Benchmark Precision and Random Initial State

Tomas Kalibera[1] Lubomir Bulej[1,2] Petr Tuma[1]

[1]Faculty of Mathematics and Physics
Charles University in Prague
Malostranske nam. 25
118 00 Prague, Czech Republic
{tomas.kalibera,petr.tuma}@mff.cuni.cz

[2]Institute of Computer Science
Czech Academy of Sciences
Pod Vodarenskou vezi 2
182 07 Prague 8, Czech Republic
bulej@cs.cas.cz

Abstract

The applications of software benchmarks place an obvious demand on the precision of the benchmark results. An intuitive and frequently employed approach to obtaining precise enough benchmark results is having the benchmark collect a large number of samples that are simply averaged or otherwise statistically processed. We show that this approach ignores an inherent and unavoidable nondeterminism in the initial state of the system that is evaluated, often leading to an implausible estimate of result precision. We proceed by outlining the sources of nondeterminism in a typical system, illustrating the impact of the nondeterminism on selected classes of benchmarks. Finally, we suggest a method for quantitatively assessing the influence of nondeterminism on a benchmark, as well as approach that provides a plausible estimate of result precision in face of the nondeterminism.

Keywords Performance Evaluation, Random Initial State, Statistical Analysis, Benchmark Precision.

1. INTRODUCTION

Software benchmarks are commonly used for empirical evaluation of performance. Typical uses of benchmarks include analysis of system behavior, evaluation of absolute system performance on selected classes of applications, or comparison of system performance on different implementations of an application [2,3]. In contrast to analytical methods or simulation, benchmarking provides results based on the behavior of a real system in a real environment. The benchmark results provide the values of various performance indicators that characterize the behavior of the system under particular workload, e.g. the typical duration of characteristic operations, memory consumption, etc.

To obtain the typical value of a performance indicator, a benchmark usually performs a number of measurements and calculates the typical value as an average, or median of the collected samples. Although this approach represents common practice, it involves making several hidden assumptions that are only rarely discussed explicitly.

The approach assumes that the values measured during a benchmark run can be considered independent and identically distributed samples of a random variable, which represents the performance indicator of interest. The justification for the assumption stems from the fact that due to complexity of contemporary hardware and software, the duration of a nontrivial operation is subject to frequently occurring but relatively small changes due to inherent nondeterminism in operation execution, as well as rarely occurring but relatively large changes due to external disruptions. The slightly changing values resulting from the nondeterminism in operation execution are considered representative for the repeated execution of the operation on a given hardware platform. The values distorted by external disruptions are, depending on the purpose of the benchmarking experiment, considered either extremal (with respect to the typical execution times of the exercised operation) or a part of the overall behavior of the system.

The typical value of a performance indicator would be then a value representative of the distribution of the random variable. Such values are usually the mean and the variance. Based on the kind of data provided by the benchmark, we can safely assume that they exist. The mean and the variance are typically estimated using average and sample variance. The typical value of a performance indicator is therefore a statistical estimate of the mean, and as such has a limited precision. Although the exact demands on the precision of the benchmark results depend on the particular use of the benchmark, we can safely state that a sufficient precision often needs to be as good as units of percents, simply because the very use of a benchmark suggests that a precise empirical evaluation, rather than a simple offhand estimate, is needed [5,12].

As follows from the Central Limit Theorem, the probability statements about the average can be approximated using the Normal distribution. The precision of the average can then be determined using the length of confidence interval for

This work was partially supported by the Grant Agency of the Czech Republic projects 201/03/0911 and 201/05/H014.

the mean, which can be estimated using a well known formula. To improve the precision of the benchmark results, a benchmark usually collects and computes the typical values using more samples. This approach can also be considered common practice, and has a theoretical foundation in the Weak Law of Large Numbers.

Consequently, a typical benchmark executes an operation multiple times and reports an average calculated from thus collected multiple samples as a result. The chance that the result is distorted by a rare sample to a degree outside the sufficient precision grows smaller with larger number of samples. In presence of extreme values caused by external disruptions, we can use robust statistic estimators such as median to mitigate the influence of the outliers, yet with the use of other estimators than average, the computations are more complex and without the theoretical backing of the Weak Law of Large Numbers and the Central Limit Theorem.

This model of a benchmark, however, fails to account for distortions caused by factors other than the inherent nondeterminism and external disruptions occurring during the operation execution. Consequently, for some benchmarks, the estimate of the mean and its precision calculated from thus obtained data do not truly represent the typical value of a performance indicator within the calculated error margin.

In this paper, we focus on improving the plausibility of the results by also accounting for distortions caused by a nondeterministic part of the initial state of the system that is evaluated. In Section 2 we point out the discrepancies between the common assumptions and the behavior observed on a simple benchmark, which suggest that the initial state is partially nondeterministic. Section 3 presents a measure of the influence of the random initial state on benchmark results and demonstrates the application of the measure on several examples of typical benchmarks. Section 4 outlines the possible sources of nondeterminism in the initial state and shows their impact on further examples of benchmarks. In Section 5, we suggest an approach to dealing with the nondeterministic part of the initial state and conclude the paper in Section 6.

2. SOMETHING IS ROTTEN

Consider the simple model of a benchmark introduced in Section 1, which empirically evaluates the duration of a specific operation execution. The duration is subject to random changes both inherent and external to the benchmarked operation. The simple model also assumes that apart from the inherent nondeterminism during execution of the operations, the duration of the operation depends only on the operation being performed. We show that this assumption is not sufficient and provide an extension to the simple model, where the duration of the operation execution also depends on the state of the system that is evaluated just before the operation execution. In itself, this extension is obvious rather than revolutionary, until the state is examined in more detail.

In a typical system, the state that can influence the duration of the operation execution consists of many parts, ranging from minute details such as the state of the processor branch prediction logic or internal memory cache or the paged state of the memory accessed during the operation execution, to significant hardware and software settings such as the bus clock speed or the virtual machine settings. Depending on the impact of the operation execution, the individual parts of the state can be broadly divided into two categories. Parts such as the state of the processor branch prediction logic or internal memory cache will most likely get changed by the operation execution, and are here termed as the *mutating* parts of the state. Parts such as the bus clock speed or the virtual machine settings will most likely stay unchanged by the operation execution, and are here termed as the *initial* parts of the state.

A benchmark typically handles the mutating and initial parts of the state in different ways. The setting of the mutating state is done by the benchmark itself, which introduces warm-up in addition to measurement. The operation execution is the same during warm-up and measurement, but samples are only collected during measurement. A long enough warm-up will set the mutating state so that the collected samples are representative for repetitive operation execution and independent of the mutating state before warm-up. The setting of the initial state is either hard coded in the benchmark or done outside the benchmark, forming a part of the system configuration to which the collected samples are representative.

A generally accepted requirement is that benchmark results should be reproducible. In terms of the mutating and initial parts of the state, reproducibility means performing long enough warm-up to make sure the mutating state is reproducible, and describing configuration precisely enough to make sure the initial state is reproducible, thus covering all parts of the state that can influence the duration of the operation execution [1, 4].

Before extending the simple model further, we can examine the assumptions made so far on an example of the FFT benchmark [10], which measures the duration of a Fast Fourier Transform of a predefined sequence.[1] Figure 1 shows individual samples collected by the benchmark during multiple runs, where a run is defined simply as a single execution of the benchmark application by the operating system. The horizontal axis is a sample index, the vertical axis is the sampled FFT computation time. Vertical lines denote new benchmark runs.

[1] The implementation of the FFT benchmark was run on Dell Precision 340 workstation with Intel Pentium 4 at 2.2 GHz and 512 MB RAM, running Fedora Core 2 with kernel 2.6.5 and gcc 3.3.3.

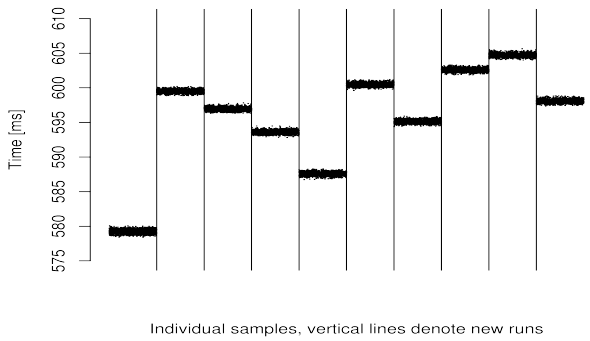

Figure 1. Impact of initial state on FFT benchmark results.

Figure 1 shows that the samples collected during a single run of the FFT benchmark are much closer to each other than the samples collected during different runs of the benchmark. The difference indicates that the initial state of the benchmark was not the same for each run, even though each run of the benchmark was executed on the same idle and isolated system, with disabled randomization of virtual memory address allocation, immediately after a reboot into a dedicated run level, using the same files and settings.

The extreme conditions of this experiment show that no reasonable effort could guarantee the initial state of the benchmark to be the same for each run. We have to cope with the deceitful reality that the initial state is partly random for each benchmark run.

As evidenced by the results of the FFT benchmark in Figure 1, the impact of the random initial state can completely overshadow the impact of the mutating state, so that taking more than several samples in a benchmark cannot improve the ability of the benchmark to describe real systems. This contrasts with the assumptions made in the simple model, where collecting more samples in a single run improves the benchmark precision. We therefore introduce a new model which recognizes the random initial state with a more realistic concept of benchmark precision discussed in Section 5.

3. QUANTIFYING THE PROBLEM

Although the influence of the random initial state can be often demonstrated graphically as shown in Figure 1, it can be difficult to notice for large data sets or in less obvious cases. Moreover, the graphical representation does not provide a simple measure of the degree to which the random initial state influences the benchmark results.

Considering the behavior of the FFT benchmark described in Section 2, a simplistic approach to quantification of the influence of random initial state would compare the standard deviation of all samples collected in all benchmark runs against the standard deviation of samples from individual benchmark runs.

However, the simplistic approach does not take into account the fact that the sample standard deviation calculated from all collected samples depends on the number of samples in each benchmark run and on the number of runs. This impairs the credibility of such a measure as it is unclear how to choose the number of samples and the number of benchmark runs.

An extreme choice would be to collect a high number of samples from a single run, thus rendering the method unusable. Instead, the choice of these numbers should be based on the measure itself, collecting a few samples from a high number of benchmark runs to reflect the impact of random initial state as well as collecting a high number of samples from a few runs to reflect the impact of the mutating state.

The dependency on the number of samples per run and the number of benchmark runs can be avoided when comparing standard deviation of samples taken from different runs against the standard deviation of samples from individual runs. Taking the above into account, we define the measure of the influence of the random initial state, termed *impact factor,* as the ratio of the standard deviation of samples from different benchmark runs to standard deviation of samples from individual benchmark runs. The value of impact factor greater than 1 indicates the influence of the random initial state on the benchmark results.

The calculation of the impact factor for a particular benchmark requires samples from multiple runs of the benchmark. To obtain plausible quantification of the influence of the random initial state on benchmark results, the calculation

Input:

1. m data sets corresponding to m benchmark runs, each containing n samples
2. number of samples c to choose randomly, $c < min(m,n)$
3. number of iterations k

Output:

1. estimate of the impact factor for the input data

Algorithm:

1. repeat k times
    1. randomly choose c data sets corresponding to c benchmark runs
        1. randomly choose 1 sample from each of the c selected data sets
        2. compute standard deviation SD1 from the c selected samples
    2. randomly choose 1 data set from the input data
        1. randomly choose c samples from the selected data set
        2. compute standard deviation SD2 from the c selected samples
    3. compute ratio SD1/SD2 of the two standard deviations
2. compute the impact factor as a median of the k ratios

Figure 2. Algorithm for estimating impact factor of random initial state.

of the impact factor requires relatively large amounts of data. In order to save time and resources, we calculate the impact factor using a bootstrap method described in Figure .

Table 1 shows the impact factor for several benchmarks and systems, and demonstrates that the influence of the random initial state on the benchmark results exists in a wide variety of benchmarks. The Marshaling benchmark measures the duration of marshaling of a string constant during a simple remote procedure call. The Ping benchmark measures the duration of a simple remote procedure call. The RUBiS benchmark measures the duration of an operation on an auction website [4].[2] We have used a modified version of the RUBiS benchmark which allowed us to track the response times of individual methods.

Table 1. Impact factor of random process initial state.

Benchmark	Runs	Samples per Run	Impact Factor (median)
FFT P4/FC2	150	2000	25.81
Marshaling P4/FC2	100	100000	2.61
Ping P4/FC2	100	100000	1.10
FFT P4/DOS	100	2000	1.06
FFT P4/W2K	100	2000	94.74
FFT IA64/Sarge	100	2000	35.91
RUBiS P4/FC2	15	15 min. ~ 5500	1.01

For each benchmark, we have executed a number of runs, as indicated in Table 1. In each run, we have collected between 2000 and 100000 samples, depending on the benchmark. In case of the RUBiS benchmark, the number of samples was determined by the default execution time limit. Where possible, the numbers of runs and samples were intentionally set high to provide more than enough data for the bootstrap. We have performed 10000 iterations of the bootstrap method on the input data, with the number of randomly chosen samples c set to $0.75m$ for increased robustness. The impact factor has been computed as the median of 10000 ratios of standard deviations of samples collected during different runs of the benchmark to standard deviations of samples collected during an individual run of the benchmark.

[2] The Pentium 4 Linux and Windows systems were Dell Precision 340 workstation with Intel Pentium 4 at 2.2 GHz and 512 MB RAM. The Itanium system was Dell PowerEdge 7150 server with two Intel Itanium processors at 800 MHz and 1 GB RAM.

The results indicate that the impact factor is very large in the FFT benchmark, except for DOS operating system where it is negligible, significant for the Marshalling benchmark, small for the Ping benchmark and negligible for the RUBiS benchmark. These results agree with the analysis of the sources of nondeterminism provided in Section 4.1.

4. SOURCES OF NONDETERMINISM

The influence of the random initial state on the benchmark results is somewhat contrary to the common understanding of a benchmark as, by and large, a reproducible process. To exclude the possibility of non-reproducible benchmark results discussed in Section 2, which points to the existence of the random initial state and its influence on the benchmark results, we further trace and explain some causes of the nondeterminism in the initial state. We also show that attempts to eliminate the sources of nondeterminism do not provide reliable, long term benefits.

4.1 Nondeterminism in Memory Allocation

One source of nondeterminism in the initial state is related to memory allocation. Among the many activities an operating system performs when starting a benchmark application is the allocation of memory for the code and the data of the benchmark. The allocation entails the selection of the virtual addresses for the code and the data of the benchmark and the assignment of physical pages to back the allocated virtual addresses. Even though neither the selection of virtual addresses nor the assignment of physical pages has to remain unchanged during the benchmark execution, it is likely to remain so, especially when the system that executes the benchmark has enough memory to avoid swapping. Even when the operating system assigns physical pages on demand, the assignment takes place during the warm-up phase of the benchmark and, again, is likely to remain unchanged during the data-collection phase of the benchmark.

The selection of virtual addresses and the assignment of physical pages can lead to a different distribution of cache hits and misses during the execution of the benchmark. The difference is mainly caused by the limited associativity of TLB (Translation Look-Aside Buffer) and memory caches, with the hardware mapping several physical addresses to the same cache slot. Since programs do not access their virtual pages in an uniform way, different assignment of physical pages to virtual addresses leads to different numbers of cache hits and misses, therefore influencing the benchmark results. This influence can be verified by relating the benchmark results with the values of processor-specific performance counters that keep track of TLB and memory cache hits and misses.

Figure 3 relates the counts of memory cache misses with the results of the FFT benchmark from Figure 1 for execution on two platforms, both with fixed selection of virtual address-

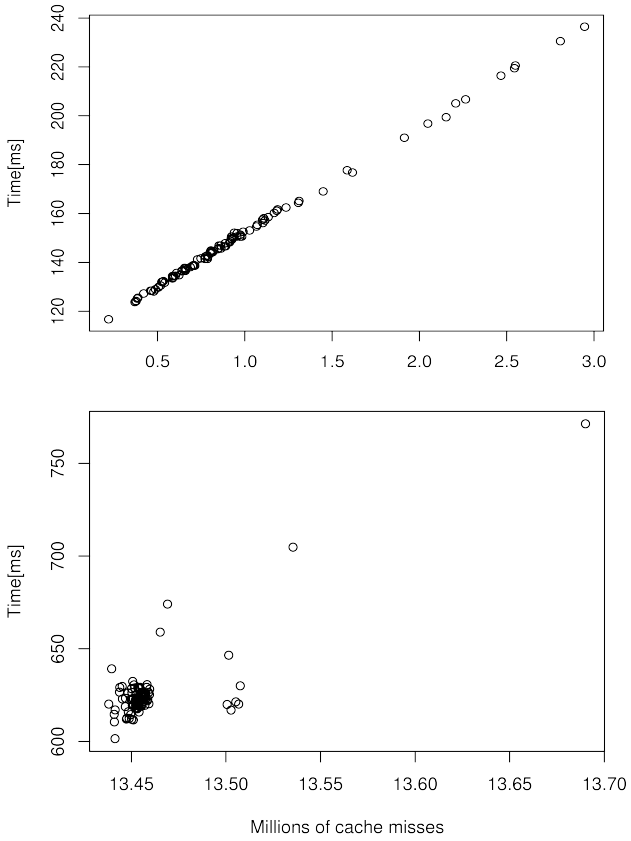

Figure 3. Correspondence of result fluctuation of FFT benchmark with memory cache misses.

es and random assignment of physical pages. The top plot in the figure is for the Intel Itanium processor and shows a strong positive linear dependency of measured times on memory cache misses. The bottom plot in the figure is for the Intel Pentium 4 processor and shows a weak positive correlation, however the results for this platform are only approximate due to a bug in the processor which prevents precise counting of memory cache events. The clear relationship visible in Figure 3, as well as the difference of impact factor for systems with different TLB and memory cache architectures in Table 1, confirms the impact of the memory cache misses on the benchmark results.

While the selection of virtual addresses by the operating system can be made deterministic and therefore reproducible at least on some operating systems, the assignment of physical pages to virtual addresses is not reproducible without nontrivial modifications of the operating system, which is obviously out of the question for many benchmarks.

4.2 Nondeterminism in Code Compilation

Another source of nondeterminism in the initial state is related to code compilation. The process of compilation and linking of a benchmark application into a binary is not necessarily reproducible even when using the same compilation and linking commands on the same benchmark sources.

Table 2 shows the effect of nondeterminism in compilation and linking on the Marshaling and Ping benchmarks from Table 1. The cause of nondeterminism in this particular example can be observed with the GNU C++ compiler [6] since version 3.3.1. The compiler uses random name mangling for symbols defined in anonymous name spaces. This can result in the linker linking the symbols in different order and placing them at different addresses in the binary. The placement of the symbols on different addresses in the binary is then reflected in placement of the symbols on different addresses in memory during dynamic linking, which in turn influences the benchmark results in a similar way as the nondeterminism in memory allocation.

Table 2. Impact factor of random initial state in compilation.

Benchmark	Runs	Impact Factor (median)
Ping	100	1.13
Marshaling	100	1.06

Although the cause of nondeterminism in this particular example can be avoided by using certain compiler options, it may not be possible for all benchmark sources. More importantly, the causes of nondeterminism described both in Section 4.1 and in Section 4.2 are difficult to avoid, which also makes it difficult to prove that they are the only causes of nondeterminism in the initial state [9]. In fact, more causes of nondeterminism in the initial state can be found depending on the benchmark and the system that executes the benchmark.

4.3 Nondeterminism is Unavoidable

The nondeterministic part of the initial state can be eliminated either by removing the individual sources of nondeterminism or by simulating the whole system. Both approaches have their drawbacks.

As outlined in previous sections, removing the sources of nondeterminism is not very feasible, mainly because we cannot ensure that all sources of nondeterminism in particular benchmark experiment have been identified and eliminated. Even though it may seem that the nondeterminism in memory allocation described in Section 4.1 could be eliminated by running a benchmark experiment immediately after system reboot, this solution is not sufficient on contemporary systems. Complex benchmarks, which require services such as databases or web servers, will never start in a deterministic fashion, even after system boot.

In case of simple benchmarks, many system services, such as page daemon, I/O daemons, or file system journaling daemon, are often provided by the operating system kernel and cannot be shut down. Moreover, these services are started concurrently with the order of their execution determined by the current hardware state, which will again result in nondeterministic initial state for a benchmark executed during system boot sequence.

The nondeterminism in code compilation described in Section 4.2 can be eliminated for some benchmarks, but may occasionally result in the compiler generating incorrect code. Also, the only way to find out whether the benchmark is still influenced by nondeterministic initial state is to run the benchmark multiple times and observe the differences in the results. Then it may actually be easier to obtain the results using the approach we describe further, which is based on analysis of the results from multiple runs of the same benchmark.

Without a sound approach to eliminating the causes of nondeterminism in the initial state, we may consider simulating the whole system, which would be a deterministic process and would allow setting identical initial conditions prior to each execution of a benchmark. While simulation is considered a useful tool in the area of performance evaluation, its application to benchmarking would be time consuming at best. Today's benchmarks, applications and system hardware are so complex that simulating even a second of the benchmark execution would consume an enormous amount of time and resources. Creating a simpler model or actually simplifying the benchmark or the application for the purpose of simulation defies the obvious goal to evaluate performance of real systems in a realistic scenario. In addition to that, specification of software and hardware that would allow designing such simulation would be hard to get for current systems mostly for licensing reasons.

This further underscores the fact that a benchmark should not be understood as a fully reproducible process, which has been the practice so far, but rather as a process that is inherently and unavoidably nondeterministic because of the existence of the random initial state and its influence on the benchmark results.

5. LIVING WITH NONDETERMINISM

The presence of nondeterminism in the initial state forces us to reconsider and ask what is the result and the precision of a benchmarking experiment.

Without considering the nondeterminism in the initial state, the answer to the question is rather straightforward. The benchmarking experiment consists of a single run of a benchmark, which collects a number of samples to determine the value of a performance indicator. These samples are representative for the repetitive execution of the measured operation as described in Section 1. The samples are considered independent and identically distributed, but the distribution is typically unknown. To estimate a single value which is a representative of the repeated execution of the measured operation, an average of the samples is typically used, for reasons outlined in Section 1. From the weak law of large numbers, even though the distribution is unknown, the average of the samples is a good estimate of the mean value of the underlying distribution, which is usually considered a representative value for the distribution. The precision of a benchmark is identical to the precision of the estimate of the mean value and can be assessed using the central limit theorem.

In presence of nondeterminism in the initial state, we can either attempt to eliminate the nondeterminism and thus achieve the deterministic initial conditions that have been so far silently assumed in common practice, or we can accept the nondeterminism as a trait of a real system and focus our effort towards obtaining meaningful results in presence of nondeterminism in initial state.

As discussed in Section 4.3, all sources of nondeterminism in a particular benchmark cannot be always reliably eliminated, not even considering the time and resources required for a detailed analysis of a system in order to identify the sources of nondeterminism. The other approach is to accept the influence of the nondeterministic initial state as a part of a real system and take it into account when benchmarking. For a performance indicator of interest, this requires determining the value and precision that are representative of both the repeated execution of the measured operation and the impact of the nondeterministic initial state.

5.1 Measuring with Nondeterminism

We have shown that the random initial state influences the benchmark results when benchmarking on real systems and that this influence cannot be easily eliminated. Because of the random initial state, multiple runs of a benchmark will yield different values of the performance indicator of interest as benchmark results. The value of the performance indicator reported by a single benchmark run is therefore a representative of the sampled values in a system with a particular realization of the random initial state, which includes the state of the operating system process with respect to assignment of physical to virtual pages as described in Section 4.1 and Section 4.2, and possibly other components.

In face of the described nondeterminism, an ideal benchmarking experiment would be set up so that the benchmark would be recompiled, restarted and warmed up before collecting each individual sample. In this setup, the collected samples would still be independent and identically distributed, therefore the average of all collected samples would still be a good estimate of the mean. This setup would ensure the repeatability of benchmarking experiments and the precision of the estimate of the mean would better reflect the behavior of the real system in a real environment.

However, there are two issues with the above scenario. First, if we only collect a single sample after warm-up phase during each benchmark run, we risk collecting a more or less distorted sample, caused by the inherent nondeterminism in operation execution or by an external disruption. To reduce the chance of invalidating the results because of a single distorted sample, we would have to collect a large number of samples. This leads us to the second issue related to the ideal scenario, which is efficiency of the benchmark experiment.

Described as it was, the ideal setup of the benchmarking experiment is very expensive to run. Imagine a Ping benchmark which uses CORBA (Common Object Request Broker Architecture) for remote communication. The compilation of the benchmark including the ORB (Object Request Broker) takes about 30 minutes, starting and warming up a benchmark takes several seconds and measuring one Ping request takes tens of microseconds.

Under such circumstances, collecting more than a few samples to obtain the sufficient precision of the estimate of the median would require an enormous amount of time. Or the other way around, the number of samples collected in a reasonable time may be far too low to obtain the sufficient precision.

Obviously, our goal is to reduce the number of runs to save time, but not so as to lower the precision by increasing the chance of collecting a distorted sample. With the knowledge of the influence of the random initial state on the benchmark results, we may save time by repeating the measured operation several times in each run to improve the precision of the result of a single run, which consequently allows us to reduce the required number of runs.

5.2 Efficient Benchmarking with Nondeterminism

Although the ideal setup of benchmarking in the face of nondeterminism outlined in Sections 4.1 and 4.2 requires rebuilding and restarting each benchmark to get precise and repeatable results, it is not always necessary to restart a benchmark run to obtain a new sample. The impact factor for a benchmark is not always so high as for the FFT benchmark and therefore there can be overlap between possible values of samples in different benchmark runs. Consequently, each benchmark run can contribute more than one sample to the estimate of the mean that is representative for both the repeated execution of the measured operation and the impact of the nondeterministic initial state.

In the following paragraphs we show that for an additive dependency of a sample on the random initial state, using this approach provides the same estimate of the mean as the one described in the model experiment from Section 5.1. Moreover, we can calculate the precision of the estimate and, given the number of samples required for benchmark warm-up, determine the optimal number of samples per benchmark run to obtain the most precise estimate of the mean. Concluding the section, we show how the additive dependency explains the influence of random initial state on benchmarks described in Section 3.

The additive model of dependency of a sample on the random initial state is defined as follows[3]:

1. the random initial state for each benchmark run is represented by a random sample a of a random variable A with finite variance and mean value
2. samples collected in a benchmark run with random initial state a (a is fixed) are independent, identically distributed samples of random variable $R(a)$, where $R(a) = a + X$, and X is a random variable with finite variance and mean value

The model applies only to samples after the warm-up phase of a benchmark. In the scope of this model, the ideal benchmarking experiment from Section 5.1 has the following interpretation:

1. for each benchmark run i, where $i = 1..k$, a single sample $r_i = a_i + x_i$ of random variable $(A+X)$ is collected
2. the average $\bar{r}_k$ of samples r_i estimates the mean value $E(A+X)$ of random variable $(A+X)$, which is representative for the repeated execution of the measured operation and the influence of the random initial state on benchmark results

The additive model of dependency is very similar to a random effect model with one-way specification from [11], which is based on normal distribution. The additive model we present here, however, does not assume the normal distribution of X and A. The distributions of X and $R(a)$ in our experiments were right-skewed, and thus could not be assumed normal.

For the additive model of dependency, we can show that the same estimate of the mean of the random variable $(A+X)$ can be obtained even when each benchmark run contributes more than one sample to the estimate. Consider a benchmarking experiment that consists of k benchmark runs, each collecting n samples of the performance indicator of interest. The i-th sample in j-th benchmark run is labeled $r_{j,i}$, $r_{j,i} = a_j + x_{j,i}$.

The average of all samples from all runs

$$\bar{r}_{k,n} = \frac{1}{k \cdot n} \cdot \sum_{j=1}^{k} \sum_{i=1}^{n} r_{j,i}$$

can be expressed as $\bar{r}_{k,n} = \bar{a}_k + \bar{x}_{k,n}$.

[3] For better readability, we use a relaxed notation, where random variables are sometimes denoted by lowercase letters, which is a common approach in literature on hierarchical models, such as [11]. We also use the word *sample* in a broader sense, not distinguishing rigorously between realizations and random variables; the precise meaning is always clear from the context.

From the central limit theorem, the distributions of $\bar{a}_k$ and $\bar{x}_{k,n}$ can be approximated as

$$\bar{a}_k \approx E\,A + N\left(0, \frac{var\,A}{k}\right) \text{ and } \bar{x}_{k,n} \approx E\,X + N\left(0, \frac{var\,X}{k \cdot n}\right).$$

From the properties of the normal distribution it follows that

$$\bar{r}_{k,n} \approx (E\,A + E\,X) + N\left(0, \frac{var\,A}{k} + \frac{var\,X}{k \cdot n}\right) = \\ = (E\,A + E\,X) + N(0,1)\sqrt{\frac{n \cdot var\,A + var\,X}{k \cdot n}}. \quad (1)$$

Therefore the mean value $E\,r_k$ of the average r_k is the mean value E(A+X) of random variable (A+X), which is representative for the repeated execution of the measured operation and the influence of the random initial state on benchmark results. The confidence interval for E(A+X) can be constructed as follows

$$\bar{r}_{k,n} \pm z_{1-\frac{\delta}{2}}\sqrt{\frac{n \cdot var\,A + var\,X}{k \cdot n}}, \quad (2)$$

where $z_{1-\delta/2}$ is a 1-δ/2 quantile of the normal distribution. With probability 1-δ, the interval contains the true value of E(A+X), which is the representative value of the performance indicator of interest and the correct result of a benchmark experiment. Even though the distributions of A and X are unknown, we can estimate their variances *var(A)* and *var(X)* because

$$var\,(\mathrm{R}(a)) = var\,(a+X) = var\,(X),$$

and thus by estimating variance of samples in any benchmark run we are also estimating the variance *var(X)* of the random variable X. Since we need to perform multiple runs of the benchmark, we can estimate the variance of each benchmark run and calculate the average of these estimates:

$$\tilde{S}(X)^2_{k \cdot n} = \frac{1}{k}\sum_{j=1}^{k} S_j(R)^2_k = \frac{1}{k}\frac{1}{n-1}\sum_{j=1}^{k}\sum_{i=1}^{n}(r_{j,i} - \bar{r}_{j,k})^2. \quad (3)$$

In addition, the following also holds

$$\mathrm{E}(R(a)) = \mathrm{E}(a+X) = a + \mathrm{E}(X).$$

To estimate the variance *var(A)*, let us consider a random variable M, the values of which are the mean values of the performance indicator in benchmark runs with different values of the sample a of the random variable A,

$$\mathrm{M} = \mathrm{E}_A(R(a)) = A + \mathrm{E}(X).$$

Since the following holds

$$var\,(M) = var\,(A + \mathrm{E}(X)) = var\,(A),$$

the estimate of variance *var(M)* also estimates the variance *var(A)*. As already mentioned, the samples of the random variable M are the mean values of different benchmark runs, and can be therefore estimated by the average of samples from the respective benchmark run.

We then estimate the variance *var(A)* as

$$\tilde{S}(A)^2_k = \frac{1}{k-1}\sum_{j=1}^{k}(\bar{r}_{j,n} - \bar{r}_{k,n})^2. \quad (4)$$

Substituting the estimates of variance (3) and (4) for the true but unknown variances in (2), the confidence interval for the mean value of the performance indicator of interest is then

$$\bar{r}_{k,n} \pm z_{1-\frac{\delta}{2}}\sqrt{\frac{n \cdot \tilde{S}(A)^2_k + \tilde{S}(X)^2_{k \cdot n}}{k \cdot n}}. \quad (5)$$

When the number of collected samples kn is small, the quantiles z of the normal distribution are replaced with quantiles of the t-distribution with $k(n - 1)$ degrees of freedom, because the true but unknown variances were replaced by sample variances.

Under the assumption of the additive model of dependency on the random initial state, the confidence interval (5) allows determining the precision of the benchmark result. The formula for constructing the confidence interval can also be used to determine the optimal number of samples that should be collected in each benchmark run.

To minimize the error of the estimate of the mean value of a performance indicator, we have to minimize the factor

$$\sqrt{\frac{n \cdot var\,A + var\,X}{k \cdot n}}. \quad (6)$$

We can define the cost of the benchmark experiment in terms of the total number of samples that need to be collected as

$$cost = k(w + n), \quad (7)$$

where w is the number of warm-up samples that have to be collected at startup of each benchmark run, but cannot be included into the evaluation; w also includes the price of starting a new benchmark run. By expressing k from (7), substituting it into (6), deriving by n and finding roots, we get the optimal value of n for the given *cost* and warm-up w

$$n = \left\lceil \sqrt{\frac{w \cdot \tilde{S}(X)^2}{\tilde{S}(A)^2}} \right\rceil. \quad (8)$$

Of particular interest should be the fact that the optimal number of samples does not depend on the value of *cost*, but only on the number of warm-up samples w and the values of $\tilde{S}(A)^2$ and $\tilde{S}(X)^2$. It is therefore possible to obtain benchmark results with better precision using existing results without a loss of efficiency with respect to the cost of the benchmarking experiment.

The formula also suggests that, for benchmarks where the variance of samples from different runs is much greater than the variance of samples from individual runs, increasing number of samples in runs does not help, which is the case of FFT benchmark (Figure 1). Also, we should note that the precision of the result obtained from (8) depends on the precision of the estimates of variances $\tilde{S}(X)^2_{k \cdot n}$ and $\tilde{S}(A)^2_k$.

Even though the additive model of the dependency on the random initial state allows for more efficient measurement (in contrast to the ideal benchmarking experiment described in Section 5.1) and determining the precision of the benchmark results, it remains to be decided how well can be a particular benchmark modeled under the assumption of additive dependency.

For this purpose, we can use the impact factor defined in Section 3. The additive dependency model is characteristic by shifting all the samples in a benchmark run by a constant. For each benchmark run with a different sample a of the random variable A the following holds

$$R(a) - \mathrm{E}(R(a)) = a + X - (\mathrm{E}X + a) = X - \mathrm{E}X.$$

Under the assumption of the additive dependency model, $(R(a) - \mathrm{E}(R(a)))$ has the same distribution with variance *var(X)* and mean value $\mathrm{E}X = 0$ for every sample a, which means that it does not depend on a particular benchmark run.

Remember the impact factor defined in Section 3, which provides a measure of difference between samples in different runs and samples in individual runs. We can apply the impact factor calculation on transformed data, in which we have subtracted the average of samples in one run from all samples in the run. If the impact factor for the transformed data is smaller than for the original data, we can consider the influence of the random initial state on that particular benchmark to follow the additive dependency model. If the value of the new impact factor tends to 1, we can say that the additive dependency model describes the influence of the random initial state on that particular benchmark well.

Table 3 shows the result of applying the impact factor calculation on transformed data of benchmarks described in Section 3. We can see that while on certain platforms the random initial state influences the results of the FFT benchmark to a very high degree (impact factor nearly 90), the additive dependency model describes the influence very well (impact factor calculated from transformed data tends to 1).

Table 3. Difference in impact factor calculated from transformed data.

Benchmark	Impact factor (original data)	Impact Factor (transformed data)
FFT P4/FC2	25.81	1.00
FFT P4/DOS	1.06	1.00
FFT IA64/Sarge	31.78	1.01
FFT P4/W2K	87.48	1.01
Marshaling P4/FC2	2.61	1.20
Ping P4/FC2	1.09	1.00

6. CONCLUSION

We have shown that the results of software benchmarks executed on contemporary computer architectures and operating systems are likely to be influenced by a random initial state of the system. Depending on the benchmark and the system that executes the benchmark, the influence of the random initial state can lead to non-realistic and non-repeatable benchmark results and an implausible estimate of the precision of the results. Due to its somewhat counterintuitive nature, this situation can remain unnoticed, especially when the benchmark is incorrectly understood as a deterministic and therefore reproducible process.

Experiments with different benchmarks representing the classes of scientific computation benchmarks, distributed middleware benchmarks and micro-benchmarks show that in presence of the random initial state, the traditional approach of providing a representative value of a performance indicator based on a single benchmark run provides results that are representative only for that particular run. Improving the precision by collecting more samples during a single run is counterproductive, because it consumes more time and resources and only improves the precision of the result with respect to that run.

Using a bootstrap method to calculate an impact factor of the random initial state on a benchmark, we can quantify how much a particular benchmark is susceptible to the influence of the random initial state. We point out that while the influence of the random initial state on the benchmark results can be traced and explained in detail, it may be unavoidable. We show that in such a case, it is possible to obtain a value of a performance indicator that is representative for the benchmarked application and the system it runs on using samples from multiple runs of the benchmark. Balancing the number of runs and the number of samples collected in each run allows us to increase the efficiency of the benchmarking experiment and achieve sufficient precision within given time.

References

[1] Buble, A.; L. Bulej; P. Tuma. 2003. “CORBA Benchmarking: A Course With Hidden Obstacles.” In

Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003) Workshop on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (Nice, France, April 22-26). IEEE, Piscataway, NJ.

[2] Bulej, L.; T. Kalibera; P. Tuma. 2005. "Repeated Results Analysis for Middleware Regression Benchmarking." *Performance Evaluation 60*, No. 1-4, May: 345-358.

[3] Bulej, L.; T. Kalibera; P. Tuma. 2004. "Regression Benchmarking with Simple Middleware Benchmarks." In *Proceedings of IPCCC 2004 Workshop on Middleware Performance* (Phoenix, AZ, USA, April 15-17). IEEE, Piscataway, NJ, 771-776.

[4] Cecchet, E.; A. Chanda; S. Elnikety; J. Marguerite; W. Zwaenepoel. 2003. "Performance Comparison of Middleware Architectures for Generating Dynamic Web Content." In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference* (Rio de Janeiro, Brazil, June 16-20).

[5] Distributed Object Computing Group. *Continuous Metrics for ACE+TAO+CIAO.* http://www.dre.vanderbilt.edu/Stats.

[6] Free Software Foundation. *The GNU Compiler Collection.* http://gcc.gnu.org.

[7] Frigo, M.; S.G. Johnson. *BenchFFT – A Program to Benchmark FFT Software.* http://www.fftw.org/benchfft.

[8] Giladi, R. and N. Ahituv. 1995. "SPEC as a Performance Evaluation Measure." *Computer 28*, No. 8, August: 33-42.

[9] Gu, D.; D. Verbrugge; E. Gagnon. 2004. "Code Layout as a Source of Noise in JVM Performance." In *Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA 2004) Workshop on Middleware Benchmarking* (Vancouver, Canada, Oct. 24-28).

[10] Mayer, R. and O. Buneman. *FFT Benchmark.* ftp://ftp.nosc.mil/pub/aburto/fft.

[11] McCulloch, C.E. and S.R. Searle. 2000. *Generalized, Linear and Mixed Models.* Wiley-Interscience, New York, NY.

[12] *OVM Predictability and Performance Benchmarking.* 2004. http://www.ovmj.org/bench.

Biography

Tomas Kalibera is a doctoral student at the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic. He received his master degree in computer science from the Charles University in 2002. His primary interests are design of component systems and performance evaluation of middleware. He is a member of the Distributed Systems Research Group of Charles University.

Lubomir Bulej received his master degree in electrical engineering from the Czech Technical University, Prague, Czech Republic in 2002 and is currently pursuing doctoral degree at the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague. Since 2002 he also holds a research assistant position at the Institute of Computer Science, Academy of Sciences of the Czech Republic. He is a member of the Distributed Systems Research Group of Charles University and his primary research interests include performance evaluation of middleware technologies and connectors in component-based software architectures.

Petr Tuma is a senior assistant professor with the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic. He received his master degree in electrical engineering from the Czech Technical University in 1994 and his doctoral degree in software systems from the Charles University in 1998. From 1998 to 1999, he worked as a researcher at INRIA Rennes in France. His primary interests include operating systems, component systems and middleware, with focus especially on design and performance evaluation. He is a member of the Distributed Systems Research Group of Charles University.