotheses also need to be validated.

Examining results in this manner also provides an opportunity to shake out potential flaws in the benchmark. Flaws may include programming bugs as well as unexpected behaviors of the benchmark. As mentioned above, using `getpid()` may be an easy way to measure the system call overhead of an operating system. Since some systems cache the results from the first time a program calls `getpid()`, this technique may not work. Comparing the results to other simple system calls or performing back-of-the-envelope [Bentley84] calculation of the number of cycles it took to execute a `getpid()` call are two ways to check that the results are reasonable.

Another advantage to using standardized benchmarks is that they are (ideally) less likely to include these types of flaws. A standard benchmark allows the researcher to concentrate on changing the system, rather than worrying about the validity of the benchmark results.

5.6 Describing a Benchmark

The goal in describing a benchmark is to provide enough information so that another researcher can reproduce the experiment. Further, the description should convince the reader that the benchmark was appropriate for the measurement, well-designed, and properly run. In other words, it should address the very questions we have discussed in the last five sub-sections.

It is not necessary to provide a detailed walk-through of the benchmarking code. Instead, it is essential to provide a description of the purpose of the benchmark (i.e., what is it designed to measure), how the benchmark measures the quantity, any subtleties that arose in designing the benchmark, and how the benchmark was run. For example, it is important to indicate the cache state (hot or cold), the congestion of the test network, the utilization of the file system, and any other factors that might affect the outcome. Making the code available on-line also provides another level of reproducibility. Finally, the standardization of benchmarks eases this task since the benchmark itself will not be in question, although it is still necessary to accurately describe the conditions under which the test is run.

6 Statistical Rigor

Understanding common statistical methods is invaluable in being able to represent results coherently and accurately. In this section we provide an overview of the statistical tools that a systems researcher should have available when analyzing data and motivate their use.

6.1 Mean, Median, Mode and Distribution

As we stated in Section 4.3, 35% of all the tests we analyzed use the mean as the result of a benchmark. In only 15% of the measurements without fixed results did the authors provided an indication of the variance of the data values, typically through the inclusion of the standard deviation. In over 40% of the cases without fixed results, *no* information was provided by the authors about the statistical rigor of the results.

If a researcher does not use sound statistical techniques when gathering data and presenting results, drawing inferences about the behavior of the measured system is a guessing game. Sound statistical techniques must be used to ensure that the results presented in a paper truly represent the behavior of the underlying system.

It is important to note that if the behavior that is being measured does not vary, statistical analysis is not necessary. For example, if we are measuring the number of instructions generated by a compiler for a particular program, or the number of disk blocks needed to store a fixed data set, we only need to measure the quantity once.

When measuring a system that does not have fixed behavior, the best technique for developing an accurate model of the system is to perform multiple measurements, analyze them, and distill them to one (or a small number of) representative values.

When doing this, the *most* important datum to include is the number of measurements taken. Without this quantity, the validity and significance of the other results reported is difficult or impossible to discern. The larger the number of measurements taken, the greater the confidence that the results shown are meaningful.

Second, when dealing with a large data set, it is often useful to choose a single representative. For a given data set, there are several different *measures of centrality*, or values that represent an "average" value for the data set.

The most commonly used measure of centrality is the *(arithmetic) mean* (the sum of the data points divided by the number of points). When data are distributed according to a normal (bell shaped) distribution, the mean is a reasonable representative for the data set.

The *median* is the central point in the data, where half the measured values are above the median and half are below the median. The median tells the reader where the midpoint of the data is, and is a coarse way to estimate the distribution of the data.

The *mode* is the measurement that appears most often in the data set. The mode gives a feeling for what value is most likely to occur if the experiment is run again.

When data follows a normal (bell shaped) distribution, the mean, median, and mode all have the same value (see Figure 1a), so all three are equivalently valuable as measures of centrality. Unfortunately, many experimental data sets do not follow a normal distribution, and the mean, median, and mode of other distributions do not line up so neatly. Just reporting the mean in this case can be misleading.

For example, on a computer with a multi-level cache, timing of individual memory accesses will yield data with multiple clusters, each cluster representing the time required to service a request from one level of the cache. In this case, the mean describes the average memory access time, but in all likelihood does not represent the latency of any actual memory access. Because of the increased availability of high-resolution timers on modern processors, it has become easier for researchers to time individual short-term events, thus uncovering the underlying distribution of event latencies.

Normal distributions are more often seen in nature. For example, monthly rainfall over a period of years, heights of individuals in a population, and results of IQ tests for a population are likely to exhibit normal distributions. Additionally, the sums or differences of random numbers are likely to exhibit a normal distribution. However, computer measurements are not naturally occurring phenomena, nor are they completely random.

Additionally, most experimental data sets exhibit a phenomenon known as a *left wall*. That is, there is some minimum value, below which it is impossible for the values in the data set to reside [Gould96]. The presence of such a wall can introduce a distribution that is skewed in one direction or another (not symmetrical about the mean). For example, when measuring network transfer time, the speed of light imposes a lower bound (left wall). Transfer time could, theoretically, take infinitely long (and often seems that way), so there is no right wall to the distribution. This is likely to introduce a skewed distribution.

Skewed distributions are not well-represented by the mean data value. In the words of Stephen J. Gould, "means can be grossly misleading ... when variation can expand markedly in one direction and little or not at all in the other" [Gould96].

In these cases, it is often useful to include the median and mode along with the mean. The median and mode, in combination with the mean, give the reader a

sense of how the data is distributed and what the expected behavior of the system will be.

In general, if the mean and median are rather close, but the mode is vastly different (or there are two candidates for the mode), a *bimodal* or *multi-modal* distribution is suggested (see Figure 1b). As described above in Section 3.2.3, the standard deviation of a bimodal distribution can be quite large, which can serve as a check on the assumption that a distribution is normal.

It is important to note that these guidelines are not fool-proof; comparing the mean, median, and mode can only *suggest* the type of distribution from which data was collected. Unfortunately, there is no rule of thumb that always works, and when in doubt, the best course of action is to plot the data, look at it, and try to determine what is happening.

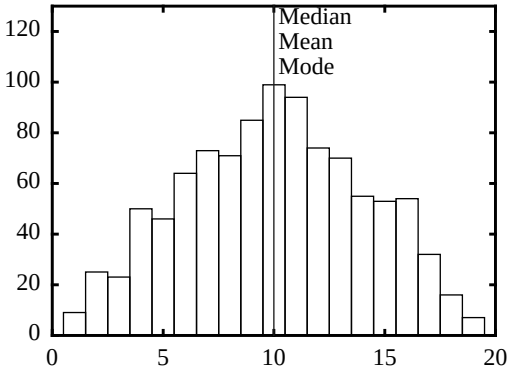
It is critical to select the appropriate metric of centrality in order to properly present data. “No mathematical rule can tell us which measure of central tendency will be most appropriate for any particular problem. Proper decisions rest upon knowledge of all factors in a given case, and upon basic honesty” [Gould96].

6.2 Expressing Variation

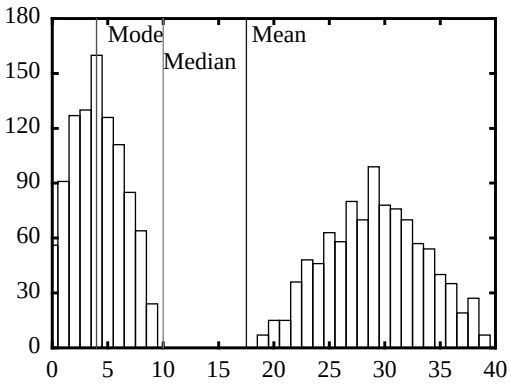
Measures of centrality are not sufficient to completely describe a data set. It is often helpful to include a measure of the variance of the data. A small variance implies that the mean is a good representative of the data, whereas a large variance implies that it is a poor one. In the papers we surveyed, we found that fewer than 15% of experiments included some measure of variance.

The most commonly used measure of variance is the *standard deviation*, which is a measure of how widely spread the data points are. As a rule of thumb, in a normal distribution, about 2/3 of the data falls within one standard deviation of the mean (in either direction, on the horizontal axis). 95% of the data falls within two standard deviations of the mean, and three standard deviations account for more than 99% of the data.

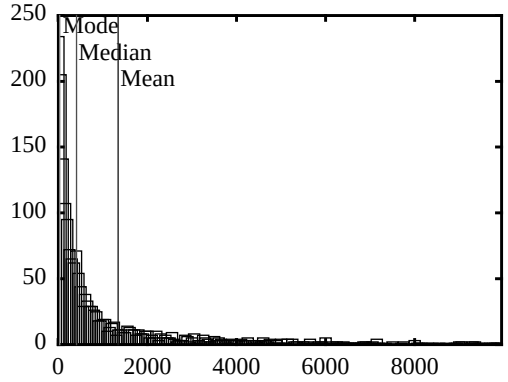
For example, in Figure 1a, which follows a normal distribution, the mean, median, and mode are equal, and the standard deviation is approximately 40% of the mean. However, in Figure 1b, which shows a bimodal distribution, the mean, median, and mode are quite different, and the standard deviation is 75% of the mean[3]. Figure 1c shows an exponential distribution where the median and mode are close, but rather different than the mean. (We discuss techniques for determining the distribution of a data set in Section 6.3.)



A: Normal Distribution



B: Bimodal Distribution



C: Exponential Distribution

Figure 1. Sample distributions. The relationship between the mean, median, and mode give hints about the distribution of the data collected. In a normal distribution, the mean is representative of the data set, while in an exponential distribution, the mode and median are more representative. In a bimodal distribution, no single metric accurately describes the data.

Another metric for analyzing the usefulness of the mean in an experiment is the *margin of error*. The margin of error expresses a range of values about the mean in which there is a high level of confidence that the true value falls. For example, if one were concluding that the latency of a disk seek is within four percent of the mean,

3. The large standard deviation here is because the distribution is bimodal, but bimodal distributions do not necessarily have to have a large standard deviation. The peaks of a bimodal distribution can be close together; in this example they are not.

the margin of error would be four percent. Assuming that this margin of error had been computed for a 0.05 level of significance, then if the experiment were repeated 100 times, 95 of those times the observed latency would be within four percent of the value computed in the corresponding experiment.

Figure 2 is an example of the importance of showing the margin of error. In our example, Figure 2a is put forward to support a claim that a new technique has reduced latency by 10%. However, this graph does not include any indication of the margin of error, or confidence intervals on the data. If the margin of error is small, as in Figure 2b, it is reasonable to believe that latency has been reduced. Figure 2c, however, shows a margin of error that is as large as the stated improvement. The 10% reduction in latency falls within the error bars, and might have arisen from experimental error.

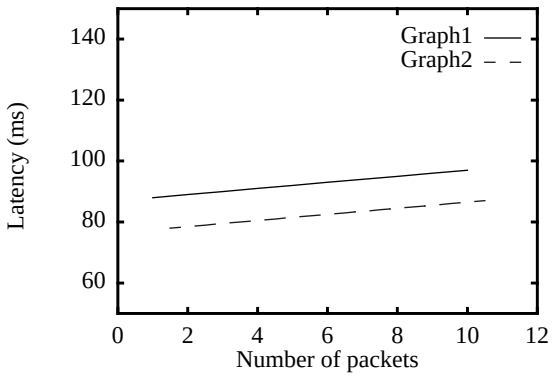
It is very useful to be able to place results in the context of an error margin, and it is essential to be able to do so when trying to determine the value of a new technique.

A related problem, which appears when measurements are taken, is mistaking measurement *precision* for measurement *accuracy*. For example, on many versions of Unix, `gettimeofday`() returns the current time in microseconds (its precision), but is only updated every ten milliseconds (its accuracy). Timing measurements taken using `gettimeofday`() on these systems will be rounded up (or down) to nearest multiple of ten milliseconds. In situations such as these, it is critical to be aware not only of how precise a measurement is, but also how accurate. On a system with a 10ms clock granularity, it is a waste of time to attempt to make distinctions at the microsecond level.

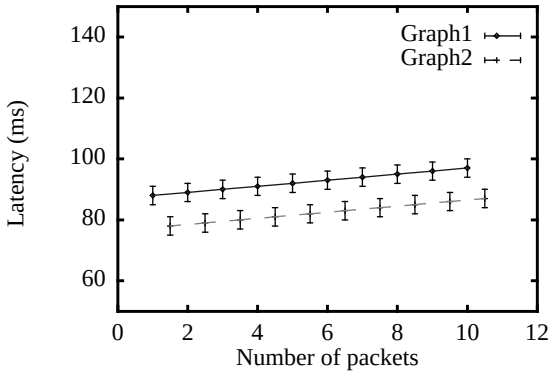
6.3 Probability Distributions and Testing

As stated above, normal distributions are commonly found in nature, but rarely found in computer science. When measuring experimental systems, one is more likely to encounter other types of distributions. Unfortunately, it is not a trivial task to correctly identify which distribution best models a given a dataset. From a statistical point of view, an *estimate* of the mean and standard deviation of can be calculated from measured data, but without knowing the actual distribution, it is impossible to calculate the *true* mean and standard deviation. Fortunately, there are simple methods for determining the distribution of a dataset.

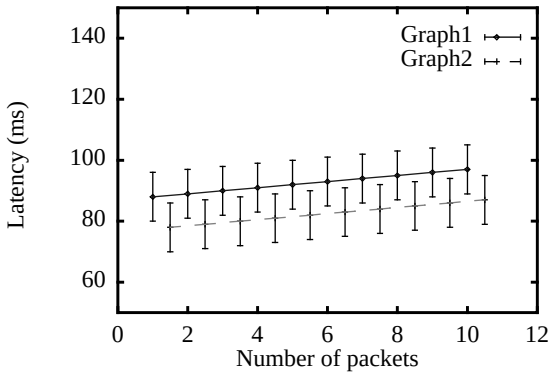
Plotting a histogram of the values in a sampled data set is easy way to get an idea of what type of distribution the data follows. Figure 1 shows examples of several common distributions with noticeably different shapes. Normal distributions (Figure 1a) are common in the natural sciences, and often represent the characteristics of repeated samples of a homogenous population. As mentioned in Section 6.1, skewed distributions often occur when some phenomenon limits either the high or low values in a distribution. Personal income is an example of a skewed distribution. An exponential distribution (Figure 1c) might be seen when modeling a continuous memoryless system, such as inter-arrival time of net-



A: Latency Improvement without error margins



B: Latency Improvement with small error margins



C: Latency Improvement with large error margins

Figure 2. Graphing and Error Margins. The value of the error margins will depict the results in completely different ways.

work packets, while a Poisson distribution results from a discrete rare occurrence. (Poisson distributions are seldom seen in computer systems research.)

The χ^2 (chi squared) test can be used to determine if sampled data follows a specific distribution. Given the sampled data points and the expected distribution of data points[4], the square of the difference between the number of points seen and the number of points expected, at each sampled value, is computed and summed. The smaller the sum, the better the two distributions match. Formally,

$$X^2 = \sum \frac{(sampled - expected)^2}{expected}$$

X^2 can be used to obtain a *p-value* from a family of χ^2 distributions. The larger the p-value, the higher the probability that the measured distribution matches the candidate distribution.

6.4 Summary

By presenting a single value, with no explanation, an author gives almost no information about the behavior of the system measured. At a minimum, by providing the number of samples taken, the mean value, and the standard deviation, the author allows the reader to begin to understand the data. By including more detail (e.g., mode, median, and distribution information), the author gives the reader more powerful tools to analyze and understand the behavior of the system.

However, without confidence intervals or error bars, it is difficult to compare sets of data. It is important for a reader to know not just the measured difference, but also the relative size of the margin of error. If the margin of error is smaller than the measured difference, the results show a true difference; if the margin of error is larger than the measured difference, the difference may just be due to experimental error.

Statistical analysis of results can be arcane. Often the simplest and most revealing thing to do is to plot the measured data; if the data forms a normal or multimodal distribution, it will be clear from a graphical depiction.

7 Presentation of Results

Performing a good experiment, collecting statistically sound data, and properly analyzing the results are three of the four important steps in conducting quantitative research; presenting the data in a coherent and illustrative manner is the final step. It is also, perhaps, the most important step, in that poorly presented data can result in misleading or misinterpreted results.

4. Methods for computing sample expectations can be found in any probability textbook (e.g., Larsen and Marx [Larsen86]).

The graphs shown in Figure 3 present the same data, but encourage dramatically different interpretations. In reality, the two different data sets have very similar values (within 3% of each other for all cases), yet this difference can be magnified by limiting the range of the dependent variable. Although the 3% difference here may be a genuine improvement, it is important to not misrepresent the magnitude of the improvement. In many cases, a 3% difference in performance is not interesting to systems researchers.

In order to clearly and accurately represent data, common sense dictates the following rules for graphical presentation:

- Use 0-based axes when data is plotted on a linear scale.
- Use log scales to depict values that range over several orders of magnitude.
- Label all axes clearly (noting the units shown and the scale if it is not linear).
- Use consistent graphical representation throughout a publication.
- Be wary of graphing packages that will put any value you specify on an axis; this can lead to meaningless scales on the axes.

As with any set of guidelines, there are exceptions. For example, in Figure 1c, the x-axis is linear although the data values span four orders of magnitude. Since the purpose of the graph is to show the shape of an exponential distribution, plotting it on a linear, rather than logarithmic scale makes sense. Similarly, we have omitted axis labels in Figure 3, to draw attention to the data values themselves and how the selection of Y values leads to dramatically differently-shaped graphs.

Even after applying common sense, selecting an appropriate representation for data is not always trivial. Edward Tufte's prime directive of graphical representation is "above all else show the data" [Tufte83]. There are two components to this: selecting an appropriate representation for the data and then using that representation in a clear and effective manner. Every graphical element (e.g., a pie-chart, bar graph, line graph, scatter plot) encourages a particular interpretation. For example, it is far too common to see graphs such as that shown in Figure 4. In this example, the selection of a line graph to represent three unrelated data values leads the reader to believe that the y value is some function of x. A bar graph would be a better way to present this data. Table 5 lists some guidelines for the selection of an appropriate graphical representation.

Once a suitable representation has been selected, it is important to display the data well. Again, we turn to Tufte who encourages the maximization of data ink (ele-

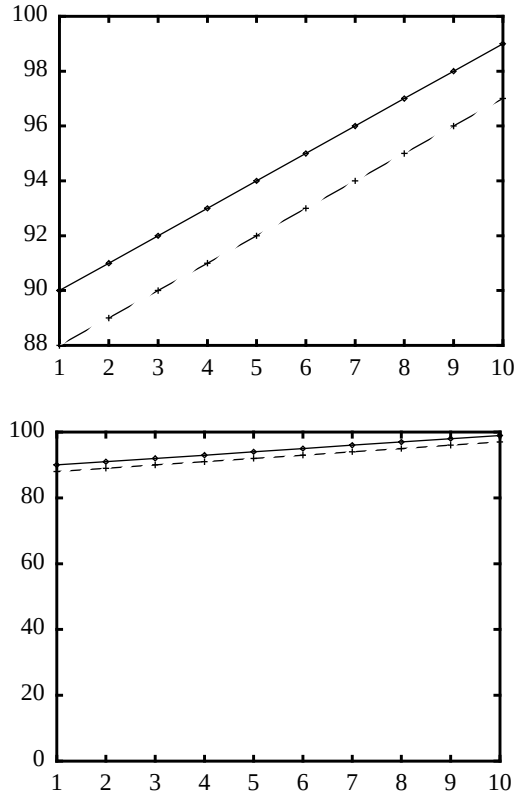


Figure 3. Misleading and Well-Labeled Y Axes. By limiting the Y axis to a narrow range of values, there appears to be a large difference between the two data sets in the top figure. The same data is shown in the lower with better Y axis selection. The data sets differ by only a small amount.

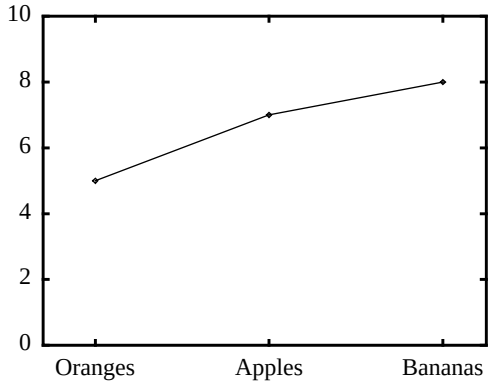


Figure 4. Improper graph selection. In this example, the three data points represent three unrelated values that are implicitly being compared. By representing this data as a line graph, we suggest that we are presenting y as a function of x.

ments on the graph that depict the actual data) and the minimization of chart ink (e.g., grid lines, labels, shading) [Tufte83]. A second suggestion is to use the "minimum effective difference." He suggests using varying shades of the same color (grey in most systems publications) as opposed to different colors or stipple patterns. Variations in shade are easily detected by the eye, and provide a much wider range of variation with greater simplicity [Tufte90].

Graph Type	When To Use
line graph	The datapoints represent a continuum of values on the X axis and the Y values are a function of the X values. It is instructive (and valid) to interpolate the values for points that have not been explicitly measured.
scatter plot	The interesting feature is the pattern or clustering (or lack thereof) of the data.
bar graph	Discrete values are being presented. Comparison between values is useful, but there is no constant relationship between the values presented.
pie chart	The issue of interest is how something decomposes into its constituent elements.

Table 5. Hints on Graph Selection. These are general guidelines for selection of an appropriate representation of data.

8 Previous Work

The problems we discuss here are not limited to computer science systems research. Cohen performed a survey of the 1990 AAAI conference [Cohen91], where he found that 41% of systems-centered papers (papers that discussed the behavior of a system that had been built) described only a single illustrative example of the system, without applying the system to any well-defined benchmark. His later book on empirical methods for Artificial Intelligence research [Cohen95] includes information on experimental design, statistical methods, and hypothesis testing.

Performing sound experimental systems research requires a firm grounding in statistics, available in any undergraduate statistics text [Walpole93, Larsen86]. "Back-of-the-envelope" calculations [Bentley84, Bentley86] provide a useful method for sanity checking results. Presenting data in a clear and effective manner is equally important. Tufte's books on information presentation [Tufte83, Tufte90] are widely regarded as the premier references on this topic.

We found that many of the problems that arise in the analysis of computer systems data occur in evolutionary biology as well. We found a kindred spirit in natural scientist Steven Jay Gould. His book, *Full House* [Gould96], clearly and entertainingly discusses how seemingly reasonable statistical arguments can be far off the mark. The book uses some of Gould's favorite examples, including the disappearance of .400 hitting in

major league baseball. The idea that some distributions have an implicit "left wall" or "right wall" is clearly and concisely explained.

Obtaining accurate and quantitatively useful data on a system is quite invaluable. It allows one to locate the bottlenecks limiting performance and monitor system resources [Lucas71]. Monitoring requires an analysis of vast amounts of statistics and data. In order to accomplish this effectively, a rigorous experimental procedure is required.

Rather than instrument separate benchmark programs, the system itself can be instrumented and continuously monitored. The Multics operating system [Saltzer70] was an early example of a system that was designed from the ground up for continuous monitoring and profiling, with the goal of regular analysis of the data and feedback into the design and tuning of the system.

We are not suggesting a radical change in systems research. Our work here can be considered a follow-on to, and a reiteration of, Levin and Redell's analysis of the 9th SOSP submissions [Levin83], which offered guidelines for constructing a submission to SOSP. Our goal is to induce authors to concentrate on statistical rigor and repeatability of experiments.

9 Summary

Computer Science should live up to its name—it should be a research science. This is especially true of computer systems research. Results of scientific research must, by definition, include a description of the experiments performed that is clear enough that others can repeat the experiments, and must perform sufficient statistical analysis that the numbers reported are believable.

Upon analyzing the proceedings of ten recent systems conferences, we were dismayed at the lack of rigor and comparability of the published work. In this paper we have outlined a suggested minimum set of requirements for systems research, and believe that the research produced by the community would be greatly improved by their adoption.

10 References

[ASPLOS92] *Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, October 1992.

[ASPLOS94] *Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 1994.

[Baker91] Baker, M., Hartman, J., Kupfer, M., Shirriff, K., Ousterhout, J., "Measurements of a Distributed File System," *Proceedings of the Thirteenth SOSP*, Pacific Grove, CA, pp. 198–212, October 1991.

[Bentley84] Bentley, J., "The Back of the Envelope," *Communications of the ACM*, *27*, 3, pp. 180–184, March 1984.

[Bentley86] Bentley, J., "The Envelope is Back," *Communications of the ACM, 29*, 3, pp. 176–182, March 1986.

[Cohen91] Cohen, P., "A Survey of the Eighth National Conference on Artificial Intelligence: Pulling Together or Pulling Apart?" *AI Magazine, 12*, 1, pp. 16–41, 1991.

[Cohen95] Cohen, P., *Empirical Methods for Artificial Intelligence*, MIT Press, Cambridge, MA, 1995.

[Gould96] Gould, S. J., *Full House*, Harmony Books, New York, NY, pp. 36–37, 1996.

[Larsen86] Larsen, R., Marx, M., *An Introduction to Mathematics Statistics and Its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[Levin88] Levin, R., Redell, D., "An Evaluation of the Ninth SOSP Submissions," *Operating Systems Review, 17*,3, pp. 35–40, July 1983.

[Lucas71] Lucas, Henry C., "Performance Evaluation and Monitoring," *ACM Computing Surveys, 3*, 3, pp. 79–91, September 1971.

[McVoy96] McVoy, L., Staelin, C., "lmbench: Portable Tools for Performance Analysis," *Proceedings of the 1996 USENIX Conference*, San Diego, CA, pp. 279–294, January 1996.

[OSDI94] *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Monterey, CA. USENIX Association, Berkeley, CA, November, 1994.

[OSDI96] *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA. USENIX Association, Berkeley, CA, November, 1996.

[Ousterhout90] Ousterhout, J., "Why Aren't Operating Systems Getting Faster As Fast As Hardware," *Proceedings of the 1990 Summer USENIX Technical Conference*, Anaheim, CA, pp. 247–256, June 1990.

[Saltzer70] Saltzer, J., Gintell, J., "The Instrumentation of Multics," *Communications of the ACM, 13*, 8, pp. 495–500, August 1970.

[SOSP91] *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, October 1991.

[SOSP93] *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, Asheville, NC, December 1993.

[SOSP95] *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, Copper Mountain, CO, December 1993.

[Tufte83] Tufte, E., "The Visual Display of Quantitative Information," Graphics Press, Cheshire, CT, 1983.

[Tufte90] Tufte, E., "Envisioning Information," Graphics Press, Cheshire, CT, 1990.

[USENIX94] *Proceedings of the Summer 1994 USENIX Conference*, Boston, MA, June 1994.

[USENIX95] *Proceedings of the 1995 USENIX Technical Conference*, New Orleans, LA, January 1995.

[USENIX96] *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[Walpole93] Walpole, R., Myers, R., *Probability and Statistics for Engineers and Scientists, 5th ed.*, Macmillan Publishing Company, New York, New York, 1993.

[Weicker84] Weicker, R. P. "Dhrystone: A Synthetic Systems Programming Benchmark," *Communications of the ACM, 27*, 10, pp. 1013–1030, October, 1984.

[Wittle93] Wittle, M., Keith, B. "LADDIS: The Next Generation in NFS File Server Benchmarking," *Proceedings of the Summer 1993 USENIX Conference*, Cincinnati, OH, pp. 111–128, June 1993.

11 Appendix

Test Type	Test Name	Occurrences of Test	Test Description
general	spec	82 (7.1%)	any SPEC test
	lmbench	26 (2.2%)	any of the lmbench test (fs, mem, net, sys)
	ouster	4 (0.3%)	any of the Ousterhout benchmarks (fs, sys)
cpu	dhrystone	8 (0.7%)	dhrystone test
	md5	0 (0.0%)	computing md5 checksum
	adhoc-cpu	41 (3.5%)	any other cpu measurement test
db	OO	8 (0.7%)	OO1 or OO7 benchmark
	postgres	5 (0.4%)	postgres or Sequoia 2000 benchmark
	tpc	4 (0.3%)	tp1, tpc-a, tpc-b, tpc-c, tpc-d
	adhoc-db	18 (1.5%)	any database microbenchmark (join, checkpoint, read, write)
fs	andrew	14 (1.2%)	(modified) andrew benchmark
	bonnie	4 (0.3%)	bonnie fs benchmark
	connect	1 (0.1%)	Connectathon benchmark
	laddis	10 (0.9%)	laddis or nhfsstone
	adhoc-fs	263 (22.6%)	any create, read, write, copy, find
mem	adhoc-bcopy	3 (0.3%)	any bcopy test
	adhoc-mem	59 (5.1%)	cache miss, tlb miss, cpi, stall cycles, num page faults, mem throughput
net	netperf	5 (0.4%)	the NetPerf tool
	ttcp	6 (0.5%)	tcp throughput measurement tool
	adhoc-rpc	15 (1.3%)	any ad hoc RPC measurement (null rpc, one byte, whatever)
	adhoc-packet	5 (0.4%)	any ad hoc packet filter test
	adhoc-net	126 (10.8%)	any ad hoc network measurement (latency, throughput)
para	dsm	15 (1.3%)	any subset of (FFT, SOR, TSP, Water, Barnes-Hut)
	nas	8 (0.7%)	NASA Ames parallel benchmark suite
	splash	13 (1.1%)	SPLASH suite (raytrace, ocean, etc.)
	adhoc-para	118 (10.1%)	any ad hoc parallel benchmark (matrix invert or multiply, FFT)
sys	appel	4 (0.3%)	the Appel-Li VM benchmarks (usually hand-implemented)
	dinero	2 (0.2%)	dinero simulator run
	kernel-build	5 (0.4%)	building a (well-specified) kernel
	aim	2 (0.2%)	AIM benchmark (simulation of multiple concurrent users)
	smalltalk-macro	1 (0.1%)	the Smalltalk “macro” benchmark
	webstone	0 (0.0%)	SGI’s webstone benchmark
	adhoc-ipc	6 (0.5%)	any ad hoc ipc test
	adhoc-syscall	11 (0.9%)	any ad hoc (null) system call
	adhoc-http	2 (0.2%)	any ad hoc web server test (other than Webstone)
	adhoc-sys	269 (23.1%)	any ad hoc system performance test

Table 6. Tests: the benchmarks and tests, and how may times each was used in the 1163 reports of the 213 papers surveyed.