Duet Benchmarking: Improving Measurement Accuracy in the Cloud

Lubomír Bulej
Vojtěch Horký
Petr Tůma
Charles University
Department of Distributed and Dependable Systems
Prague, Czech Republic
{name.surname}@d3s.mff.cuni.cz

François Farquet
Aleksandar Prokopec
Oracle Labs
Zurich, Switzerland
{name.surname}@oracle.com

ABSTRACT

We investigate the duet measurement procedure, which helps improve the accuracy of performance comparison experiments conducted on shared machines by executing the measured artifacts in parallel and evaluating their relative performance together, rather than individually. Specifically, we analyze the behavior of the procedure in multiple cloud environments and use experimental evidence to answer multiple research questions concerning the assumption underlying the procedure. We demonstrate improvements in accuracy ranging from 2.3× to 12.5× (5.03× on average) for the tested ScalaBench (and DaCapo) workloads, and from 23.8× to 82.4× (37.4× on average) for the SPEC CPU 2017 workloads.

ACM Reference Format:
Lubomír Bulej, Vojtěch Horký, Petr Tůma, François Farquet, and Aleksandar Prokopec. 2020. Duet Benchmarking: Improving Measurement Accuracy in the Cloud. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3358960.3379132

1 INTRODUCTION

At the heart of various performance comparison activities is a measurement experiment, whose statistical nature involves an inherent trade off between execution time and sensitivity to differences in performance. Longer experiment times average over noise in the measurement data and provide more accurate results, but are also expensive both in terms of time and computing resources. Conversely, shorter execution times may cause the loss of sensitivity or report false alarms. This is a problem when automating performance test execution and evaluation [14, 22].

Importantly, the resource requirements for performance testing are not constant, but rather reflect the development activities, the test scenarios, and the desired level of sensitivity. To satisfy the changing resource requirements, it is therefore attractive to consider offloading the performance testing activities to the cloud.

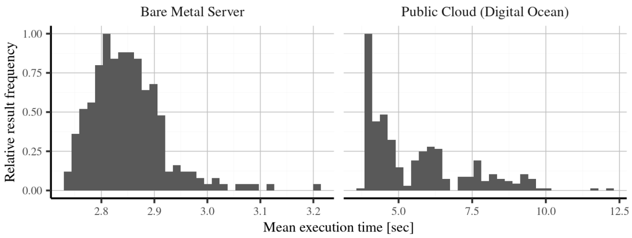

Figure 1: Distribution of observed mean execution times of the avrora benchmark, running on an otherwise idle bare-metal server and on a public cloud machine. Note the min-max range, which is about 16 % of the mean on the bare-metal server and about 150 % in the cloud.

A specific hurdle in this context is the fact that the cloud does not necessarily provide the performance stability required for performance testing. Performance measurements in the cloud are noisy, in part due to lack of control over hardware configuration, in part due to overhead of virtualization, but most importantly due to interference from colocated workloads of other tenants [15, 17, 18]. To illustrate this, Figure 1 shows the distribution of mean task execution times for iterations of an example benchmark from the DaCapo suite, both on a bare-metal server and on a virtual machine running in a public cloud.

Our earlier work [5] introduced the idea of the *duet measurement procedure*, which improves measurement accuracy in shared resource environments, such as virtual machine instances in the cloud. The procedure is based on the assumption that performance fluctuations due to interference tend to impact similar tenants equally, and attempts to maximize the likelihood of such equal impact by executing the measured artifacts in parallel. The subsequent computation filters out the fluctuations by considering the relative performance of the measured artifacts together.

The assumptions of the duet measurement procedure hinge on detailed technical properties of both the measurement platform and the executing workloads. In the cloud, such properties typically cannot be controlled or guaranteed, we therefore subject the procedure to a thorough experimental evaluation with the goal of analyzing the overall behavior and documenting the observed accuracy. Based on experimental evidence, we answer specific research questions

ICPE '20, April 20–24, 2020, Edmonton, AB, Canada
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6991-6/20/04…$15.00
https://doi.org/10.1145/3358960.3379132

concerning the assumption underlying the procedure and explain the technical mechanisms behind the observations:

- We demonstrate improvements in accuracy that range from 2.3× to 12.5× (5.03× on average) for the tested ScalaBench [27] (and DaCapo [3]) workloads, and from 23.8× to 82.4× (37.4× on average) for SPEC CPU 2017 workloads [29].
- We show that the accuracy improvements are due to the ability of the duet procedure to isolate synchronized interference, and that this interference arises with resource sharing.
- We evaluate how the specific patterns of concurrent execution and uneven resource utilization impact the ability of the duet procedure to measure performance differences.

As an essential overall contribution, our results indicate that cloud-based virtual machines can provide a viable platform for conducting an entire class of performance testing experiments based on comparing task execution times of benchmark workloads.

Section 2 provides additional background and motivation for performance regression testing as our specific application context. Section 3 presents an overview of the duet measurement procedure and the associated computations. Section 4 presents experimental evaluation answering specific research questions that naturally arise when using duet measurements and observing the effects on measurement accuracy. We review related work in Section 5 and conclude the paper in Section 6.

2 BACKGROUND AND MOTIVATION

The motivation for our work is performance regression testing, that is, the task of detecting performance changes between two versions of a software project. To this end, we use benchmark workloads to exercise both versions of the project, measuring and comparing task execution times of individual workloads between the two versions.

Essential to performance regression testing is robust performance change detection. The task execution times observed on a real system are influenced by different sources of variability at different levels of granularity – the comparison therefore relies on statistical hypothesis testing to accommodate the inherent variability in the data, and the performance testing procedure must ensure that significant sources of variability are sufficiently represented in the data [2, 4, 9].

To provide sufficient variability, benchmarks repeatedly execute the same task (in a single process) and measure the task execution time in each *iteration*. This captures variability caused by factors that can manifest at any time during benchmark execution, and which can influence the execution time of any iteration, such as scheduling, memory caches, or background load. In addition, benchmarks are executed repeatedly to obtain execution times from multiple benchmark *runs* (in multiple processes). This captures variability caused by factors that can change between runs, but rarely change within a single run, such as process memory layout, or decisions of managed platforms such as the Java Virtual Machine.

As a general rule, the variability in the observed execution times determines the magnitude of performance changes that can be reliably detected in a given time, or alternatively, the time needed to detect performance changes of a given magnitude. For a quick illustration of the computational resources needed for performance regression testing, we use the open source GraalVM project [23], where the developers contribute on average 5 merge commits per day and want to test these commits for performance changes on a selection of 60 workloads from multiple benchmark suites. When using Java workloads for tests at the 99 % confidence level, we can realistically assume to need data from 30 benchmark runs, each executing for 10 minutes (to get past some of the warm up effects). This sums up to 10 machine hours for a single experiment involving one version pair and one benchmark, and becomes 3000 machine hours per day for all experiments, which is an overwhelming figure.

To pare down the resource demands, we can limit the amount of testing actually done [12, 22], however, that alone may not solve the problem of infrastructure capacity limits. This is where cloud resources come into consideration, yet it is unclear if they are of any use for performance regression testing – the degree of control over the experimental platform, which allows obtaining accurate measurements on the local infrastructure, is not available in the cloud. Furthermore, cloud providers offer abstract virtual machine types that can run on different types of physical hosts [18], resulting in different execution times even for the same code. Finally, cloud virtual machines suffer from performance interference of neighbor workloads, which the virtualization technology cannot entirely eliminate. This also holds for continuous integration solutions executing in the cloud, such as Travis [30] or GitLab Runner [10].

In summary, we need a procedure that takes the characteristics of the cloud into account and makes it useful for performance testing, even if it only allows to quickly process many versions and flag suspect cases for more thorough measurements on dedicated infrastructure.

3 DUET MEASUREMENT PROCEDURE

Measurements in the cloud are subject to performance interference, which manifests as noise that may randomly affect any measured data. To account for the probabilistic nature of the interference, we have to repeat the measured operation enough times to obtain a representative sample of measurements, and then calculate confidence intervals for any values derived from the measurements. In experiments involving multiple workloads there is a risk of a systematic bias in the measured data if the probability of a workload being influenced by interference is not equal for all workloads. The current best practice uses randomized interleaving of workloads [2], which—for a long enough experiment—avoids the bias by equalizing the probability of interference for all workloads.

The *duet measurement procedure* also avoids bias by equalizing probability of interference, but is specifically tailored for experiments comparing performance of two (related) workloads. The two workloads are executed in parallel, inside a virtual machine with two virtual cores, with each workload restricted to one virtual core. The workloads are synchronized using a shared memory barrier, so that their measured operations always start at the same time. This setting ensures that any external interference on the virtual machine impacts both workloads simultaneously, which equalizes the probability of interference between the workloads for each paired measurement and thus avoids the bias immediately—rather than only for a long enough experiment.

We derive the confidence interval for the ratio of task execution times, which describes the relative performance of the two workloads, using a Monte Carlo procedure based on standard bootstrap confidence interval computation [13], explained in detail in [5]:

(1) For a pair of workloads x and y and an experiment with R runs of I iterations each, we denote $x_{r,i}$ and $y_{r,i}$ the task execution times of the respective workloads, measured in iteration $i \in 1 \ldots I$ of run $r \in 1 \ldots R$.
(2) For each r and i, we use the paired samples $x_{r,i}$ and $y_{r,i}$ to calculate the corresponding (speedup) sample $s_{r,i}$ of the ratio between task execution times of workloads x and y:
$$\forall r \in 1 \ldots R, \forall i \in 1 \ldots I : s_{r,i} = \frac{x_{r,i}}{y_{r,i}}$$
(3) For each run, we aggregate the speedup samples across iterations in a run by computing the geometric mean:
$$\forall r \in 1 \ldots R : gms_r = \sqrt[I]{s_{r,1} \cdot s_{r,2} \ldots s_{r,I}}$$
(4) We aggregate the geometric means across all runs in an experiment by computing the grand geometric mean:
$$ggms = \sqrt[R]{gms_1 \cdot gms_2 \ldots gms_R}$$
The value *ggms* represents a point estimate of the ratio of task execution times between workloads x and y, i.e., the relative performance of the two workloads.
(5) We use non-parametric bootstrap to estimate the percentile confidence interval for *ggms*, drawing with replacement from $gms_\bullet$ and computing $ggms^*$ (step 4 applied on the sample drawn from $gms_\bullet$) as Monte Carlo estimates for *ggms*.

When the confidence interval for *ggms* (mean ratio of task execution times) straddles 1.0, we consider the observed performance of the two workloads equal, otherwise we report a performance difference.

4 EXPERIMENTAL EVALUATION

We examine the duet measurements using multiple experiments designed to answer specific research questions. Before introducing the research questions and the experiments, we outline the experimental environment. For detailed information, please consult the online appendix [1].

The duet measurements target shared resource environments common in clouds, most of our measurements therefore execute in clouds. As the main cloud platform, we use the Amazon Elastic Cloud, specifically the `t3.medium`, `t3a.medium`, `m5.large` and `m5a.large` instance types. As our second cloud platform, we use the Travis CI infrastructure [30], which in turn uses otherwise unspecified Google Compute Engine platform machine instances. As our third cloud platform, we use the GitLab CI infrastructure [10] backed by Digital Ocean machine instances. In addition to the three public cloud platforms, we carry out measurements on a private cloud running the Proxmox Virtual Environment. Finally, we run bare metal measurements that are to represent the most stable baseline for comparison.

To approximate realistic workloads, we use benchmark suites – SPEC CPU 2017 [29] for statically compiled and optimized workloads, and ScalaBench [27] (with DaCapo [3]) for dynamically compiled and optimized workloads. From SPEC CPU 2017, we execute the rate workload variants (23 workloads in total). From ScalaBench and DaCapo, we execute all workloads except actors, batik, eclipse, tomcat, tradebeans and tradesoap, which fail for various reasons (20 workloads in total). We use the OpenJDK 1.8.0 JVM, run with fixed heap size and disabled garbage collector ergonomics, other virtual machine settings were left at their defaults.

To provide information on result variance, we execute all benchmarks multiple times (on average over 20 runs for each workload on the Amazon `t` instances, over 40 runs on the Amazon `m` instances, and over 100 runs on the other platforms), and use random samples of 10 runs for all computations. On the faster execution platforms (public cloud at full speed, private cloud, bare metal), we collect the timing of the first 100 iterations or 10 first minutes of execution within each run, whichever comes first. On the slower execution platforms (public clouds with token bucket processor allocation), it is 100 iterations or 60 minutes. We do not execute the SPEC CPU 2017 workloads on the Amazon `t` instances and on the Travis CI infrastructure, because both lack the computing power to execute the benchmark in reasonable time. For the SPEC CPU 2017 workloads, which exhibit virtually no startup artifacts, we use the timing of all iterations. For the ScalaBench workloads, which exhibit startup artifacts related to dynamic compilation, we discard the timing of the first half of iterations. We apply outlier filtering with winsorization in all computations, replacing at most one observation in a run with its nearest neighbor when that observation is further than 20% away from the min-max range of the remaining observations. Our bootstrap computations use 10000 replicates.

The constants above were determined by informal experiments to provide reasonable measurement time and reasonable stability across the workload spectrum. In an actual performance testing environment, the numbers would be chosen per platform and per workload using established procedures such as [11, 20], however, introducing this practice here would prevent us from comparing different measurement procedures under similar conditions.

4.1 RQ1: Accuracy Improvements

The very purpose of the duet procedure is to improve the accuracy of performance comparison experiments. Our first research question directly addresses this purpose: *Are the performance comparisons made with the duet procedure more accurate than performance comparisons done using standard methods ? (RQ1)*

The standard way to express the measurement accuracy is to treat the individual measurements as observations of a random variable with an unknown parameter of interest, such as the mean value. The goal of the measurement is to estimate this unknown parameter, and the accuracy of this estimate characterizes the overall measurement accuracy. An intuitive way to present the accuracy of the estimate, which we also use in this paper, is with confidence intervals [13]. For the duet measurements, we use the 99% confidence intervals for the mean of ratios computed with the procedure in Section 3. As a representative standard method that we compare against, we use the common 99% bootstrap confidence intervals for the difference of means, computed using the procedure in [4], with random measurement interleaving, as recommended in [2].

We collect the accuracy information using A/A measurements, that is, we compare two sets of measurements that use the same workload and the same instance type. For each workload and instance type, the comparison gives us two confidence intervals, one for the mean ratio of the workload execution times computed using the duet procedure, and one for the difference of the mean workload execution times computed using the standard method. By construction of the experiment, the two intervals must respectively straddle 1.0 and 0.0, and the width of the two intervals expresses the accuracy achieved by the two procedures.

A direct comparison of the two confidence intervals is hindered by the fact that the intervals produced by the duet procedure are centered around 1.0 but the intervals produced by the standard method are centered around 0.0. We therefore convert both types of confidence intervals to a value expressing their width relative to mean performance – for the mean of ratios interval $(ggms_{lo}, ggms_{hi})$ we report $ggms_{hi} - ggms_{lo}$, and for a difference of means interval $(diff_{lo}, diff_{hi})$ we report $(diff_{hi} - diff_{lo})/mean$, where $mean$ is the sample mean computed from all samples (all samples concern the same workload and can therefore be averaged).

Table 1: Average reduction in relative 99% confidence interval width from the standard procedure to the duet procedure, geomean.

Platform	ScalaBench	SPEC CPU 2017
Amazon m5.large	2.3×	26.6×
Amazon m5a.large	3.86×	82.4×
Amazon t3.medium	9.13×	–
Amazon t3a.medium	3.99×	–
GitLab CI	12.5×	23.8×
Travis CI	3.97×	–
Average	5.03×	37.4×

Figure 2 shows the distribution of the 99 % confidence interval widths on the public cloud platforms, aggregated across all workloads.[1] The distribution indicates that the duet procedure generally delivers more narrow confidence intervals and therefore better accuracy. Table 1 aggregates the improvement in accuracy for each platform and benchmark, expressed as the average reduction of the relative confidence interval width. For the ScalaBench workloads, the duet procedure computes on average 5.03 times more narrow intervals than the standard method. For the SPEC CPU 2017 workloads, the duet procedure computes on average 37.4 times more narrow intervals, in part because the workloads are much more stable and even small measurement fluctuations due to resource sharing are therefore more significant. Figure 3 provides more insight into this behavior by plotting the individual measurement samples for both the duet procedure and the standard method on one arbitrarily selected workload and platform combination. While the measurement fluctuations are always present, the samples collected in parallel by the duet procedure move (vertically) very much in tandem, almost perfectly matching the assumptions of the duet procedure.

To give an intuitive illustration of the improvement in accuracy, we look at the associated measurement costs. The mean confidence intervals tend to shrink with the square root of the sample counts – asymptotically, this holds due to the Central Limit Theorem, but here we refer rather to empirical observations at small sample counts, where we see similar behavior. A twofold improvement in accuracy at constant sample count therefore roughly corresponds to a fourfold reduction in sample count at constant accuracy. Note that the measurement costs are also impacted by different platform requirements – where the standard method requires sufficient resources to run a single workload copy, the duet procedure requires resources for two workloads executing concurrently.

4.2 RQ2: Synchronized Interference

At the core of the duet procedure is the idea to expose the compared workloads to the same interference. To achieve that, the procedure modifies the way the workloads are executed and the way the results are processed. We therefore need to determine whether the observed accuracy improvements are due to the synchronized interference, rather than a side effect of the modifications in workload execution and results processing. *Can we attribute the improved accuracy exhibited by the duet procedure to both workloads suffering from synchronized interference ? (RQ2)*

To isolate the contribution of synchronized interference from the other modifications introduced by the duet procedure, we use the existing measurements, but adjust the confidence interval computation from Section 3. Where the duet procedure normally computes ratios from measurements collected at the same time, we now perform a random shuffle and use ratios from unrelated measurements. That way, we preserve all other aspects of the duet procedure, but obtain results that do not benefit from synchronized interference.

Figure 4 shows the impact of shuffling on the distribution of the confidence interval widths. The distribution demonstrates that the duet procedure indeed benefits particularly from synchronized interference. We can also note that the confidence interval widths obtained with shuffling are very similar to the confidence interval widths from Figure 2 computed by the standard method. If we compute the aggregate improvement in accuracy after shuffling – an analogue of Table 1 but without synchronized interference – we obtain a total of 1.02 for the ScalaBench workloads and 1.03 for the SPEC CPU 2017 workloads, suggesting not only that the ability to deal with synchronized interference is the major factor contributing to improved accuracy, but also that other factors inherent to the duet procedure, such as the concurrent workload execution, are not a major detriment.

4.3 RQ3: Resource Sharing

The third aspect of the duet procedure we investigate is whether the presence of synchronized interference is due to resource sharing common in clouds, or whether some other property of our experiments may account for the observed behavior. *Is the presence of synchronized interference associated with the existence of other workloads that share the same computing platform ? (RQ3)*

The only way to control other workloads on the same platform in the public could is to rent an entire physical machine, however, that option also removes the virtualization infrastructure, making apples-to-apples comparison impossible. Instead, we therefore use private cloud measurements and control the utilization of the physical servers backing the virtual machine instances. In one set of

[1] We use the 99 % confidence level throughout the presentation, however, other confidence levels provide reasonably similar results.

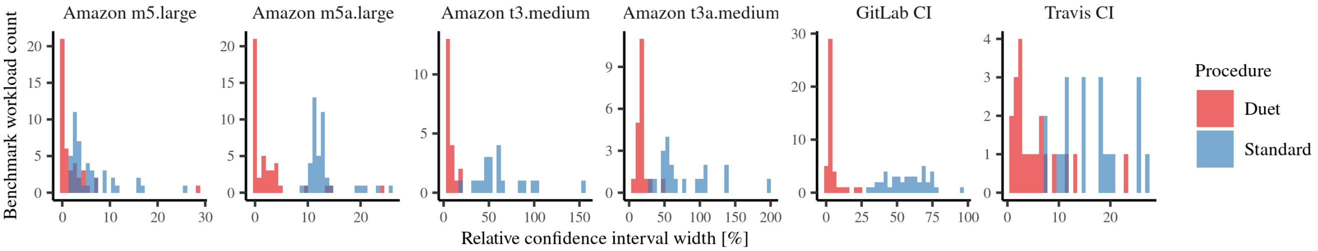

Figure 2: Accuracy expressed as relative 99% confidence interval width, 10 runs, aggregated across all workloads.

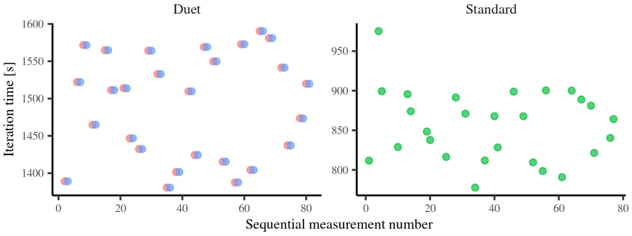

Figure 3: Individual measurement samples for the 503.bwaves_r workload on the Amazon `m5a.large` platform. Colors in the duet procedure distinguish samples collected in parallel.

measurements, we make sure each physical server runs only the measured workload. In the other set of measurements, we add a competing workload with the potential to saturate the physical server. Our competing workload is the composite configuration of the SPEC JBB 2015 benchmark, which generates a variable workload pattern across all cores of the physical server, moving between zero and peak utilization with a period of about 150 minutes. The workload approximates an enterprise business application and is therefore relevant in the cloud context.

Figure 5 demonstrates the impact of resource sharing on confidence intervals, again computed using either ratios from measurements collected at the same time, or ratios from unrelated measurements after a random shuffle. In the left-hand part of the plot, where the measurements were performed with resource contention, shuffling changes the confidence intervals significantly. In the right-hand part of the plot, where the measurements were without resource contention, shuffling has almost no effect. This confirms our hypothesis that the synchronized interference we observe and tackle with the duet procedure is indeed due to resource sharing.

4.4 RQ4: Measuring Differences

The duet procedure does not always utilize the computing resources evenly. Assume A/B measurements where the duet workloads differ in length, with A shorter and B longer. The concurrent workload execution phase, as long as A, will be followed by an isolated workload execution phase, as long as the remaining part of B. This makes the execution conditions for the two workloads differ – while A always competes for the shared resources, B executes partially with and partially without such competition. It may therefore finish faster than if the computing resources were utilized evenly, making the duet procedure underestimate the workload execution time ratio.

An underestimated workload execution time ratio is not necessarily a serious issue. Our motivation is the ability to detect performance changes during regression testing. In this context, it is enough to use the cloud to reliably detect the presence of a change, additional measurements to assess the magnitude can be performed in a controlled environment. We should, however, still seek to understand the impact of uneven resource on the measurements. *How does uneven resource utilization impact the estimated workload execution time ratio ? (RQ4)*

We answer the research question by arranging workloads with known execution time ratio in an A/B measurement and looking at the actual ratio measured and reported by the duet procedure. We do this first in the private cloud, where we have more control over the workload duration and resource utilization, and next in the public cloud, where we can use previous measurements.

Private cloud. To get sufficient control over workload duration and resource utilization, we move from the benchmarks to four entirely artificial workloads, designed to utilize a given resource for a given operation count. We refer to the four workloads as *integer* (an integer loop running entirely from level 1 caches), *float* (a floating point computation also running entirely from level 1 caches), *cache* (a linear memory walk over 4 MiB of data that mostly hits in the last level cache), and *memory* (a random memory walk over 64 MiB of data that mostly misses in the last level cache). The *integer* and *float* workloads are sensitive mostly to hyperthreading and power management, while the *cache* and *memory* workloads add sensitivity to competition on the memory resources.

We first calibrate the artificial workloads on the private cloud platform, obtaining operation counts that yield roughly 100 ms executions. For each artificial workload, we then execute A/B measurements where A executes the workload using the calibrated operation count and B executes the same workload using twice the count of A. For the artificial workload, the operation count translates directly into execution time, we would therefore desire to observe iteration times with the ratio of 2.0.[2]

As Figure 6 illustrates, the observed ratio of iteration times for the two workloads is indeed very close to 2.0. We can observe the

[2] Note that the relationship between operation count and execution time does not hold for the benchmark workloads, one reason why artificial workloads are used here.

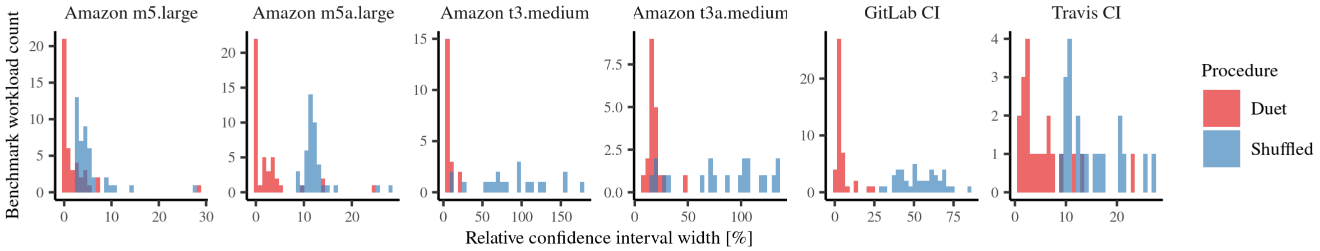

Figure 4: Impact of random shuffling on relative 99% confidence interval width, 10 runs, aggregated across all workloads.

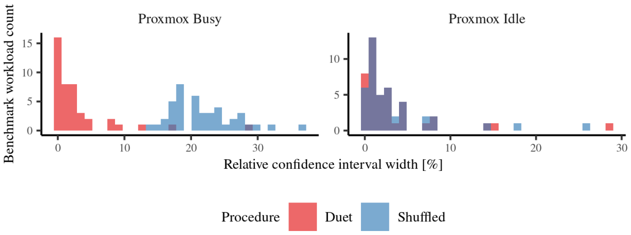

Figure 5: Impact of resource sharing on random shuffling in private cloud, idle vs busy with competing workload, expressed as relative 99% confidence interval width, 10 runs, aggregated across all workloads.

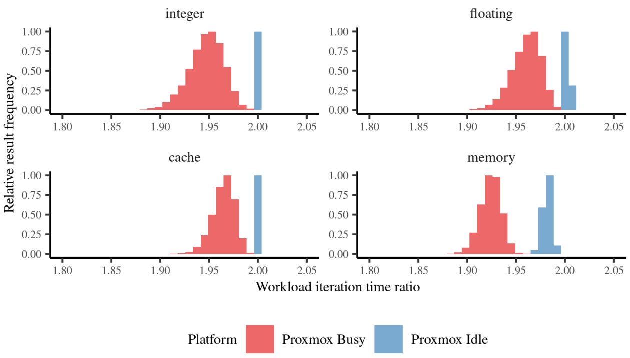

Figure 6: Distribution of observed mean iteration time ratios for individual artificial workloads in private cloud, idle vs busy with competing workload, 10 runs.

ratio decreasing slightly when the platform suffers from additional resource contention, generated again using the composite configuration of the SPEC JBB 2015 benchmark running across all cores of the physical server. This is most visible with the *memory* workload, which makes practical sense because out of the four artificial workloads, *memory* is most sensitive to memory bandwidth, which is shared across the entire physical server. We can conclude that on the local cloud, the impact of uneven resource utilization is negligible.

Public cloud. We can also assess the impact of uneven resource utilization using the previous A/A measurements on the public cloud. In the private cloud, we have constructed an A/B measurement where B was twice as long as A, and examined the ratio. Each A/B duet measurement had two phases, a concurrent phase where both A and B executed, and an isolated phase, where A already finished and B executed in isolation. Here, we observe that the concurrent phase of the A/B duet measurement resembles an A/A duet measurement, and the isolated phase of the A/B duet measurement resembles a standard isolated measurement of B. Both are measurements we have collected previously, we can therefore use the resemblance to construct a hypothetical A/B measurement scenario.

The ratios of mean iteration times for the public cloud platforms are in Figure 7. On GitLab CI, the ratios are close to 1.0, suggesting that the uneven resource utilization is not an issue. On the other public cloud platforms, the ratios are larger – in other words, the same workloads take longer when executed as A/A duet measurement than when executed using a standard isolated measurement. In the hypothetical A/B measurement scenario, this translates into an underestimated workload execution time ratio.

We attribute the difference between the platforms to two factors – hyperthreading and token bucket processor allocation. On the Travis CI and Amazon `m` platforms, the ratios range between 1.0 and 2.0, which corresponds to hyperthreading splitting the computing power of a single hardware core between two virtual cores for the duet workloads.[3] On the Amazon `t` platforms, the ratios exceed 2.0, likely because the token bucket processor allocation throttles the concurrent workloads executing on two virtual cores more than the isolated workloads executing on one virtual core.[4]

Returning to the research question, our results put an upper bound on how much we can underestimate the workload execution time ratio. For example, if an A/A execution takes 3 times as much time as A executing alone, and B executing alone takes 2 times as much as A, the desired ratio of 2.0 would instead be measured as $(3 + 1)/3 \approx 1.3$. Figure 7 suggests this would be an extreme case.

At the same time, our experiments provide a way to address this concern if required. Because the underestimated workload

[3] Although the Proxmox private cloud also uses hyperthreading, it does not have the same impact. This is because the private cloud schedules virtual cores across all physical cores, unlike the Amazon public cloud, which likely binds the virtual cores to the hardware threads of one physical core.

[4] Somewhat surprisingly, this would suggest that it is more cost efficient to use Amazon `t` instances as single-core rather than dual-core machines.

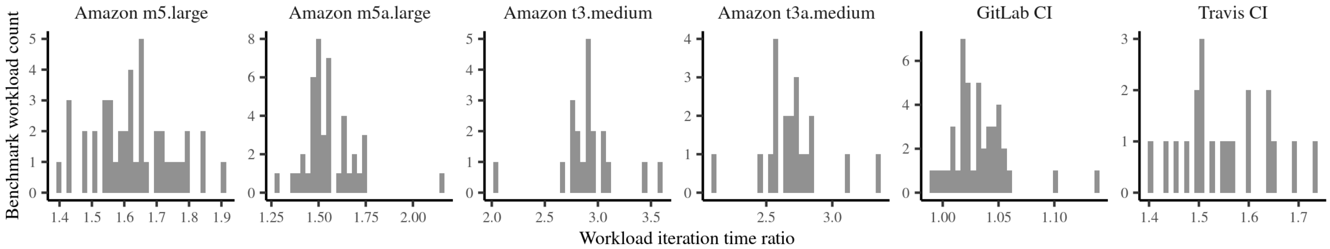

Figure 7: Distribution of observed ratios of mean iteration times between A/A duet procedure measurements and standard isolated measurements.

execution time ratio is associated with uneven resource utilization, we can simply adjust the duet procedure to continue (repeatedly) executing the shorter workload until the longer workload finishes, rather than leaving the resources of the shorter workload idle. This measure obviously removes the uneven resource utilization.

4.5 Discussion

The combined answers to the four research questions prove that the duet procedure improves performance comparison accuracy on shared resource platforms by relying on the synchronized nature of resource sharing interference. Our experiments suggest the assumption of synchronized interference is safe to make on many platforms – although it hinges on a multitude of technical details, these boil down to expecting that the platforms treat similar workloads in symmetrical situations equally.

On the flip side of the same argument, the duet procedure may not improve accuracy when comparing workloads with very different bottleneck resources, such as a CPU-bound workload and an I/O-bound workload. There is no reason to expect any resource sharing interference to impact most different resources equally. This is a threat to external validity of our results.

We can also argue that comparing workloads with different bottleneck resources is inherently fraught with issues. The relative performance of the workloads is more likely to change between platforms with different resource parameters, making comparison results less portable and therefore less useful.

A very general threat to both external and internal validity concerns the complex and diverse nature of public cloud platforms. Because cloud performance characteristics may vary significantly across platforms, our conclusions are potentially restricted to the platforms and workloads we use. Also, some of the effects we observe may be due to internal mechanisms we do not analyze. While characterizing every platform and workload is clearly not possible, we do use multiple platforms and workloads to at least partially address this concern.

We have mostly limited our experiments to the application of the duet procedure for change detection in the cloud, however, we do see more application opportunities both in the cloud and on bare metal systems. One interesting challenge is integration into CI/CD pipelines without dedicated virtual machine instances. Such platforms can possibly use fine-grained processor-scheduling policies in place of binding workloads to cores, and still achieve a reasonable comparison accuracy.

5 RELATED WORK

Our related work section includes a condensed version of an earlier analysis in [5]. We start with the paper by Laaber et al. [17], which investigates the accuracy achievable in the cloud with standard measurement methods, that is, when executing the evaluated workloads one after another with randomization as recommended by [2]. Laaber et al. demonstrate that when using the standard confidence interval overlap test with 95% confidence intervals for the mean, A/A testing needs fairly high experiment repetition counts to reduce the false alarm rate below 5%. The authors conclude that for most of their workloads, "small slowdowns (less than 5%) cannot reliably be detected in the cloud, at least not with the maximum number of instances (they) tested (20)" [17]. Our duet procedure improves on this result.

The work of Abedi and Brecht [2] shows how the ordering of trials can impact the experiment conclusions. Utilizing A/A testing, the authors show that possible regularity in performance interference can be incorrectly interpreted as actual difference in performance between alternatives. Randomized ordering of trials is proposed as a remedy. Our duet measurements similarly randomize the assignment of workloads to processors.

Existing research also often deals with the question of how many measurements to collect to achieve certain measurement accuracy, examples of recent work include He et al. [11] for virtual machine instances or Maricq et al. [20] for bare metal instances. Applying this work alongside our duet procedure is not necessarily straightforward, because the measurement accuracy metrics may not work with performance expressed as a ratio. Other than this, the work is complementary to our duet procedure.

In a broader sense, our work is connected to research on cloud performance characteristics. A study by Leitner and Cito [18] collects previously published observations on cloud performance and tests these observations with experiments. Especially relevant to our work are their conclusions on the performance stability of individual instances – this is shown to depend on the workload, with I/O-bound workload performance being sensitive to noisy neighbors, and CPU-bound workload performance depending mostly on actual allocated hardware.

Among studies that show significant performance variability in the cloud, many attribute that variability mainly to hardware heterogeneity. Cerotti et al. [6] investigate the effects of hardware heterogeneity on instance performance in the Amazon public cloud, showing that instances of the same type can be backed by different CPU types and differ in performance by 20% to 30%. Farley et al. [8] also examine the effects of hardware heterogeneity in the Amazon public cloud. Different CPU types are shown to differ in performance by as much as 280%. Differences of around 15% are observed among different instances with the same CPU types, similar differences are observed for the same instance across time. Ou et al. [24] report similar findings. For Amazon public cloud and performance differences between instances of the same type, CPU performance variability ranges between 10% and 20% and memory performance variability reaches as much as 270%. Other studies that concern various aspects of cloud performance variability include [7, 15, 19, 25, 26, 28]. Often, the purpose of the studies is to work towards efficient strategies of cloud resource allocation.

Although performance variability in public cloud is an accepted fact, the actual numbers observed in individual studies can rarely be compared directly due to differences in experimental settings. In our experiments, we have observed very little processor heterogeneity, and are mostly concerned with variability in time. If this were not the case, strategies to reduce processor heterogeneity in allocated instances can be utilized during testing.

Some authors propose mechanisms that help detect the presence of performance interference. Joshi et al. [16] measure an application in controlled conditions, constructing a throughput-vs-utilization curve. In real deployment, significant departure from that curve is interpreted as a sign of performance interference. Similarly, Mukherjee et al. [21] measure the performance characteristics of a lightweight probe deployed together with an application, and detect interference when the workload of the application cannot account for the changes in performance of the probe. Both strategies require some prior knowledge of the application to be deployed, and are therefore difficult to combine with performance testing of the type we consider.

6 CONCLUSION

Our experimental evaluation on 23 SPEC CPU 2017 workloads and 20 ScalaBench and DaCapo workloads suggests that duet measurement in the cloud is significantly more accurate than existing methodologies based on sequential measurements. Furthermore, our evaluation confirms that the improved accuracy is because the paired workloads are subjected to synchronized external interference. This external interference is an inherent property of running the workloads in the cloud, where the underlying resources are shared with the workloads of other users – whereas earlier techniques provide the same accuracy as duet measurement when there is no resource sharing, their accuracy deteriorates considerably in the presence of sharing.

The duet measurement procedure can introduce competition on the resources between the paired workloads and uneven resource utilization patterns. We show that these effects are either negligible or bounded and therefore do not prevent the detection of performance regressions.

Our observations imply that duet measurement is a viable technique for performance regression testing on both bare metal systems and in public cloud environments that support dedicated virtual machine instances. An interesting question is whether this technique can also improve accuracy of CI/CD pipelines without dedicated instances—we leave the answer to future work.

Acknowledgments. This work was partially supported by the ECSEL Joint Undertaking (JU) under grant agreement No 783162 and the Charles University institutional funding (SVV).

REFERENCES

[1] Online Appendix. 2020. http://arxiv.org/abs/2001.05811.

[2] A. Abedi and T. Brecht. 2017. Conducting Repeatable Experiments in Highly Variable Cloud Computing Environments. In *ICPE*. ACM.

[3] S. M. Blackburn, R. Garner, C. Hoffmann, et al. 2006. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. In *OOPSLA*. ACM.

[4] L. Bulej, T. Bureš, V. Horký, et al. 2016. Unit Testing Performance with Stochastic Performance Logic. *Automated Software Engineering* (2016).

[5] L. Bulej, V. Horký, and P. Tůma. 2019. Initial Experiments with Duet Benchmarking: Performance Testing Interference in the Cloud. In *MASCOTS*.

[6] D. Cerotti, M. Gribaudo, P. Piazzolla, and G. Serazzi. 2012. Flexible CPU Provisioning in Clouds: A New Source of Performance Unpredictability. In *QEST*.

[7] J. Ericson, M. Mohammadian, and F. Santana. 2017. Analysis of Performance Variability in Public Cloud Computing. In *IRI*.

[8] B. Farley, A. Juels, V. Varadarajan, et al. 2012. More for Your Money: Exploiting Performance Heterogeneity in Public Clouds. In *SoCC*. ACM.

[9] A. Georges, D. Buytaert, and L. Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. In *OOPSLA*.

[10] GitLab Inc. 2019. GitLab Runner. https://about.gitlab.com.

[11] S. He, G. Manns, J. Saunders, et al. 2019. A Statistics-Based Performance Testing Methodology for Cloud Applications. In *ESEC/FSE*. ACM, New York, NY, USA.

[12] C. Heger, J. Happe, and R. Farahbod. 2013. Automated Root Cause Isolation of Performance Regressions During Software Development. In *ICPE*. ACM.

[13] T. Hesterberg. 2014. What Teachers Should Know about the Bootstrap: Resampling in the Undergraduate Statistics Curriculum. *arXiv:1411.5279 [stat]* (2014).

[14] P. Huang, X. Ma, D. Shen, and Y. Zhou. 2014. Performance Regression Testing Target Prioritization via Performance Risk Analysis. In *ICSE*. ACM.

[15] A. Iosup, N. Yigitbasi, and D. Epema. 2011. On the Performance Variability of Production Cloud Services. In *CCGRID*.

[16] K. Joshi, A. Raj, and D. Janakiram. 2017. Sherlock: Lightweight Detection of Performance Interference in Containerized Cloud Services. In *HPCC*.

[17] C. Laaber, J. Scheuner, and P. Leitner. 2019. Software Microbenchmarking in the Cloud. How Bad is it Really? *Empirical Software Engineering* (2019).

[18] P. Leitner and J. Cito. 2016. Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Trans. Internet Technol.* 16, 3 (2016).

[19] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann. 2011. What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings. In *CLOUD*.

[20] A. Maricq, D. Duplyakin, I. Jimenez, et al. 2018. Taming Performance Variability. In *OSDI*. USENIX Association, Berkeley, CA, USA.

[21] J. Mukherjee, D. Krishnamurthy, and M. Wang. 2017. Subscriber-Driven Interference Detection for Cloud-Based Web Services. *IEEE Trans. on Network and Service Management* 14, 1 (2017).

[22] A. B. D. Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney. 2017. Perphecy: Performance Regression Test Selection Made Simple but Effective. In *ICST*.

[23] Oracle. 2019. GraalVM Repository at GitHub. https://github.com/oracle/graal.

[24] Z. Ou, H. Zhuang, A. Lukyanenko, et al. 2013. Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds. *IEEE Trans. on Cloud Computing* 1, 2 (2013).

[25] S. Ristov, R. Mathá, and R. Prodan. 2017. Analysing the Performance Instability Correlation with Various Workflow and Cloud Parameters. In *PDP*.

[26] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. 2010. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *VLDB Endow.* 3, 1-2 (2010).

[27] A. Sewe, M. Mezini, A. Sarimbekov, and W. Binder. 2011. Da Capo Con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine. In *OOPSLA*. ACM.

[28] S. Shankar, J. M. Acken, and N. K. Sehgal. 2018. Measuring Performance Variability in the Clouds. *IETE Technical Review* 35, 6 (2018).

[29] Standard Performance Evaluation Corporation. 2017. SPEC CPU 2017. https://www.spec.org/cpu2017.

[30] Travis CI, GmbH. 2019. Travis CI. https://travis-ci.com.