Performance Testing for Cloud Computing with Dependent Data Bootstrapping

Sen He
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
sen.he@utsa.edu

Tianyi Liu
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
tianyi.liu@utsa.edu

Palden Lama
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
palden.lama@utsa.edu

Jaewoo Lee
Department of Computer Science
University of Georgia
Athens, USA
jaewoo.lee@uga.edu

In Kee Kim
Department of Computer Science
University of Georgia
Athens, USA
inkee.kim@uga.edu

Wei Wang
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
wei.wang@utsa.edu

***Abstract*—To effectively utilize cloud computing, cloud practice and research require accurate knowledge of the performance of cloud applications. However, due to the random performance fluctuations, obtaining accurate performance results in the cloud is extremely difficult. To handle this random fluctuation, prior research on cloud performance testing relied on a non-parametric statistic tool called bootstrapping to design their stop criteria. However, in this paper, we show that the basic bootstrapping employed by prior work overlooks the internal dependency within cloud performance test data, which leads to inaccurate performance results.**

We then present *Metior*, a novel automated cloud performance testing methodology, which is designed based on statistical tools of block bootstrapping, the law of large numbers, and autocorrelation. These statistical tools allow *Metior* to properly consider the internal dependency within cloud performance test data. They also provide better coverage of cloud performance fluctuation and reduce the testing cost. Experimental evaluation on two public clouds showed that 98% of *Metior*'s tests could provide performance results with less than 3% error. *Metior* also significantly outperformed existing cloud performance testing methodologies in terms of accuracy and cost – with up to 14% increase in the accurate test count and up to 3.1 times reduction in testing cost.

I. Introduction

Cloud computing is widely adopted today due to its high cost-efficiency. To effectively utilize the cloud, cloud users need to have accurate knowledge of the performance of their applications. Accurate knowledge of cloud performance allows them to determine whether a virtual machine (VM) configuration (e.g., type and count of VMs) or an auto-scaling policy satisfy their performance requirements [1–3]. Accurate performance data are also required for cloud research to evaluate new optimization algorithms or to be used as the training and testing data to develop new models [1, 4–9].

Performance testing is a standard procedure to obtain the performance for any applications [10, 11]. For a cloud application, performance testing can be used to obtain its performance results as point estimates. That is, to obtain the mean or the percentile of its performance, such as the mean throughput or 90%ile execution time [9, 12, 13]. There are two fundamental requirements for cloud performance testing. First, the performance results should be accurate. Second, as cloud performance testing also incurs cloud usage expenditure, this testing should not cause excessive cost.

Performance testing is typically conducted by repeatedly executing the application-under-test (AUT) with a set of representative workloads/inputs until a *stop criterion* deems that the results obtained from the test are accurate. This stop criterion is the key to ensure the above two requirements are satisfied. However, as shown in prior work, due to the random performance fluctuation [12–17], it is extremely challenging to design good stop criteria for cloud performance testing.

To handle the random performance fluctuation, prior studies relied on non-parametric statistics tools to design their stop criteria. In particular, prior studies have employed non-parametric bootstrapping, which is a re-sampling technique used to calculate the confidence interval (CI) of a performance testing result (e.g., the CI of the mean performance) [9, 12, 13]. The confidence interval is typically viewed as the margin-of-error [18, 19]. Therefore, if the width of the confidence interval is smaller than a (user-) predefined maximum allowed error, the performance result is deemed accurate enough and the performance test can be stopped.

Unfortunately, prior work had shown that the performance testing methodology using bootstrapping cannot always provide accurate performance results [13, 20]. Prior work concluded that the inaccuracy was partially caused by the test's incomplete coverage of the cloud performance fluctuations. However, in our research, we discovered that there is another fundamental issue related to the bootstrapping methodology.

In this paper, we first present an analysis of the existing bootstrapping-based testing methodology [12]. This analysis reveals a critical issue with this methodology that is unknown to the current research – *the overlooked internal dependency*

within the performance test data. Because cloud performance fluctuation is mainly caused by the hardware resource contention from multi-tenancy (i.e., cloud applications/VMs sharing hardware) [15, 21], the performance data obtained from the tests are internally correlated. That is, the performance data from continuous tests within a short period are similar to each other, and the performance fluctuations may also have repeated patterns. However, the basic bootstrapping employed by prior studies does not consider this internal dependency, causing incorrectly calculated confidence intervals, which in turn, lead to incorrect performance results. Consequently, a new performance testing methodology that considers the internal dependency of cloud performance tests is required.

Moreover, our analysis also shows that an advanced bootstrapping technique originally designed for time series data, called Block Bootstrapping, can retain the internal dependency during the re-sampling [22]. Hence, it may provide more accurate performance results. Nonetheless, blindly applying block bootstrapping does not guarantee accurate results, as its block size must be tuned for individual cloud applications and cloud platforms. Moreover, enough tests must also be conducted to fully cover the potential cloud fluctuations.

Based on the above analysis, we developed *Metior*, a novel automated cloud performance testing methodology. *Metior* has two components. The first component is the new stop criterion to determine when performance tests can be stopped and accurate performance results are obtained. The new stop criterion is based on block bootstrapping and the "law of large numbers." Block bootstrapping allows *Metior* to properly consider the internal dependency of performance test data, while the "law of large numbers" ensures good coverage of cloud performance fluctuations. To handle the varying block size, *Metior* employs a novel technique that can automatically determine the best block size for each cloud application and platform [23]. The second component is the low-cost test execution strategy. As the performance of continuous tests is similar to each other, there is no need to continuously execute the AUT. Therefore, *Metior* executes the AUT in small intervals/periods (one day) and intermittently (4 executions per hour) to reduce the overall number of executions and the associated cloud usage cost. We also provided a systematic approach to determine the interval length and the frequency of the intermittent execution using the aforementioned best block size and autocorrelation [24]. *Metior* is implemented as a fully automated performance tester for several public clouds, including Amazon Web Services (AWS) [25], Google Cloud [26], and Chameleon cloud [27].

We evaluated *Metior* with six benchmarks on two public clouds, AWS [25] and Chameleon [27], using six different VM configurations. The results show that *Metior* can provide accurate performance results – among the thousands of tests conducted, 98% of them provided performance results with less than 3% errors. *Metior* also significantly outperformed existing cloud performance testing methodologies in terms of accuracy and cost – it could increase accurate test count by up to 14% and reduce the testing cost by up to 3.1 times. We also applied *Metior* to a state-of-the-art cloud performance prediction technique. The evaluation results showed that by providing more accurate training data, *Metior* could improve the accuracy of the prediction technique by 17.3% on average.

The contributions of this paper include:

1. An analysis of the basic bootstrapping which reveals that the overlooked internal dependency of cloud performance test data caused inaccurate cloud performance results.

2. A reliable and automated cloud performance testing methodology, *Metior*, which is designed based on block bootstrapping, the law of large numbers, and autocorrelation.

3. A thorough evaluation of *Metior* on two public clouds with 33 different benchmarks/VM configurations to show the accuracy and cost benefits of *Metior*.

4. A case study showing how *Metior* benefits recent cloud research by bringing in reliable performance results.

The rest of this paper is organized as follows: Section II provides the analysis on the basic bootstrapping; Section III presents the design of the *Metior*; Section IV provides experimental evaluations. Section V presents a case study of applying *Metior* in cloud research. Section VI discusses the limitation of *Metior*. Section VII discusses related work, and Section VIII concludes the paper.

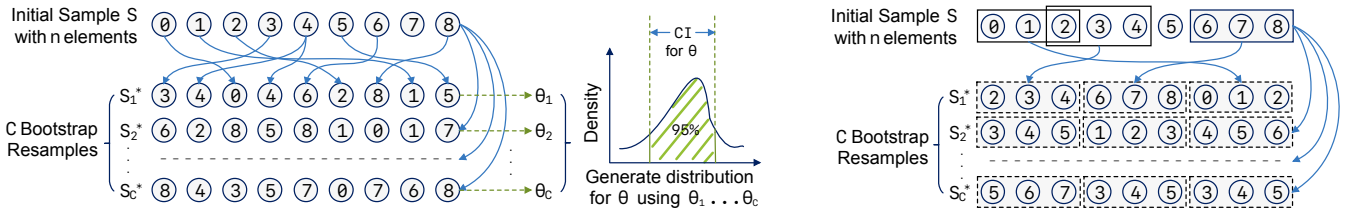
II. Analysis of Bootstrapping Cloud Performance

A performance test typically involves repeatedly executing the AUT with one or more representative inputs. For each execution, the *performance data*, such as the execution time, latency, or throughput, are recorded. Based on the performance data from a series of executions, the *performance testing result*, such as the mean latency or 90%ile execution time, can be calculated. The confidence interval (CI) of the performance testing result can also be calculated, which is usually viewed as the margin-of-error of this result [18, 19]. The width of this CI is usually used as the stop criterion of the performance test– if the CI width is smaller than a *user-defined maximum allowed error* (e.g., 3% maximum error), then the test can be stopped [12, 19]. In this paper, we call a performance testing result *accurate* if it indeed has an error less than the maximum allowed error.

For non-cloud performance testing, the CIs are usually computed using t-value or z-value, assuming the performance is normally distributed [19]. However, the performance of cloud applications is usually not normal [13]. Therefore, current research employed a non-parametric statistics technique, bootstrapping (BT), to calculate the CIs for cloud performance testing results [9, 12, 28, 29]. Unfortunately, the performance tests conducted with the current bootstrapping-based methodology tend to provide inaccurate results. This section provides an analysis of the cause of this inaccuracy.

A. Background on Bootstrapping

Basic Bootstrapping. Bootstrapping (BT) is essentially a resampling technique. Without loss of generality, here we show how to use bootstrapping to determine the CI of the 90%ile execution time of some performance test data with a



(a) Basic Bootstrap: resampling and computing CI w. 95% confidence level

(b) Block bootstrap: resampling

Fig. 1. Examples of resampling and computing CIs with two bootstrapping methods.

95% confidence level (CL). Let S be the set of execution times obtained from a performance test with n repeated executions. That is, $|S| = n$. Let θ be the 90%ile execution time calculated from S. A resample of S, denoted by S^*, is constructed by randomly selecting execution times from S. Figure 1a illustrates the construction of S^* [29, 30]. Each selection randomly picks one execution time from S. This selection is then repeated for n times to select n execution times from S with replacement. These n execution times constitute S^*.

The resampling is repeated for c times to generate c resamples, denoted by $S_1^*, S_2^*, \ldots, S_c^*$. Usually, c should be larger than 1000 for bootstrapping to work properly [29, 30]. For each resample, its 90%ile can be calculated, providing c 90%ile execution times. Let these 90%iles be $\theta_1, \theta_2, \ldots, \theta_c$. In bootstrapping, the empirical distribution constructed from these 90%iles is considered as a close approximation of the distribution of θ (the 90%ile of S). Therefore, the center 95% area of this distribution is then the CI of θ. More specifically, $\theta_1, \theta_2, \ldots, \theta_c$ are first sorted, and the 2.5%ile and 97.5%ile of the sorted list are the lower and upper bounds of the CI.

Intuitively, bootstrapping works if the resampling on S closely resembles how S is obtained from the real population. S is essentially a random sample of the real population. The resampling on S, in turn, views S as the population. The resamples, $S_1^*, S_2^*, \ldots, S_c^*$, are then the random samples of S. If the resampling process closely resembles how S is sampled from the population, then the variation of the resamples also closely resembles the variation when sampling the real population. Therefore, the distribution of the resamples can then be used to empirically calculate the CI for statistics estimates of the real population. However, as we will show later, the resampling process of basic bootstrapping does not resemble how cloud performance data are obtained from the real population, and hence, cannot always provide reliable CIs.

Block Bootstrapping. Block bootstrapping is mainly used to bootstrap time series, where the data have internal dependencies and/or seasonality [23]. Although performance test data are not strictly time series, they still contain internal dependencies. Therefore, after we discovered that the basic bootstrapping could not preserve the internal data dependencies during resampling, we started to experiment with block bootstrapping, which considerably outperformed the basic bootstrapping. Therefore, to provide a more comprehensive analysis, block bootstrapping is also introduced here.

In block bootstrapping, a resample, S^*, is also constructed from S using random selections. However, for each selection, instead of just selecting one data point (e.g., one execution time), a block of continuous data points is selected. By selecting a block of data points, the internal dependency within the data points is preserved. If there are b data points in a block, then $\frac{n}{b}$ random selections will be performed to obtain S^*. Figure 1b illustrates the procedure of block bootstrapping. Similarly to basic bootstrapping, c resamples are constructed, and the CI was computed using these resamples.

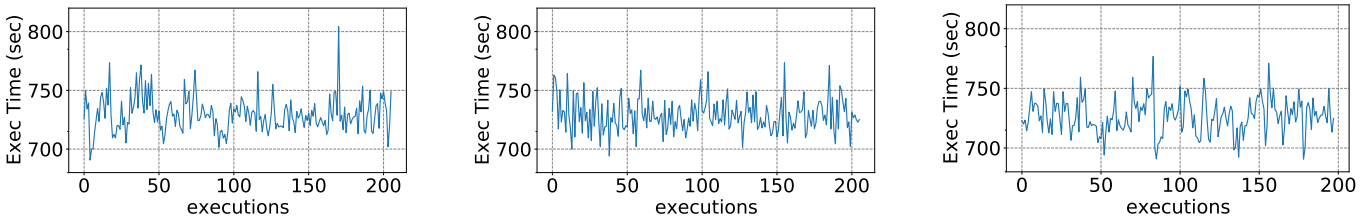
Clearly, the size of the block (i.e., the number of selected data points) is an important parameter. If the block is too large or too small, then the internal dependency may be incorrectly resampled. For this analysis, we used a fixed block size of 24. However, in *Metior*, the block size is automatically adjusted for each cloud application and cloud platform.

B. Cloud Performance Data Internal Dependency and Bootstrapping

In this analysis, we used the performance test data provided by the PT4Cloud data sets [13]. More specifically, the performance data of two benchmarks, the *ft* from NAS Parallel Benchmark Suite (NPB) [31] and *InMemory Analytics* (*IMA*) from the Cloud Suite [32], on two public clouds, Chameleon (CHM) and Amazon Web Service (AWS), are analyzed. For each benchmark, its one-week continuous execution performance data are used here. For Chameleon, the *Large* VM data are used. For AWS, the *m5.2xlarge* VM data are used. More details about these data sets are provided in Section IV.

1) Internal Dependence of Cloud Performance Test Data: Figure 2 gives the trace of the execution times of *IMA* when it was executed in AWS. Due to space limitation, we cannot show the execution time traces for *ft* and Chameleon, although the same conclusion can be reached with them. As Figure 2 shows, the execution times of *IMA* had considerable fluctuation, and these execution times showed some degree of internal dependency and repeated patterns. In particular, continuous executions had similar execution times.

In addition to the visual illustration, we also evaluated the internal dependency of these performance test data quantitatively using *Autocorrelation* (*ACF*) [24]. ACF computes the internal Pearson Correlation Coefficient (PCC) of a data set – it computes the PCC between the original data set and a derived data set obtained by shifting the data points in the original data set by 1. Hence, ACF effectively evaluates the internal dependency between two consecutive data points in



(a) Original performance test data (ACF=0.44) (b) A basic bootstrapped resample (ACF=0) (c) A block bootstrapped resample (ACF=0.47)

Fig. 2. Traces of the execution times from a one-week performance test for *IMA-AWS*, including the original performance test data and two bootstrapped resamples. Traces are down-sampled to 200 for better visibility.

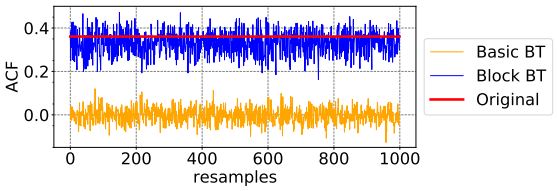


Fig. 3. Traces of the ACFs of 1000 resamples generated by two bootstrapping methods for IMA-AWS, along with the ACF of the original testing data.

TABLE I
ACF FOR EACH BENCHMARK, AND THE AVERAGE ACF OF THE RESAMPLES GENERATED BY TWO BOOTSTRAPPING (BT) METHODS.

Benchmark	Original	Basic BT	Block BT
ft - CHM	0.24	0	0.23
ft - AWS	0.16	0	0.15
IMA - CHM	0.44	0	0.41
IMA - AWS	0.36	0	0.33

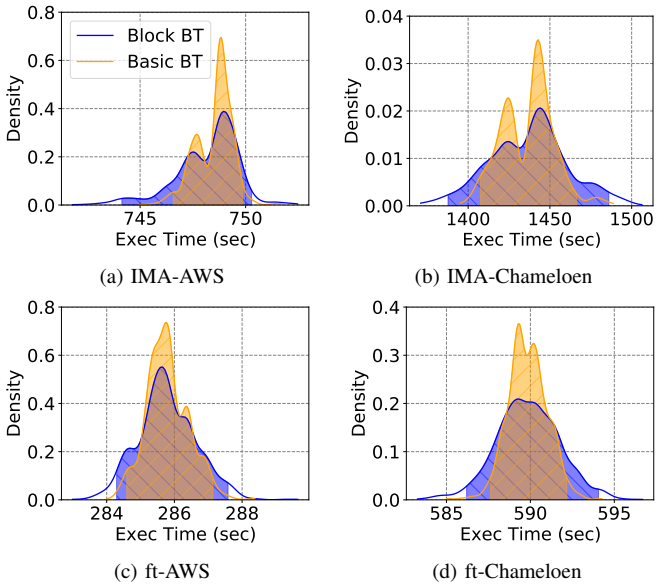
the series. Table I gives the ACF of the two benchmarks on AWS and Chameleon. As Table I shows, the ACFs for these benchmarks are above 0.1, indicating the existence of internal data dependency [33].[1]

Prior studies on cloud performance also observed these internal dependencies [13, 15]. The performance fluctuation in the cloud is mainly caused by the hardware resource contention between simultaneously running VMs [34]. As the set of VMs that are running simultaneously usually do not change rapidly, the performance of a cloud application usually also varies little within a short period, which explains the existence of the internal data dependency.

2) Bootstrapping and Internal Data Dependency: To illustrate the impact of the internal data dependency on bootstrapping, we applied the basic and block bootstrapping to the four performance test data sets of *ft* and *IMA*. For each bootstrapping, 1000 resamples were generated (i.e., c is 1000), following the standard practice [29, 30].

Figure 2 also shows the traces of two resamples using the basic and block bootstrapping for *IMA* on AWS. As Figure 2 shows, the basic bootstrapping resample is more random than the block bootstrapping resample. Figure 3 gives the ACFs of these bootstrap resamples for *IMA* on AWS, which reveals the main issue of the basic bootstrapping – its resamples

[1]Note that, ACF and PCC evaluate the existence of linear correlation. Because the internal dependency of cloud performance data is unlikely strongly linear, the ACFs for cloud performance data are usually less than 0.5. Here, we use ACF mainly to show the existence of internal dependency and show that block bootstrapping preserves the same level of internal dependency.



(a) IMA-AWS (b) IMA-Chameleon

(c) ft-AWS (d) ft-Chameleon

Fig. 4. The distributions and CIs for 90%ile execution times generated by two bootstrapping methods. Shaded areas show the ranges covered by the CIs.

usually had very low ACF. For the majority of the basic bootstrap resamples, the ACF was nearly 0, indicating that these resamples nearly had no internal data dependencies. However, the ACFs of the resamples from block bootstrapping were usually close to the original performance data.

Table I also reports the average ACF of the two bootstrapping methods for all four performance data sets, where the basic bootstrapping has average ACFs of nearly 0. However, block bootstrapping has average ACFs very close to the original testing data, indicating that block bootstrapping can indeed preserve a similar level of internal dependency.

Fundamentally, the resampling of the basic bootstrapping is different than how the original performance data are collected. In basic bootstrapping, the resampling assumes each execution is independent. However, for the original test, consecutive executions had similar execution environments (e.g., similar co-running VMs). Hence, continuous executions are not independent, unlike assumed in the basic bootstrapping. However, the resampling in the block bootstrapping assumes consecutive executions are correlated, and thus has a better resemblance to the original performance test.

This difference in resample ACFs, in turn, leads to the

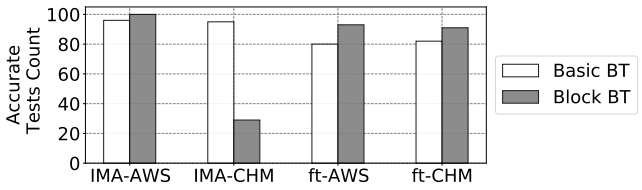


Fig. 5. Numbers of performance tests that produced results (90%ile exec. time) with less than 3% errors.

differences of the CIs generated based on these resamples. Figure 4 shows the CIs of the 90%ile execution time for each pair of benchmark and cloud. As Figure 4 shows, the CIs generated by the basic bootstrapping are usually smaller than those generated by block bootstrapping. Smaller CI indicates that basic bootstrapping estimated that the performance results had smaller errors. However, if the estimated error is smaller than the actual error, the users would incorrectly assume their performance results are accurate and stop their tests too early.

At last, we conducted 100 performance tests using the basic and block bootstrapping to test for the 90%ile execution times of the above benchmarks. For these tests, the maximum allowed error was set to be 3%. Therefore, the tests were stopped when the CIs were smaller than $(-3\%, +3\%)$. After the tests, the accuracy of the 90%ile execution times was compared with the ground truth 90%ile execution times, where were calculated based on another 5-week data for each benchmark from the PT4Cloud data set. Figure 5 gives the number of accurate tests (i.e., had an error less than 3%). As Figure 5 shows, for *ft* on both clouds and *IMA* on AWS, block bootstrapping indeed increased the number of accurate tests. Therefore, block bootstrapping should be used in cloud performance testing than the basic bootstrapping.

For *IMA* on Chameleon, however, the testing result accuracy was worse with block bootstrapping. The worse results were due to two issues. First, the CIs were generated based on a fixed set of testing data, and thus, may produce inaccurate results if the test data do not cover all potential performance fluctuations [13]. Second, we used a fixed block size (24) in these tests. However, the proper block size depends on the behavior of the cloud application and the cloud. This result shows that block bootstrapping cannot be blindly applied to cloud performance testing without addressing these two issues.

III. The Design of *Metior*

This section presents our cloud performance testing methodology, *Metior*, which addresses the aforementioned two issues of block bootstrapping. It also employs a periodical and intermittent AUT execution strategy to reduce testing cost.

To address the incomplete test coverage issue, the stop criterion used by *Metior* employs the "law of large numbers," with states that the experimental mean should be close to the true mean when the number of trials is large and tend to become closer to the true mean as the number of trials increases [35]. For cloud performance testing, this intuition may be rephrased as: if the mean from a large number of executions in the cloud covers all potential fluctuations and is close to the true mean, then adding a substantial number of more executions should not significantly change the value of the mean. Based on this intuition, the stop criterion of *Metior* stops a test when the performance result obtained from the test stays unchanged after adding significantly more executions. Note that, although the "law of large numbers" is only about the mean, this intuition can also be applied to percentiles of the performance, as shown in Section IV.

To address the issue where the block size varies with cloud application and cloud platform, we employed a methodology developed by Politis and White to automatically select the block size [23]. Intuitively, this automatic selection finds the "optimal" block size using the minimum block size that provides a non-negligible autocorrelation [23]. By adopting the automatic selection technique, the "optimal" block size allows block bootstrapping to retain similar level of internal data dependency as the original sample. In *Metior*, for each bootstrapping, this automatic block size selection is performed so that each performance test uses its own block size.

Metior is implemented as a fully automated performance tester. To apply *Metior*, its user only need to provide the VM configuration, a cloud application (i.e., AUT) and its input data, a maximum allowed error, and a confidence level. *Metior* automatically allocates VMs, conducts tests, and applies bootstrapping, using a cloud service's programming interface. Currently, *Metior* support cloud services including AWS, Google Cloud, and Chameleon. Note that, *Metior* is not designed to generate or prioritizing inputs. Instead, it is designed to provide accurate cloud performance testing results for any inputs.

A. *Overview of* Metior

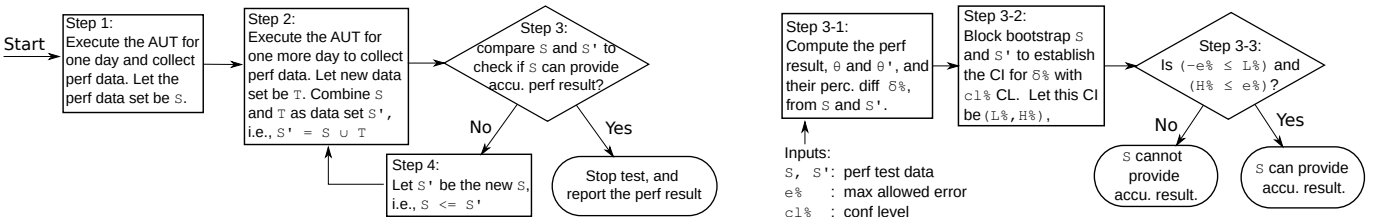
Figure 6a gives the overall workflow of *Metior*. In Step 1, *Metior* executes the AUT repeatedly for one day. Let the set of performance data collected from these executions be S. In Step 2, *Metior* executes the AUT repeatedly for another day to obtain a new data set T. T is then combined with S to obtain S', i.e., $S' = S \cup T$. Note that, Section III-C provides the rationale for conducting the tests in terms of days.

In Step 3, *Metior* compares S and S' to determine if there is a significant change in the performance results obtained from S to S'. This comparison follows the intuition of the "law of large numbers" – if the extra data in S' does not significantly change the performance results, then the performance result from S is deemed accurate, and the test can be stopped.

However, if the change from S to S' is significant, *Metior* deems that more executions are required. Hence, in Step 4, *Metior* let S' become the S. It then goes back to Step 2 to conduct more executions and collect more performance data to generate a new S'. With the new S and S', a new comparison is performed at Step 3 to determine if the test can be stopped.

B. *The Stop Criterion of* Metior

Metior's stop criterion uses block bootstrapping to determine if there is a significant change in the performance results obtained from S and S'. Similar to existing cloud performance testing methodologies, the user of *Metior* needs to select a



(a) Workflow of *Metior*. (b) *Metior*'s stop criterion.

Fig. 6. Overall workflow of using *Metior* to conduct a performance test.

maximum allowed error, denoted by $e\%$, and a confidence level, denoted by $cl\%$. *Metior* uses block bootstrapping to determine if the performance results obtained from S and S' have a maximum possible change (difference) larger than $e\%$ with $cl\%$ confidence. If the maximum possible change (difference) is less than $e\%$, then *Metior* deems that S can provide a performance result with less than $e\%$ error under $cl\%$ confidence. This stop criterion is based on the following intuition – if both S and S' can provide accurate performance results with less than $e\%$ error, then the maximum difference between the performance results obtained from S and S' should usually be smaller than $e\%$.

Figure 6b gives the steps performed by *Metior*'s stop criterion. These steps correspond to the internal operations of Step 3 in Figure 6a. The first step of this stop criterion (Step 3-1 in Figure 6b) is to calculate the performance results, θ and θ', from S and S'. Let the percentage difference of θ and θ' be $\delta\%$ ($\delta\% = \frac{\theta'-\theta}{\theta}$). In Step 3-2, *Metior* calculates the CI of $\delta\%$ with $cl\%$ confidence. This CI represents the maximum possible difference between the performance results obtained from S and S' with $cl\%$ possibility. This CI is calculated using the comparison block bootstrapping with the following procedure [30]. First, two resamples, S^* and S'^*, are resampled from S and S' using block bootstrapping. As stated above, the block sizes are automatically selected based on the data of S and S'. Second, the percentage difference $\delta^*\%$ between the performance results (mean or percentile) of S^* and S'^* is computed. Third, the above resampling is performed 1000 times, providing 1000 differences, i.e., $\delta_1^*\%, \delta_2^*\%, \ldots, \delta_{1000}^*\%$. Fourth, the 1000 $\delta^*\%$s are sorted, and the center $cl\%$ of the sorted list then gives the CI of $\delta\%$ with $cl\%$ confidence.

Let the CI of $\delta\%$ be $(L\%, H\%)$. In Step 3-3, *Metior* checks if the conditions, $-e\% < L\%$ and $H\% < e\%$, are true. If both conditions are true, then the maximum possible difference between the performance results of S and S' is less than $e\%$, and the performance result of S is considered to be accurate by *Metior*. Hence, the test can be stopped. Otherwise, the test continues to Step 4 in Figure 6a.

C. Low-cost Test Execution in Metior

Because cloud performance fluctuates over time, we choose to conduct the performance test in small intervals/periods of executions rather than a specific number of executions. That is, in Figure 6a, *Metior* executes the AUT for one day in Step 1 and 2. Moreover, our analysis on bootstrapping in Section II shows that continuous executions usually have similar performance. Therefore, there is no need to continuously execute the AUT. Instead, the AUT can be executed intermittently while still providing good coverage of performance fluctuations. As cloud usage cost is charged in terms of seconds or minutes, intermittently execution can reduce the number of executions and the cloud usage expenditure.

Nevertheless, the interval length and intermittent frequency must be carefully chosen to obtain accurate performance results. Prior work employed a similar periodical and intermittent execution strategy [13]. However, the prior work did not provide a systematic approach to determine the interval length and intermittent frequency. Here, we employed a data-driven approach. To determine the interval length, we evaluated the "optimal" block sizes for the performance tests data of *ft* and *IMA* in Section II using the aforementioned automatic block size selection technique. The largest block size we found was 24 hours of executions. Therefore, to ensure the execution count in Step 2 is indeed significantly large, we set the interval length to be one day so that Step 2 can provide at least one new block of data. To determine the intermittent frequency, we again used the performance test data from Section II to determine how many executions can be discarded from the test data without significantly reducing the autocorrelation (ACF). We discovered that it needed at least four executions per hour to maintain an ACF similar (less than 0.1 smaller) to the original data. We used 0.1 as the threshold because less-than-0.1 ACF is considered as no correlation [33].

In summary, based on the above analysis, in Step 1 and 2 of *Metior*, we choose to execute the AUT intermittently 4 times per hour and repeat hourly for one day.

IV. Experimental Evaluation

This section describes the experimental evaluation of *Metior*, which answers the following two research questions: 1) What is the accuracy of the performance results obtained *Metior*? 2) What is the cost of applying *Metior*?

A. Experiment Setup

Benchmarks and Clouds. For this evaluation, we used the performance data sets from PT4Cloud [13]. PT4Cloud data sets provide the performance data of executing six benchmarks on two public clouds, AWS and Chameleon, continuously for eight (8) weeks. Table II gives the details of these six benchmarks, and Table III gives the types and counts of the six

TABLE II
BENCHMARKS USED IN THE EVALUATION.

Benchmark	Domain	Type of Perf.	Origin
ft	HPC	execution time	NPB [31]
ep	HPC	execution time	NPB [31]
JPetStore (JPS)	Web	throughput	J2EE [36]
YCSB	DB	throughput	YCSB [37]
TPC-C	DB	throughput	OLTPBench [38]
InMem Analy. (IMA)	ML	execution time	CloudSuite [32]

TABLE III
VM CONFIGURATIONS FROM CHAMELEON (C) AND AWS (A) USED IN THE EVALUATION.

Config.	VM Cnt x VM Type	Cores / VM	Mem / VM
C-S	4 x Small	1	2GB
C-M	2 x Medium	2	4GB
C-L	1 x Large	4	8GB
A-S	4 x m5.large	2	8GB
A-M	2 x m5.xlarge	4	16GB
A-L	1 x m5.2xlarge	8	32GB

Virtual Machine (VM) configurations used in PT4Cloud. Each benchmark was executed on all six VM configurations, except for benchmarks *ft*, *ep*, and *IMA*, which could not be executed on the small VMs of Chameleon (i.e., *C-Sm* in Table III) due to inefficient memory. Here, a pair of benchmark and VM configuration is called a *benchmark configuration*. In total, all 33 benchmark configurations from PT4Cloud were evaluated.

Evaluation Methodology. We partitioned the 8-week performance data for each benchmark configuration into two parts. The first part contained 3-week of data and was used to conduct performance tests, whereas the rest 5-week data were used as ground truth. The 3-week data have a large number of performance data points, with each data point contains the performance (execution time or throughput) of one execution. From each data point, a performance test could be conducted (simulated). For instance, starting from a data point, a performance test with *Metior* can be simulated by continuously reading in data points until *Metior*'s stop criterion deems that the performance test can be stopped. After the performance test is stopped, the data points read during the test can then be used to calculate the performance results. Because the 3-week data of each benchmark configuration contain hundreds or thousands of data points, repeating the simulated performance test starting from every data point led to hundreds or thousands of simulated performance tests for each benchmark configuration, allowing a thorough evaluation.

In this evaluation, the performance tests were used to obtain performance results as the mean and 90%ile of the execution time or throughput, reflecting the average and tail performance. The maximum allowed errors (i.e., the $e\%$ in Figure 6) were set to be 3%, 5%, and 10%. The confidence level (i.e., the $cl\%$ in Figure 6) was chosen to be 95%.

Baselines. As stated above, the last 5-week data of each benchmark configuration were used to obtain the ground truth performance. For cloud performance testing, the ground truth should be the performance results obtained from extremely long performance tests that can truly cover all performance fluctuations. Our current performance results showed that the performance results from 4 (or more) weeks of executions are usually stable enough to be used as ground truth. Besides the ground truth, we also compared *Metior* with three state-of-the-art cloud performance testing methodologies. The first methodology, BasicBT, used the basic bootstrapping [12]. The second methodology, CoV, uses the changes in the coefficient of variation of the performance data as the stop criterion [20]. The third methodology is PT4Cloud [13].

Metrics. The percentage error of each performance result obtain with *Metior*, $Perf_{Metior}$, is calculated by comparing it with the ground truth performance, $Perf_{true}$, using the following equation,

$$err = |\frac{Perf_{Metior} - Perf_{true}}{Perf_{true}}| \times 100\%. \quad (1)$$

Because large numbers of performance tests were conducted using *Metior* for each benchmark configuration, we report the percentage of tests that provided performance results with less than the maximum allowed error. For example, if the maximum allowed error is 3%, then we report the percentage of the performance tests that indeed provided performance results with less than 3% error. Moreover, recall that, in this paper, we call a performance test result as *accurate* if it indeed has an error less than the maximum allowed error.

Open Data. Our data and source code are available at https://doi.org/10.5281/zenodo.5093934.

B. Accuracy Evaluation with 3% Maximum Allowed Error

1) Accuracy of Metior*:* Figure 7 gives the percentage of the *Metior* performance tests that provided accurate mean performance (i.e., with less than 3% error). As Figure 7 shows, 100% of *Metior*'s tests conducted on AWS provided mean performance with less than 3% error. On Chameleon, except for benchmark configurations of *JPS-C-S* and *TPCC-C-M*, *Metior* ensured more than 90% of the tests were accurate for all benchmark configurations. The benchmark performance on Chameleon had larger fluctuations than AWS, making it more difficult to obtain accurate results. Nonetheless, even for *JPS-C-S* and *TPCC-C-M*, the tests could still provide performance results with low error. For *JPS-C-S*, the largest error among all the tests conducted for it was only 4.2%.[2] For *TPCC-C-M*, the largest error was 5.8%. Figure 9a gives the average percentages of *Metior*'s tests for mean performance that were accurate. Overall, 96% of the tests on Chameleon and 100% of AWS tests had less than 3% errors. 98% of all tests on two clouds had less than 3% errors.

Figure 8 gives the percentage of the *Metior* performance tests that provided accurate 90%ile performance (i.e., with less than 3% error). Again, 100% of *Metior*'s tests conducted on AWS provided 90%ile performance with less than 3% errors. On Chameleon, except for the configurations of *YCSB-C-S*

[2]Because hundreds or thousands of tests were conducted for each benchmark configuration, it is impossible to provide the error for each test in Figure 7 and Figure 8. Due to space limitation, we also cannot provide the max and average error for tests of each benchmark configuration.

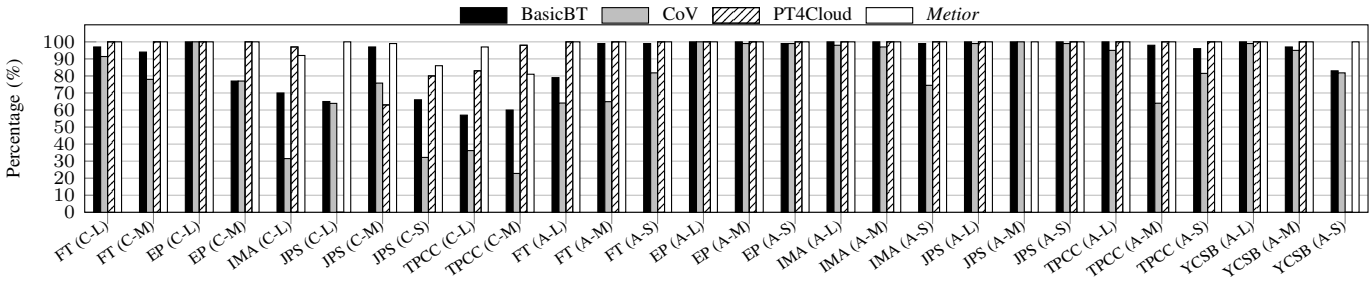


Fig. 7. Percentages of the tests that provided mean performance results with less-than-3% errors.

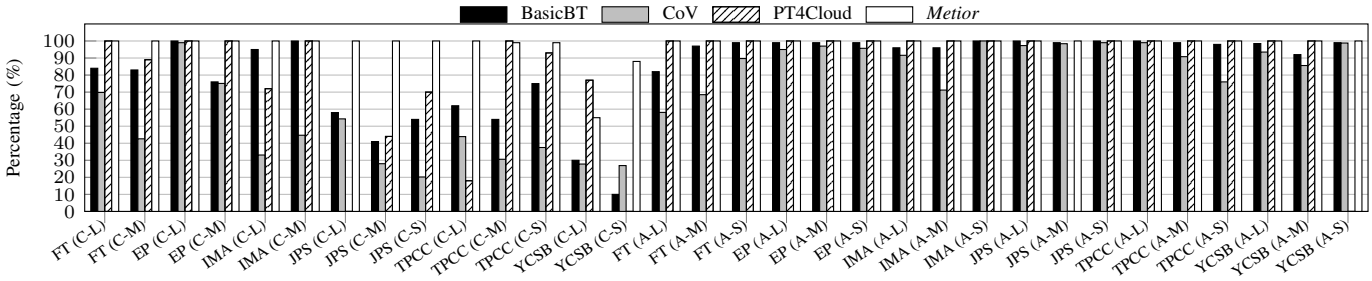
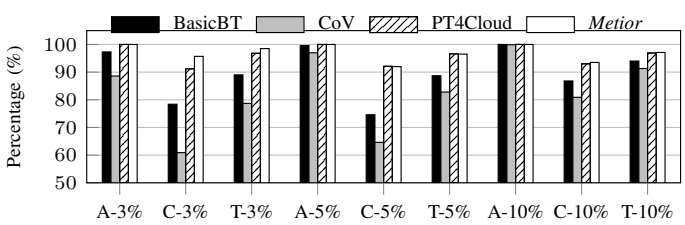
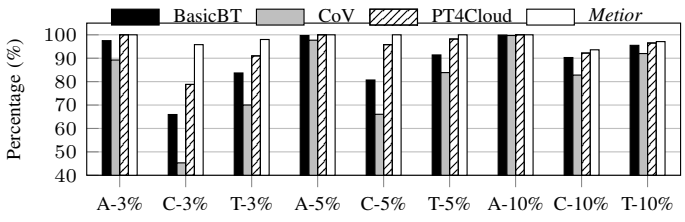


Fig. 8. Percentages of the tests that provided 90%ile performance results with less-than-3% errors.



(a) Tests for mean performance.



(b) Tests for 90%ile performance.

Fig. 9. Summary of the percentages of the performance tests that were accurate. Results are reported for three (3%, 5%, and 10%) maximum allowed errors on AWS (A), Chameleon (C) and all tests (T).

and *YCSB-C-L*, *Metior* ensured more than 90% of the tests were accurate for all benchmark configurations. Similar to the mean performance, *Metior*'s performance results still had low errors for *YCSB-C-S* and *YCSB-C-L* – the maximum errors of the tests conducted for *YCSB-C-S* and *YCSB-C-L* were only 3.6% and 4.6%, respectively.[2] Figure 9b gives the average percentages of *Metior*'s tests for 90%ile performance that were accurate. Overall, 96% of the Chameleon tests and 100% of AWS tests had less than 3% errors. 98% of all tests on two clouds had less than 3% errors.

Note that, for mean performance results, the tests for five benchmark configurations could not stop with just 3-week of data. For 90%ile performance results, the performance tests for *YCSB-C-M* could not stop with 3-week of data. These benchmark configurations are not shown in Figure 7 and Figure 8, but are discussed in the next section.

2) Unstoppable Benchmark Configurations: For the mean performance, for five benchmark configurations, *IMA-C-M*, *TPCC-C-S*, *YCSB-C-L*, *YCSB-C-M*, and *YCSB-C-S*, *Metior* deemed that all their performance tests should not stop (i.e., could not provide accurate results) with just 3-week data. For the 90%ile performance, *Metior* deemed that all the tests of *YCBS-C-M* should not stop with just 3-week data. Therefore, the results of these benchmark configurations are not shown in Figure 7 and Figure 8. Note that, as PT4Cloud only provided 8 weeks of performance data, it was also impossible for us to continue the tests without running into ground truth data. Therefore, we terminated the tests once all 3-week data were used and report these tests as unstoppable.

For three benchmark configurations, *IMA-C-M*, *TPCC-C-S*, and *YCSB-C-M*, 3-week data indeed could not provide mean or 90%ile performance with less than 3% error. For *YCSB-C-L* and *YCSB-C-S*, although 3-week data could provide accurate performance results, their performance data had high variations. These high variations resulted in wide CIs for their mean or 90%ile performance, making it impossible to conclude that the performance results from 3-week data were accurate with 95% confidence. Therefore, we concluded that *Metior* indeed should not stop the tests and behaved as expected for these benchmark configurations. These unstoppable tests also reflect the difficulty of conducting cloud performance testing under large performance fluctuation and show the importance of exploring reliable cloud performance testing methodologies.

3) Comparison with PT4Cloud: The percentages of the accurate tests for PT4Cloud are also given in Figure 7 and Figure 8. As the figures show, except for the above four configurations, *Metior* had more accurate tests than PT4Cloud for all other benchmark configurations. When testing for the mean of *IMA-C-L* and 90%ile of *TPCC-C-M*, PT4Cloud was slightly better than *Metior* with less than 5% more accurate tests. When testing for the mean of *TPCC-C-M* and 90%ile of *YCSB-C-L*, P4Cloud had 17% and 22% more accurate tests than *Metior*. Nonetheless, when testing for the mean of *TPCC-C-M*, the average and maximum errors of the tests conducted by PT4Cloud and *Metior* were similar – the average errors for PT4cloud and *Metior* were 1% and 2%, and the maximum errors for PT4Cloud and *Metior* were 3.8% and 5.8%.[2] For the 90%ile tests of *YCSB-C-L*, the average errors of PT4cloud and *Metior* were both 3%, whereas PT4Cloud maximum error (8.6%) was higher than *Metior* (4.6%). *Metior*'s similar or even better average or maximum errors show that although *Metior* had fewer accurate tests for these two configurations, the accuracy of individual tests of *Metior* was still similar to those of PT4Cloud, and *Metior*'s tests also had errors very close to the 3% maximum desired error. Moreover, *Metior* was more accurate for all other benchmark configurations.

Figure 9 also gives the overall accuracy of PT4Cloud, which shows that PT4Cloud's overall accuracy was similar or worse than *Metior*. Especially when testing for the 90%ile on Chameleon with 3% max error, *Metior* has 17% more accurate tests than PT4Cloud. PT4Cloud stopped the test with the anticipation that the distribution (i.e., most of the percentiles) had less than 3% error. However, it did not imply that the mean or every percentile also had less than 3% error. Therefore, the errors for some point estimates may still be larger than 3%, causing P4Cloud to have lower accuracy than *Metior*.

Note that, for several benchmark configurations, PT4Cloud could not stop the tests with 3-week of data (in addition to those discussed in Section IV-B2). The PT4Cloud bars of these configurations are omitted in Figures 7 and 8.

4) Comparison with BasicBT method: Figures 7 and 8 also show the percentages of tests that had less than 3% errors for BasicBT [12]. As both figures show, BasicBT had lower or similar accuracy than *Metior* for every benchmark configuration. In several cases, BasicBT was significantly less accurate than *Metior*, such as JPS-C-L and TPCC-C-L.

Figure 9 gives the overall accuracy of BasicBT. On AWS, BasicBT had performed reasonably well with 97% of all BasicBT's tests had less than 3% errors. Nonetheless, it was still lower than the 100% of *Metior*. Moreover, the large performance fluctuation on Chameleon had made it particularly difficult for BasicBT to stay accurate. On Chameleon, 78% of BasicBT's tests for mean and 66% of BasicBT's tests for 90%ile had less than 3% errors, whereas 96% of *Metior*'s tests on either mean or 90%ile were accurate. As analyzed in Section II, BasicBT has two issues of incomplete performance fluctuation coverage and not considering internal data dependency, causing the relatively low accuracy.

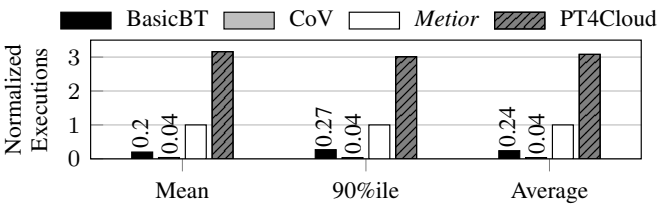


Fig. 10. The number of executions conducted by evaluated testing methodologies, normalized to *Metior*.

5) Comparison with the CoV method: Figures 7 and 8 also show the percentages of tests that had less than 3% errors for the CoV method [20]. Overall, CoV had the lowest accuracy among the three cloud performance testing methodologies. On AWS, 89% of CoV's tests could provide accurate means or 90%iles. On Chameleon, 61% of CoV's tests could provide accurate means, and 45% of its tests could provide accurate 90%ile performance. The CoV method was not designed to obtain accurate performance results with a maximum desired error. Instead, it was designed to detect if the performance results from microbenchmarks were stable enough. Hence, because of having a different design goal, CoV's accuracy was lower than the other methodologies in this evaluation.

C. Testing Cost for 3% Max Allowed Errors

To obtain the mean performance, *Metior* conducted 290 executions per test on average. For the 90%ile performance, *Metior* conducted 306.2 executions per test on average. These executions roughly translate into three days of execution per test. As we used existing data sets, we do not have the AWS bills for these tests. Therefore, we estimated the testing cost based on the test execution time and AWS's cloud usage rates (Chameleon is a free research cloud without charges). The estimation shows that *Metior*'s cost for one test on AWS ranged from $1.2 to $57.6, with an average cost of $17.6.

Figure 10 compares the average number of executions per test required by the evaluated performance testing methodologies. As Figure 10 shows, the number of executions required by PT4Cloud was 3.1 times of those used by *Metior* on average. This high number of executions was because PT4Cloud was designed to obtain performance distributions. When a whole distribution is deemed accurate (i.e., with less than 3% error) by PT4Cloud, most of the percentiles of the performance distribution had about 3% error, which required many more tests than just obtaining one accurate mean or percentile. Consequently, PT4Cloud incurs unnecessary costs for performance tests that only need to obtain point estimates, and its cost may be prohibitively high when a cloud application has a large number of inputs that need to be tested. Note that, *Metior*'s reduction in execution count does not only imply less monetary cost, it also indicates a reduction in temporal cost, and hence, a faster development/deployment cycle.

BasicBT and CoV required fewer executions than *Metior*, as shown in Figure 10. Specifically, the average execution counts per test for BasicBT were 20% to 27% of *Metior*'s execution counts, and CoV's execution counts were about 4%

of *Metior*. However, as BasicBT and CoV had lower accuracy than *Metior*, these fewer executions did not indicate a cost reduction, but indicated that BasicBT and CoV stopped their tests too early before obtaining accurate performance results.

D. Sensitivity to Maximum Allowed Errors

We also evaluated *Metior* with 5% and 10% maximum allowed errors. The evaluation results are summarized in Figure 9. As Figure 9 shows, *Metior*'s retained its accuracy when the maximum allowed errors were increased. On AWS, 100% of *Metior*'s tests were still accurate. On Chameleon, more than 92% of *Metior*'s tests were accurate when the maximum error was 5%, and more than 94% of *Metior*'s tests were accurate when the maximum error was 10%.

The accuracy of PT4Cloud, BasicBT, and CoV was also improved under 5% and 10% maximum errors. However, PT4Cloud still had lower overall accuracy than *Metior*. Moreover, the numbers of executions per test of PT4Cloud were still 3.2 and 3.6 times more than *Metior* for 5% than 10% maximum errors. BasicBT still struggled on Chameleon and had considerably lower accuracy than *Metior* even with larger maximum errors. The accuracy of CoV was even lower than BasicBT, as it is not designed to obtain performance with specified maximum errors.

V. Case Study and Application of *Metior*

A Case Study. To demonstrate the usefulness of *Metior* to cloud researchers and practitioners, we applied *Metior* to a cloud optimization technique, CherryPick [6], which selects the best VM configuration for a cloud application by predicting its performance on different VM configurations. CherryPick employs Gaussian Process (GP) to make the prediction [39], which requires a training data set which consists of the performance of the application running in other VMs. Here, we applied *Metior* to obtain accurate performance results as the training sets to show that *Metior* can improve CherryPick's prediction accuracy.

More specifically, CherryPick was used to predict the performance of the six benchmarks in Table II when they were executing on the AWS 2×m5.xlarge configuration (i.e., *A-M*). The training data sets were composed of performance data points for five VM configurations, including 4×m5.large (*A-S*), 1×m5.2xlarge (*A-L*), 1×m5.large, 2×m5.large and 1×m5.xlarge. The original CherryPick techniques asked for 6 data points for each VM configuration in the training set. However, for the *Metior*-enhanced CherryPick, the data points for each VM configuration must be many enough to provide an accurate mean using the *Metior* methodology. That is, *Metior* was used to obtain accurate mean performances for each VM configuration, then the acquired mean performances of each VM configuration were used as training data sets for CherryPick to make predictions. New tests were conducted for the VM configurations (i.e., 1×m5.large, 2×m5.large and 1×m5.xlarge) that are not included in the PT4Cloud data sets. For a thorough evaluation, 10000 performance predictions were made using the original and *Metior*-enhanced Cherrypick. Then for each performance prediction, its percentage error was computed by comparing the prediction with the ground truth performance, which was the same five-week performance used in the previous evaluation. The mean absolute percentage errors (MAPE) are then reported in Figure 11. Note that, this case study was only conducted on AWS, as Chameleon only offered three VM configurations (CherryPick requires more than 3 configurations).

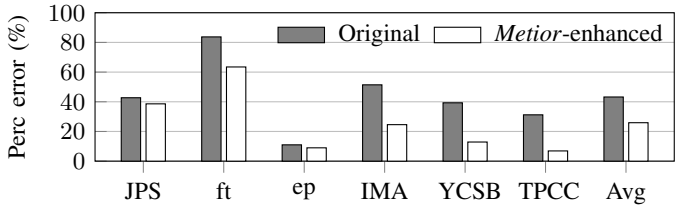


Fig. 11. Average prediction errors for the original CherryPick and *Metior*-enhanced CherryPick.

As Figure 11 shows, *Metior*-enhanced CherryPick had higher accuracy than the original CherryPick for every benchmark. On average, *Metior*-enhanced CherryPick's prediction error was 17.3% less than the original CherryPick. Moreover, *Metior*-enhanced CherryPick had more stable predictions (i.e., less variation in prediction accuracy). The main benefit of *Metior* is that it can provide more accurate training data sets, which contained many more samples than the original CherryPick. Note that, simply adding more training data do not always increase CherryPick's accuracy, because without a reliable performance testing methodology, it is unclear how many more training data should be added to ensure good accuracy. The authors of CherryPick also noted that GP was considered as just accurate enough to separate fast VM configurations from slower ones. Nonetheless, a later study showed that a GP model with better accuracy could improve the accuracy of identifying better-performing VMs [40]. The higher accuracy could improve CherryPick's ability at VM configuration optimization [40]. These results illustrate that, by providing more reliable training sets and accurate performance results, *Metior* is valuable for both cloud research and practice.

Application of *Metior*. The necessity of accurate cloud performance results was documented by prior cloud performance test studies [9, 12, 13, 20]. In cloud deployments, a key step is to select the proper VM configuration that meets the performance requirement [1, 6, 41]. The selection of auto-scaling policies also requires determining a proper VM configuration that meets the performance requirement as the scaling target [5, 42, 43]. The most reliable way to determine if a VM configuration meets a performance requirement is performance testing. For cloud research, obtaining accurate performance results is also the fundamental requirement. Accurate performance results are required to evaluate new cloud system/application designs [44] and develop new optimization techniques (as shown with the case study).

VI. Threats to Validity

Execution Environment Changes. *Metior* assumes that the execution environments, including the cloud hardware infrastructure and the statistical behavior of multi-tenancy, remain unchanged during the performance test and after the deployment. In our experience, the hardware infrastructure at Chameleon and AWS remained unchanged for years, and the multi-tenancy behaviors were also consistent within at least year. Therefore, performance tests conducted with *Metior* with only a few days or weeks of executions can accurately provide the performance of cloud deployments within at least one year. However, if the execution environment changes, new performance tests should be conducted.

Nonetheless, the cloud execution environment does experience long-term (e.g., after multiple years) changes. Therefore, new performance testing methodologies are required to tackle this long-term performance change. We plan to redesign *Metior* into cloud APIs to allow it to detect performance variations caused by execution environment changes.

Other Cloud Applications, Test Inputs, and Cloud Service providers. Although we strive to provide a comprehensive evaluation, the exact accuracy of *Metior* may change with the cloud applications, performance test inputs, and the cloud service providers. Nonetheless, we expect *Metior*'s behavior to be generally consistent over a variety of cloud applications, inputs, and cloud services. Moreover, *Metior* is used to obtain the accurate performance of a cloud application given one test input. *Metior* does not aim at determining what test inputs should be included in the performance tests.

VII. Related Work

Cloud Performance Testing. Performance testing is a fundamental task in computer science [10, 11]. Jain documented the methodology of using CI for performance measurement in detail [19]. Maricq et al. recently improved this methodology to cloud computing by employing the basic bootstrapping [12]. Wang et al. also employed basic bootstrapping in cloud performance testing [9]. However, as shown in this paper, the basic bootstrapping based testing methods had lower accuracy partially due to overlooking internal data dependency. PT4Cloud was a performance testing technique for obtaining performance distributions of cloud applications [13]. Our work is inspired by PT4Cloud, especially in the use of periodical and intermittent executions. However, PT4Cloud determined the parameters of interval length and intermittent frequency empirically, whereas *Metior* employed a systematic approach to select these parameters. Moreover, as shown in Section IV-B3, PT4Cloud had higher errors and higher costs than *Metior*. Laaber et al. proposed a stop criterion for executing microbenchmark in the cloud using the coefficient of variation [20]. However, as shown in Section IV-B3, this stop criteria had lower accuracy, as it was not designed to obtain performance results with a maximum allowed error. Alghmadi et al. proposed a stop condition for performance testing by determining the repetitions in performance data [45]. As shown in prior work, this stop condition was not suitable for testing performance in cloud computing [13]. Duet benchmarking is a technique to compare the performance of two cloud system designs or optimizations [44, 46]. While this technique provides accurate comparisons, it was not designed to determine the exact performance of a cloud application.

Other Related Work Cloud performance prediction models were built to predict a cloud application's performance on a VM configuration or cloud service to aid cloud resource allocation [1, 6, 40, 47]. These models used performance testing results as their training data. Some studies also predicted and estimated non-cloud software performance with models and simulators [48–55]. As shown with our case study (Section V), *Metior* can provide more accurate cloud performance results as reliable training and testing data sets to facilitate the development of performance modeling and simulation. There were also studies on test inputs generation and prioritization for performance testing [43, 56, 57, 57–70]. Several studies also investigated performance change identification in configurable systems [71, 72]. These studies are orthogonal to *Metior*, as *Metior* focused on providing accurate performance results for any test inputs and/or configurations.

VIII. Conclusion and Future Work

This paper addressed the cloud performance testing problem. We first conducted an analysis to show that the basic bootstrapping could not always provide accurate performance results due to the overlooked internal dependency with cloud performance data. We then present *Metior*, a reliable automated performance testing methodology using block bootstrapping, which considers the internal dependency of cloud performance data. To improve performance fluctuation coverage and further reduce testing cost, *Metior* also employed the "law of large numbers" and conducted tests periodically and intermittently. Experiment results showed that *Metior* ensured that more than 98% of tests had less than 3% error. For future work, we will use *Metior* to test different types of cloud services, more cloud service providers, as well as additional types of applications. We also plan to implement *Metior* as a serverless API so that it can easily used by cloud practitioners for performance testing and for long term performance fluctuation monitoring.

Acknowledgment

This work was supported by the National Science Foundation under grants CCF-1617390, CCF-1618310, and CNS-1911012. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of NSF. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. Selecting the Best VM Across Multiple Public Clouds: A Data-driven Performance Modeling Approach. In *ACM Symp. on Cloud Computing*, 2017.

[2] Stoyan Stefanov. YSlow 2.0. In *CSDN Software Development 2.0 Conference*, 2008.

[3] Marissa Mayer. In Search of A better, faster, strong Web, 2009.

[4] Timothy Zhu, Michael A. Kozuch, and Mor Harchol-Balter. WorkloadCompactor: Reducing datacenter cost while providing tail latency SLO guarantees. In *ACM Symp. on Cloud Computing*, 2017.

[5] Ming Mao and Marty Humphrey. Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *Proc. of Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, 2011.

[6] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *USENIX Symp. on Networked Systems Design and Implementation*, 2017.

[7] F. L. Ferraris, D. Franceschelli, M. P. Gioiosa, D. Lucia, D. Ardagna, E. Di Nitto, and T. Sharif. Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds. In *Int'l Symp. on Symbolic and Numeric Algorithms for Scientific Computing*, 2012.

[8] Mark Grechanik, Qi Luo, Denys Poshyvanyk, and Adam Porter. Enhancing Rules For Cloud Resource Provisioning Via Learned Software Performance Models. In *ACM/SPEC on Int'l Conf. on Performance Engineering*, 2016.

[9] W. Wang, N. Tian, S. Huang, S. He, A. Srivastava, M. L. Soffa, and L. Pollock. Testing Cloud Applications under Cloud-Uncertainty Performance Effects. In *Int'l Conf. on Software Testing, Verification and Validation*, 2018.

[10] Yutong Zhao, Lu Xiao, Xiao Wang, Bihuan Chen, and Yang Liu. Localized or Architectural: An Empirical Study of Performance Issues Dichotomy. In *Int'l Conf. on Software Engineering: Companion Proceedings*, 2019.

[11] Andreas Burger, Heiko Koziolek, Julius Rückert, Marie Platenius-Mohr, and Gösta Stomberg. Bottleneck Identification and Performance Modeling of OPC UA Communication Models. In *Proc. of ACM/SPEC Int'l Conf. on Performance Engineering*, ICPE '19, 2019.

[12] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming Performance Variability. In *USENIX Symp. on Operating Systems Design and Implementation*, 2018.

[13] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. A Statistics-Based Performance Testing Methodology for Cloud Applications. In *Proc. of ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, 2019.

[14] Philipp Leitner and Jürgen Cito. Patterns in the Chaos: A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Trans. Internet Technol.*, 16(3):15:1–15:23, April 2016.

[15] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transcations on Parallel Distributed System*, 22(6):931–945, June 2011.

[16] Christoph Laaber, Joel Scheuner, and Philipp Leitner. Software Microbenchmarking in the Cloud. How Bad is It Really? *Empirical Software Engineering*, 24(4):2469–2508, 2019.

[17] Catia Trubiani, Pooyan Jamshidi, Jurgen Cito, Weiyi Shang, Zhen Ming Jiang, and Markus Borg. Performance Issues? Hey DevOps, Mind the Uncertainty. *IEEE Software*, 36(2):110–117, 2019.

[18] James E. II Bartlett, W. Joe Kotrlik, and Chadwick C. Higgins. Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal*, 19(1):43, 2001.

[19] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1990.

[20] Christoph Laaber, Stefan Würsten, Harald C. Gall, and Philipp Leitner. Dynamically Reconfiguring Software Microbenchmarks: Reducing Execution Time without Sacrificing Result Quality. In *Proc. of European Software Engineering Conference and Symp. on the Foundations of Software Engineering*, 2020.

[21] David Shue, Michael J. Freedman, and Anees Shaikh. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *Proc. of USENIX Conf. on Operating Systems Design and Implementation*, 2012.

[22] Soumendra Nath Lahiri. *Resampling Methods for Dependent Data*. Springer Science & Business Media, 2013.

[23] Dimitris N. Politis and Halbert White. Automatic Block-Length Selection for the Dependent Bootstrap. *Econometric Reviews*, 23(1):53–70, 2004.

[24] John A Gubner. *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, 2006.

[25] Amazon. Amazon Web Services. https://aws.amazon.com. [Online].

[26] Google. https://cloud.google.com/appengine/docs. [Online].

[27] A Configurable Experimental Environment for Large-scale Cloud Research. https://www.chameleoncloud.org/. [Online].

[28] A Colin Cameron and Pravin K Trivedi. *Microeconometrics: Methods and Applications*. Cambridge university press, 2005.

[29] Bradley Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*. SIAM, 1982.

[30] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, 2013.

[31] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, et al. The NAS Parallel Benchmarks Summary and Preliminary Results. In *Int'l Conf. on Supercomputing*, 1991.

[32] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2012.

[33] Patrick Schober, Christa Boer, and Lothar A Schwarte. Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia*, 126(5):1763–1768, 2018.

[34] M. Hajjat, R. Liu, Y. Chang, T. S. E. Ng, and S. Rao. Application-specific configuration selection in the cloud: Impact of provider policy and potential of systematic testing. In *IEEE Conf. on Computer Communications (INFOCOM)*, 2015.

[35] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. *A Modern Introduction to Probability and Statistics*. Springer Science & Business Media, 2005.

[36] Oracle. Java Platform, Enterprise Edition (Java EE) 7. https://docs.oracle.com/javaee/7/index.html. [Online].

[37] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proc. of ACM Symposium on Cloud Computing*, 2010.

[38] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proc. VLDB Endow.*, 7(4), December 2013.

[39] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[40] C. Hsu, V. Nair, V. W. Freeh, and T. Menzies. Arrow: Low-Level

Augmented Bayesian Optimization for Finding the Best Cloud VM. In *IEEE Int'l Conf on Distributed Computing Systems*, 2018.

[41] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost- and Deadline-constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[42] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal Cloud Resource Auto-Scaling for Web Applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 58–65, 2013.

[43] M. Albonico, S. D. Alesio, J. Mottu, S. Sen, and G. Sunyé. Generating Test Sequences to Assess the Performance of Elastic Cloud-Based Systems. In *IEEE Int'l Conference on Cloud Computing (CLOUD)*, 2017.

[44] Ali Abedi and Tim Brecht. Conducting Repeatable Experiments in Highly Variable Cloud Computing Environments. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017.

[45] H. M. Alghmadi, M. D. Syer, W. Shang, and A. E. Hassan. An Automated Approach for Recommending When to Stop Performance Tests. In *Int'l Conf. on Software Maintenance and Evolution*, 2016.

[46] Lubomír Bulej, Vojtěch Horký, Petr Tuma, François Farquet, and Aleksandar Prokopec. Duet Benchmarking: Improving Measurement Accuracy in the Cloud. In *ACM/SPEC Int'l Conf. on Performance Engineering*, ICPE '20, 2020.

[47] Luke Bertot, Stéphane Genaud, and Julien Gossa. Improving Cloud Simulation Using the Monte-Carlo Method. In *Euro-Par: Parallel Processing*, 2018.

[48] Steffen Becker, Lars Grunske, Raffaela Mirandola, and Sven Overhage. Performance prediction of component-based systems. In Ralf H. Reussner, Judith A. Stafford, and Clemens A. Szyperski, editors, *Architecting Systems with Trustworthy Components*, pages 169–192, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[49] Catia Trubiani, Indika Meedeniya, Vittorio Cortellessa, Aldeida Aleti, and Lars Grunske. Model-Based Performance Analysis of Software Architectures under Uncertainty. In *Proceedings of Int'l ACM Sigsoft Conf. on Quality of Software Architectures*, 2013.

[50] Ivan Postolski, Victor Braberman, Diego Garbervetsky, and Sebastian Uchitel. Simulator-Based Diff-Time Performance Testing. In *Proc. of Int'l Conf. on Software Engineering: New Ideas and Emerging Results*, 2019.

[51] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting Performance via Automated Feature-interaction Detection. In *Int'l Conf. on Software Engineering (ICSE)*, 2012.

[52] Guoliang Zhao, Safwat Hassan, Ying Zou, Derek Truong, and Toby Corbin. Predicting Performance Anomalies in Software Systems at Run-time. *ACM Transactions on Software Engineering and Methodology*, December 2020.

[53] Xusheng Xiao, Shi Han, Dongmei Zhang, and Tao Xie. Context-Sensitive Delta Inference for Identifying Workload-Dependent Performance Bottlenecks. In *International Symposium on Software Testing and Analysis*, ISSTA 2013, 2013.

[54] H. Ha and H. Zhang. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. In *IEEE/ACM Int'l Conf on Software Engineering (ICSE)*, 2019.

[55] Yuhui Lin, Adam Barker, and John Thomson. Modelling VM Latent Characteristics and Predicting Application Performance using Semi-supervised Non-negative Matrix Factorization. In *IEEE Int'l Conf. on Cloud Computing (CLOUD)*, 2020.

[56] Emilio Coppa, Camil Demetrescu, and Irene Finocchi. Input-sensitive Profiling. In *Proc. of the Conf. on Programming Language Design and Implementation*, 2012.

[57] Sudipta Chattopadhyay, Lee Kee Chong, and Abhik Roychoudhury. Program Performance Spectrum. In *Proc. of ACM Conf. on Languages, Compilers and Tools for Embedded Systems*, 2013.

[58] Mark D. Syer, Zhen Ming Jiang, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Continuous Validation of Load Test Suites. In *Int'l Conf. on Performance Engineering*, 2014.

[59] Cornel Barna, Marin Litoiu, and Hamoun Ghanbari. Autonomic Load-testing Framework. In *Int'l Conf. on Autonomic Computing*, 2011.

[60] Jacob Burnim, Sudeep Juvekar, and Koushik Sen. WISE: Automated Test Generation for Worst-case Complexity. In *Proc.s of Int'l Conference on Software Engineering*, 2009.

[61] Pingyu Zhang, Sebastian Elbaum, and Matthew B. Dwyer. Automatic generation of load tests. In *Proc. of Int'l Conf. on Automated Software Engineering*, 2011.

[62] Bihuan Chen, Yang Liu, and Wei Le. Generating Performance Distributions via Probabilistic Symbolic Execution. In *Proc. of Int'l Conf on Software Engineering*, 2016.

[63] Michael Pradel, Markus Huggler, and Thomas R. Gross. Performance Regression Testing of Concurrent Classes. In *Int'l Symp. on Software Testing and Analysis*, 2014.

[64] Xue Han, Tingting Yu, and David Lo. PerfLearner: Learning from Bug Reports to Understand and Generate Performance Test Frames. In *ACM/IEEE Int'l Conf on Automated Software Engineering*, 2018.

[65] D. Marijan and M. Liaaen. Effect of Time Window on the Performance of Continuous Regression Testing. In *IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME)*, 2016.

[66] Pengyu Nie, Ahmet Celik, Matthew Coley, Aleksandar Milicevic, Jonathan Bell, and Milos Gligoric. Debugging the Performance of Maven's Test Isolation: Experience Report. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, 2020.

[67] Marija Selakovic, Thomas Glaser, and Michael Pradel. An Actionable Performance Profiler for Optimizing the Order of Evaluations. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, 2017.

[68] A. Banerjee, S. Chattopadhyay, and A. Roychoudhury. Static analysis driven cache performance testing. In *IEEE 34th Real-Time Systems Symposium*, 2013.

[69] Hazem Samoaa and Philipp Leitner. An Exploratory Study of the Impact of Parameterization on JMH Measurement Results in Open-Source Projects. In *ACM/SPEC Int'l Conf. on Performance Engineering*, ICPE '21, 2021.

[70] Hammam M. AlGhamdi, Cor-Paul Bezemer, Weiyi Shang, Ahmed E. Hassan, and Parminder Flora. Towards reducing the time needed for load testing. *Journal of Software: Evolution and Process*, page e2276, 2020.

[71] S. Mühlbauer, S. Apel, and N. Siegmund. Identifying Software Performance Changes Across Variants and Versions. In *IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE)*, 2020.

[72] C. Trubiani and S. Apel. PLUS: Performance Learning for Uncertainty of Software. In *IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019.