A Statistics-Based Performance Testing Methodology for Cloud Applications

Sen He
sen.he@utsa.edu
University of Texas at San Antonio
USA

Glenna Manns
gmn6jd@virginia.edu
University of Virginia
USA

John Saunders
js8ra@virginia.edu
University of Virginia
USA

Wei Wang
wei.wang@utsa.edu
University of Texas at San Antonio
USA

Lori Pollock
pollock@udel.edu
University of Delaware
USA

Mary Lou Soffa
soffa@virginia.edu
University of Virginia
USA

ABSTRACT

The low cost of resource ownership and flexibility have led users to increasingly port their applications to the clouds. To fully realize the cost benefits of cloud services, users usually need to reliably know the execution performance of their applications. However, due to the random performance fluctuations experienced by cloud applications, the black box nature of public clouds and the cloud usage costs, testing on clouds to acquire accurate performance results is extremely difficult. In this paper, we present a novel cloud performance testing methodology called PT4Cloud. By employing non-parametric statistical approaches of likelihood theory and the bootstrap method, PT4Cloud provides reliable stop conditions to obtain highly accurate performance distributions with confidence bands. These statistical approaches also allow users to specify intuitive accuracy goals and easily trade between accuracy and testing cost. We evaluated PT4Cloud with 33 benchmark configurations on Amazon Web Service and Chameleon clouds. When compared with performance data obtained from extensive performance tests, PT4Cloud provides testing results with 95.4% accuracy on average while reducing the number of test runs by 62%. We also propose two test execution reduction techniques for PT4Cloud which can reduce the number of test runs by 90.1% while retaining an average accuracy of 91%. We compared our technique to three other test execution reduction techniques for PT4Cloud and shows, the performance distributions obtained from the two tests are drastically different, and their average throughputs are different by 12%. It is evident that these two test results cannot be both accurate. In fact, as shown in Section 5, both results are inaccurate, and their average throughputs are different by 12%. It is evident that these two test results cannot be both accurate.

CCS CONCEPTS

- General and reference → Performance; → Computer systems organization → Cloud computing; → Software and its engineering → Software testing and debugging.

KEYWORDS

performance testing, cloud computing, resource contention, non-parametric statistics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION

The low costs of ownership, flexibility, and resource elasticity have prompted many organizations to shift applications to the cloud, in particular, Infrastructure-as-a-Service (IaaS) clouds, to host their applications [54]. For cloud deployments, it is critical that the users have accurate knowledge of the performance of their applications so that they can select the appropriate virtual machine (VM) configurations to satisfy their performance and cost objectives [47, 59, 63]. The effectiveness of cloud elasticity policies also relies on the accurate knowledge of application performance [4, 23, 28, 45, 65].

The most reliable approach to determine the performance of an application on the cloud is performance testing. To obtain accurate results, performance testing usually has two requirements [12, 14, 48, 64]. First, the test inputs should be accurately generated based on the desired use cases. Second, it requires that the performance of each test input is independently and accurately determined. For better accuracy, a common practice is to run the application-under-test with a test input multiple times to obtain the average or certain percentiles of its performance [3, 14, 48, 53].

However, it extremely difficult to determine when enough performance test runs are executed and accurate results are obtained on the cloud. For example, Figure 1 gives the results of two performance tests of the same benchmark, YCSB, on the same Amazon Web Service (AWS) VM [6, 20]. For each test, the same test input is executed repeatedly for an extended period of 15 hours. As Figure 1 shows, the performance distributions obtained from the two tests are drastically different, and their average throughputs are different by 12%. It is evident that these two test results cannot be both accurate. In fact, as shown in Section 5, both results are inaccurate, and more test runs need to be conducted to get reliable performance. This difficulty to get reliable performance is also a major obstacle faced by system research [2].

The difficulty of getting accurate performance testing results on the cloud is caused by cloud performance uncertainty, which...
constitutes the considerable and unpredictable performance fluctuation of cloud applications [37]. Because of this uncertainty, a new performance testing methodology is required for the cloud to obtain accurate performance for each test input. In particular, a good methodology should address the following challenges posed by cloud performance uncertainty.

The first challenge arises from the various factors that cause performance uncertainty, including hardware resource contentions from multi-tenancy (multiple VMs sharing hardware) and the randomness in VM scheduling [26, 31, 43, 58]. To obtain accurate results, performance testing on a cloud should cover all the uncertainty factors. Even more importantly, it should cover the uneven impacts of all these factors proportional to their frequencies experienced on the cloud.

The second challenge is that cloud services are usually provided to the users as black boxes and do not allow users to control the execution environments. This black-box environment makes it nearly impossible to know what uncertainty factors are covered by a test run. Hence, it is also impossible to know exactly when enough runs have been conducted to cover all uncertainty factors.

The third challenge comes from the cloud usage cost incurred by performance testing. Theoretically, executing a test input for months can produce accurate results. However, such long tests can incur a high cost. To minimize testing costs, a good performance testing methodology should stop testing immediately after it determines that the results are accurate.

In this paper, we present a novel performance testing methodology called PT4Cloud for IaaS clouds. Our primary goal with PT4Cloud is to provide reliable stop conditions to terminate repeated test runs of a test input to obtain highly accurate performance result. While ensuring high accuracy, our second goal is to reduce the number of test runs to cut down on the cost of performance testing. PT4Cloud is designed to obtain performance distributions, since in many use cases, it is important to know the best-case, worst-case and percentiles of the performance in addition to averages [14, 64].

To determine the number of performance test runs required for accurate result for one test input in black-box clouds, PT4Cloud leverages the observation of statistical stability, which states that the frequencies and averages converge (i.e., become stable) given a large number of samples [27]. Based on this observation, if a performance distribution obtained from the test runs is stable, this distribution may be deemed accurate. That is, performance test runs can be stopped once the results are statistically stable with the expectation that the results are accurate, and all uncertainty factors are properly covered.

Moreover, as the performance distributions of cloud applications do not always follow known distributions, common parametric statistical approaches (e.g., Student’s t-test) cannot be used for stability determination. Therefore, PT4Cloud employs non-parametric statistical approaches from likelihood theory [42, 57]. These statistical approaches also allow the users to easily specify accuracy objectives and trade accuracy for testing cost. Additionally, to help users interpret the performance testing results with more confidence, we employ the bootstrap method to generate confidence bands for the resulting performance distributions [21]. To further reduce the testing cost, we also explored two test reduction techniques.

We evaluated PT4Cloud on the Chameleon cloud [1] and AWS [6], using six benchmarks representing web, database, machine learning and scientific applications on six VM configurations. The evaluation results show that the performance distributions acquired with PT4Cloud always have an accuracy higher than 90% (with 95.4% accuracy on average) when compared with the performance results obtained from extensive benchmark executions while reducing test runs by 62%. Moreover, our test reduction techniques can reduce the test runs by 90.1% while retaining an average accuracy of 91%. Our results also showed that PT4Cloud had significantly higher accuracy than state-of-the-art testing and prediction methodologies from software engineering and system research.

The contributions of this paper include:
1) The cloud performance testing methodology, PT4Cloud, which employs non-parametric statistical approaches to provide reliable stopping conditions to obtain highly accurate performance distributions. PT4Cloud also allows the users to specify intuitive accuracy objectives and trade accuracy for testing cost.
2) Two test reduction techniques that can significantly reduce the number of test runs while retaining a high level of accuracy.
3) A thorough evaluation of PT4Cloud with six benchmarks on six VM configurations on AWS and Chameleon cloud to examine the PT4Cloud’s accuracy and test reduction efficiency, and its benefit over the state-of-the-art approaches.

2 OVERVIEW OF PT4CLOUD

2.1 The PT4Cloud Methodology

PT4Cloud conducts performance tests on cloud applications in multiple time intervals (periods) of test runs. In each time interval (time period), the application-under-test is executed with its test input repeatedly to acquire n performance samples. Then PT4Cloud determines if adding these new n samples significantly changes the performance distribution acquired from previous intervals. If the
change is insignificant, then the current performance distribution is considered stable and representative of the actual performance of the application on the cloud. Otherwise, more test runs for another time interval are required. We conduct the test in time intervals instead of the number of test runs due to the difficulty to control the cloud execution environments. More discussions on this time interval are provided in Section 2.2.

Figure 2 shows the PT4Cloud workflow. The first step (Step 1) of PT4Cloud is to execute an application on the target cloud with a user-selected VM configuration (i.e., VM types and count) repeatedly for a time interval \( I \). Let the set of performance data collected from these tests be \( S_1 \). Based on \( S_1 \), an initial performance distribution \( d_1 \) can be calculated. Particularly, \( d_1 \) here represents the non-parametric probability density function (PDF) of \( S_1 \) and can be calculated using the kernel density estimation (KDE) technique [52, 55].

In Step 2, the application is again executed for another time interval \( I \) using the same VM configuration. Let the set of performance data from these new test runs be \( S_2 \). Combining \( S_1 \) and \( S_2 \), we obtain a new set of performance data, \( S \). That is, \( S = S_1 \cup S_2 \). Using \( S \), it is possible to calculate another performance distribution \( d_2 \). Intuitively, \( d_1 \) always represents the current distribution, whereas \( d_2 \) always represents the distribution with new data. If \( d_1 \) and \( d_2 \) are the same, then adding new data does not change the distribution.

In Step 3, \( d_1 \) and \( d_2 \) are compared using non-parametric statistical approaches to determine if the performance distributions are stable. More specifically, this comparison determines the probability \( p \) that \( d_1 \) and \( d_2 \) are the same distribution. Users can choose an objective probability \( p_o \) beforehand. If \( p \geq p_o \), then the performance distributions are deemed stable, and \( d_2 \) is reported as the performance distribution of this application. If \( p < p_o \), then more tests need to be conducted to acquire more stable results. Note that, a user can choose any objective probability. A higher objective probability may require more tests, but it can also produce more accurate performance results. When reporting performance distribution \( d_2 \), PT4Cloud also computes confidence bands with a user-selected confidence level (CL). While a confidence interval is for a single point of estimation (e.g., mean), a confidence band is for a series of estimations (e.g., distribution).

If the test results are deemed unstable, more test runs are required. In Step 4, the new \( S_1 \) is set to be \( S \) and the new \( d_1 \) to be \( d_2 \). That is, \( S_1 = S \) and \( d_1 \Leftarrow d_2 \). Then PT4Cloud directs the performance test to go back to Step 2 to test for another time interval \( I \) and repeat the comparison for stability. This loop repeats until the performance results are stable.

2.2 Coverage Criteria and Time Interval Length
As stated above, the coverage criteria for cloud performance testing should include all performance uncertainty factors and their proportional impacts. However, as clouds are provided to the users as black boxes, it is impossible to directly determine what factors are covered in one test or a series of tests. Consequently, we adopted an indirect approach to ensure the coverage criteria are satisfied.

Previous studies observed that application performance on the cloud roughly exhibits periodic (e.g., daily or weekly) cycles [38]. These periodic cycles reflect the fact that the main factors of performance fluctuation – various resource contentions among VMs caused by multi-tenancy – have a dependency on time. Motivated by this observation, we choose to conduct tests in terms of multiple time intervals to satisfy the coverage criteria. In this paper, we choose the length of each time interval \( I \) to be a week. Based on a prior study and our experimental results for this paper, a week is long enough to provide good coverage of uncertainty factors for the types of applications and cloud services studied in this paper [38].

Note that, a week is the longest time period required for cloud performance testing based on our observations. Depending on the application and the VM configuration, the smallest required time interval length may be less than a week. It is also unnecessary to continuously execute tests for each time interval. We also explored reducing the interval length and test runs per time interval and reported the results in Section 6.

2.3 Examples of Applying PT4Cloud
To illustrate the complete process of using PT4Cloud methodology to conduct performance testing, we present two examples of testing the performance of canneal and swaptions applications from the PARSEC benchmark suite [11] on AWS.

Figure 3a gives the performance distribution \( d_1 \) for canneal after Step 1, where canneal is executed continuously on a VM instance with VM type `t2.medium` for a time interval length of a week [5]. Figure 3a also gives the performance distribution \( d_2 \) for canneal after Step 2, which is calculated from two time intervals (weeks) of test runs. Using the stability comparison in Step 3, the probability that \( d_1 \) and \( d_2 \) are the same is determined to be 92.8%. As shown in Figure 3a, \( d_1 \) is very close to \( d_2 \). If the user sets the objective probability \( p_o \) to be 90%, then \( d_2 \) is reported as the final testing result. The shaded area in Figure 3a shows the confidence band of \( d_2 \) with 99% CL.

Figure 3b gives the performance distribution \( d_1 \) for swaptions after Step 1, which includes executing swaptions continuously on a `t2.medium` instance for a week. Figure 3b also shows the performance distribution \( d_2 \) for swaptions after Step 2, which is calculated from two intervals/weeks of test runs. Using the stability comparison in Step 3, the probability that \( d_1 \) and \( d_2 \) are the same is determined to be 80.7%. In Figure 3b, \( d_1 \) and \( d_2 \) are still close, but they are clearly less similar than those of canneal. If the user’s objective probability \( p_o \) is 80% or less, then \( d_2 \) is reported as the final testing result. Otherwise, the test input has to be executed from more time intervals for more accurate results.
3 STATISTICS-BASED STOP CONDITIONS

Section 2 states that PT4Cloud determines whether the performance test can be stopped by determining if two distributions, $d_1$ and $d_2$, are similar (stable). That is, PT4Cloud determines if adding another interval of tests significantly changes the performance distribution.

A key issue here is to identify the proper statistical approach for this distribution comparison. A statistical approach that is proper for cloud performance testing should satisfy three requirements. First, the approach should be able to handle non-typical distributions, as performance distributions of cloud applications usually do not follow known distributions. Second, this approach should be able to handle distributions acquired from experiments, as it is practically impossible to make a hypothesis about the exact theoretical distribution for a cloud application’s performance. Third, the comparison result from this approach should be intuitive so that the ordinary user can understand. The comparison result should also provide a quantitative definition for “significant change in distributions” to ensure testing results are accurate.

The most common approach for distribution comparison is Goodness of Fit (GoF) statistical tests [49]. However, many GoF tests are only designed for specific types of distribution (e.g., Shapiro–Wilk test for normal distributions) or require one distribution to be theoretical instead of experimental (e.g., Kolmogorov–Smirnov test) [49]. The Anderson–Darling test is a generic GoF test that can compare two distributions acquired from experiments. However, this test requires critical values that do not yet exist for the non-typical cloud performance distributions. The Chi-square ($\chi^2$) test is another generic GoF test. To use $\chi^2$-test, one needs to first divide the range of performance data (e.g., execution times) acquired from tests into bins. Unfortunately, the $\chi^2$-test is sensitive to the widths of the bins [49]. We experimented with $\chi^2$-test, but were unable to find a bin width that worked well for the diverse types of distributions obtained from cloud performance tests.

The distribution comparison approach that we eventually adopted is Kullback–Leibler (KL) divergence, which can handle any types of distributions [42]. More specifically, KL-divergence measures how one distribution diverges from a second distribution. Without loss of generality, consider two distributions $P$ and $Q$ over random variable $x$. The equation to compute how $P$ diverges from $Q$ with KL-divergence is,

$$D_{\text{KL}}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx.$$  

(1)

The value of KL-divergence (i.e., $D_{\text{KL}}(P||Q)$) ranges from 0 to infinity. A value of 0 for KL-divergence indicates no divergence, whereas infinity indicates two distributions are completely different. However, this interpretation is not intuitive for users to understand the amount of difference (divergence). To help a user interpret KL-divergence, we employed multinomial likelihood $L$ [57] from the likelihood theory, which can be computed as,

$$L(P||Q) = 2^{-D_{\text{KL}}(P||Q)}$$  

(2)

Intuitively, $L$ represents the probability that $P$ is different from $Q$. As our goal is to compare the similarity between distributions $d_1$ and $d_2$, it requires considering $d_1$ and $d_2$ symmetrically. That is, if $d_1$ and $d_2$ are similar, then $d_1$ is different from $d_2$, and $d_2$ is not different from $d_1$. Therefore, we define the probability $p$ that $d_1$ and $d_2$ are similar as,

$$p = L(d_1||d_2) \times L(d_2||d_1)$$  

(3)

As described in Section 2, $p$ is the probability used in PT4Cloud to determine whether the performance distributions obtained from performance tests are stable. Note that, to compute the integration in Eq (1), we employed numerical integration by partitioning each distribution into 1000 strips. For our experiments, using 1000 strips was sufficient, adding more strips changes the probabilities by less than 0.1%.

Note that, KL-Divergence is commonly used as an asymmetric metric. We used the symmetric KL-Divergence following its original definition as we treat $d_1$ and $d_2$ equally in the comparison. [42]. A potential variant of PT4Cloud may use asymmetric KL-Divergence. While using asymmetric KL-Divergence does not affect the results of our experiments, its exact impact requires more investigation.

4 ESTABLISHING CONFIDENCE BANDS

To help users better understand their application’s performance and interpret the performance testing results with higher confidence, PT4Cloud also presents each final performance distribution with its confidence band.

Because the performance distributions of cloud applications do not necessarily follow known distributions, we chose to compute point-wise confidence bands (CB) using bootstrap [19, 34, 44]. Point-wise confidence bands are commonly used to describe non-parametric distributions, and bootstrap is a statistical method for treating non-parametric distributions.

Bootstrap is essentially a resampling technique. To generate a CB, the original performance data set $S$ is resampled. Each resample samples a new data set with $|S|$ data points from $S$ with replacement, and a new probability density function (PDF) is generated based on the new data set. Repeating the resampling for $R$ times allows us to calculate $R$ PDFs. Then a point-wise confidence band with confidence level (CL) $\%$ can be determined by calculating the probability density region that contains $\%$ of the $R$ Bootstrapped PDFs. Figure 3 also shows the confidence bands with shaded areas.

Bootstrap can produce correct results assuming re-sampling on the data set $S$ behaves similarly to when $S$ is sampled from the true population. This assumption is usually true when $S$ is complete and $R$ are sufficiently large [19]. We set $R$ to be 1000, following common practice [19]. $S$ is deemed complete by PT4Cloud. Therefore, as long as PT4Cloud methodology is accurate, confidence band generated with $S$ is also reliable.

5 EXPERIMENTAL EVALUATION

This section presents the methodology and findings from evaluating PT4Cloud on two public clouds. This evaluation seeks to answer the following research questions: 1) How accurate are the performance distributions acquired with PT4Cloud? 2) How does PT4Cloud compare with the state of the art?

5.1 Experimental Setup

**Benchmarks** We evaluated PT4Cloud with a variety of benchmarks that represent web applications, high-performance computing (HPC) applications, database (DB) applications and machine
Table 1: Benchmarks and their application domains.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Domain</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft.C</td>
<td>HPC</td>
<td>NPB</td>
</tr>
<tr>
<td>ep.C</td>
<td>HPC</td>
<td>NPB</td>
</tr>
<tr>
<td>JPetStore</td>
<td>Web</td>
<td>J2EE</td>
</tr>
<tr>
<td>YCSB</td>
<td>DB</td>
<td>YCSB</td>
</tr>
<tr>
<td>TPC-C</td>
<td>DB</td>
<td>OLTPBench</td>
</tr>
</tbody>
</table>

Table 2: VM configurations used for evaluation.

<table>
<thead>
<tr>
<th>Config.</th>
<th>VM Cnt x VM Type</th>
<th>Core-Cnt</th>
<th>Mem Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHM-1</td>
<td>4 x Small (Sm)</td>
<td>1/VM</td>
<td>2GB/VM</td>
</tr>
<tr>
<td>CHM-2</td>
<td>2 x Medium (Md)</td>
<td>2/VM</td>
<td>4GB/VM</td>
</tr>
<tr>
<td>CHM-3</td>
<td>1 x Large (Lg)</td>
<td>4/VM</td>
<td>8GB/VM</td>
</tr>
<tr>
<td>AWS-1</td>
<td>4 x m5.large (M5Lg)</td>
<td>2/VM</td>
<td>8GB/VM</td>
</tr>
<tr>
<td>AWS-2</td>
<td>2 x m5.xlarge (M5XLg)</td>
<td>4/VM</td>
<td>16GB/VM</td>
</tr>
<tr>
<td>AWS-3</td>
<td>1 x m5.2xlarge (M52XLg)</td>
<td>8/VM</td>
<td>32GB/VM</td>
</tr>
</tbody>
</table>

Table 3: The number of intervals/weeks (W) required to obtain stable performance distributions, and the accuracy of the performance distributions obtained with PT4Cloud.

<table>
<thead>
<tr>
<th>“Test Run Length” / “Accuracy (%)”</th>
<th>CHM-1</th>
<th>CHM-2</th>
<th>CHM-3</th>
<th>AWS-1</th>
<th>AWS-2</th>
<th>AWS-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPetStore</td>
<td>N/A</td>
<td>2W/98</td>
<td>2W/99</td>
<td>3W/96</td>
<td>2W/99</td>
<td>2W/96</td>
</tr>
<tr>
<td>YCSB</td>
<td>3W/94</td>
<td>3W/93</td>
<td>2W/93</td>
<td>2W/93</td>
<td>2W/93</td>
<td>2W/93</td>
</tr>
<tr>
<td>TPC-C</td>
<td>2W/94</td>
<td>2W/92</td>
<td>2W/97</td>
<td>2W/96</td>
<td>2W/95</td>
<td>2W/95</td>
</tr>
<tr>
<td>IMA</td>
<td>N/A</td>
<td>2W/95</td>
<td>2W/95</td>
<td>2W/94</td>
<td>2W/96</td>
<td>2W/95</td>
</tr>
</tbody>
</table>

learning (ML) applications to demonstrate that PT4Cloud can be applied to diverse cloud applications. Table 1 gives the benchmarks and their application domains that were used in our evaluation. More specifically, we used JPetStore (JPS) from Java 2 Platform Enterprise Edition (J2EE), ft and ep from NAS Parallel Benchmarks (NPB), Yahoo! Cloud System Benchmark (YCSB) with Apache Cassandra database, TPC-C from OLTPBench and In-Memory Analytics (IMA) from CloudSuite [7, 17, 20, 22, 51]. For ft and ep, we used the NPB’s C workloads so that the input data can fit in the memory of a medium sized VM. For YCSB, we used its workload A which has 50% reads and 50% writes.

Public Clouds and VM Configurations To show that PT4Cloud works properly on public clouds, we evaluated it on two public clouds, Chameleon (CHM) [1], and the Amazon Web Services EC2 (AWS) [6]. We used three VM configurations on each cloud that represent use cases with both single and multiple VMs. Table 2 details the VM configurations used in our experiments. Note that AWS has a large selection of VM types; in this work, we chose m5 VMs as they are the latest general purpose VMs. Except for ft, ep and IMA, each benchmark was tested on all configurations. Due to insufficient memory, ft, ep and IMA could not execute on Chameleon’s small VMs. In the rest of this paper, we call an experiment with one benchmark on one VM configuration as a benchmark configuration. In total, 33 benchmark configurations are evaluated.

Parameters of PT4Cloud For this evaluation, we chose the object probability ($p_o$) for distribution stability comparison to be 90% (i.e., expecting the accuracy of performance testing results to be at least 90%). We also set the confidence level (CL) to be 99% for the confidence bands generation. Additionally, as stated in Section 2, we used a time interval length of one week in this evaluation.

Evaluation Methodology and Metric For each benchmark configuration, we evaluate the accuracy of its performance distribution with PT4Cloud by comparing this distribution with a ground truth performance distribution. We then report the probability (i.e., the multinomial likelihood introduced in Section 3) that the PT4Cloud and ground truth distributions are the same. For the rest of this paper, we simply refer to this probability as accuracy. To obtain the ground truth distribution, we executed each benchmark configuration for six weeks (in addition to the performance tests conducted with PT4Cloud) and calculated the performance distributions using the performance data from these six weeks.


5.2 Perf. Dist. Acquired with PT4Cloud

Figure 4 presents the performance distributions acquired with PT4Cloud for eight benchmarks configurations. Due to space limitation, only four benchmarks on two VM configurations – CHM-2 and AWS-2 – are shown in Figure 4. The rest of the benchmarks and configurations have similar results. Figure 4 shows that the performance distributions for these benchmarks gradually became stable over time. That is, the changes in the distributions become smaller after each new interval/week. It can also be seen from Figure 4 that the distributions from the first interval/week can be dramatically different from the final stable distributions, suggesting the necessity of a performance testing methodology like PT4Cloud. The distributions in Figure 4 also do not always follow known distributions and vary with benchmark configurations, proving the need for using non-parametric statistical methods as employed by PT4Cloud.

Table 3 provides the numbers of intervals of test runs that were conducted to obtain stable performance distributions with more than 90% stability probability (i.e., $p_o$ is 90%) for all benchmark configurations. The performance distributions of the majority of the benchmarks were stable within two weeks. On certain VM configurations, the distributions of DB and web applications required three weeks to become stable as I/O (network and disk) performance has higher fluctuations than CPU and memory performance [43].

5.3 Accuracy of PT4Cloud

Figure 5 compares the performance distributions obtained with PT4Cloud and the ground truth performance distributions. Due to space limitation, only four benchmarks on two VM configurations are shown in Figure 5. As the figure shows, the ground truth performance is very close to the performance obtained from PT4Cloud.

Table 3 also gives the accuracy of PT4Cloud’s performance distributions when compared with ground truth distributions for all benchmark configurations. As shown in Table 3, the accuracy of PT4Cloud’s performance distributions is always higher than 90%. The average accuracy is 95.4%. These results indicate that PT4Cloud methodology is highly accurate for cloud performance testing. Moreover, PT4Cloud methodology executed considerably fewer tests than the ground truth tests. Figure 8 gives how many fewer
On average, \textit{PT4Cloud} executed 62% fewer test runs. Figure 5 also shows the confidence bands (CB) for the final distributions with 99% CL. With \( \rho_0 \) being 90%, the probability that the true distribution falling within the CB is expected to be roughly 90% \( \times \) 99% = 89%. Note the ground truth distributions do not necessarily fall within the confidence bands with this same probability, as they are still experimental (i.e., not true) distributions. Nonetheless, the majority of the ground truth distributions (including those not shown in Figure 5), fall within the confidence bands.

5.4 Comparison with State-of-the-Art

To demonstrate the importance of a new methodology like \textit{PT4Cloud} for performance testing on the cloud, we compared \textit{PT4Cloud} with three performance test stopping and performance prediction methodologies from software engineering and computer system research.
The first methodology (SotA1) stops the performance test by detecting the repetitiveness of the performance data obtained from test runs [3]. More specifically, it picks a short period of data from the whole performance testing results and compares this period with other periods in the whole data set to determine if this period of data is repeated. The methodology repeats this process for 1000 times to estimate the percentage of the performance results that are repeated (i.e., repetitiveness). If the repetitiveness remains unchanged for a user-defined time span, then the test can be stopped. We reproduced this methodology using the 18 sets of parameters given in that paper and reported the accuracy of the performance distributions obtained with this methodology in Figure 6. For each benchmark configuration, only the best accuracy of the 18 sets of parameters is reported. For all of the benchmark configurations, this methodology stopped the tests relative early — always in less than 450 minutes on AWS and 250 minutes on Chameleon. Due to space limitation, we cannot provide detailed stop times.

As Figure 6 shows, this first methodology has an average accuracy of only 65.7% due to the early stop times. Many benchmark configurations experienced lower than 50% accuracy. We observe that this low accuracy is mainly because this methodology is only designed to detect performance changes due to an application’s internal factors instead of external factors, such as cloud performance uncertainty factors. For internal factors, only measuring whether a datum is repeated or not may be enough. However, for external factors, how frequent each datum being repeated is also important. Additionally, we found that the accuracy of this methodology relied heavily on its parameters, i.e., the length of the period and the user-defined time span. Nonetheless, it is unclear how to select these parameters to maximize accuracy.

The second methodology we compared (SotA2) is a classical technique for performance analysis [39, 46, 49]. In this methodology, the performance test is stopped if the confidence interval (CI) for certain statistics “narrows to a desired width” [39, 46]. As we tested for performance distribution, we extended this idea to stop running the tests if the 99% CI for each percentile of the performance dropped within 10% of the observed percentile performance. This 10% is chosen to reflect ±5% margin of error, similar to our 90% accuracy goal (i.e., p0). With this extension, we applied this methodology to our benchmarks. The CIs were generated using the bootstrap approach. This methodology caused the performance testing to stop at variable times, from 2 hours to even 7 weeks. The extra long tests were caused by performance outliers, which made the CIs of certain fringe percentiles harder to converge. The accuracy of the performance distributions obtained with this methodology is also reported in Figure 6. As Figure 6 shows, the average accuracy for this methodology is only 66.2%, with the lowest accuracy being 8.73% (EP-CHM3).

This low accuracy shows that CI width is a poor stopping criteria for cloud performance testing. That CIs failed to determine required sample sizes is also observed in other science fields [29]. The main issue is that, for non-parametric distributions, CIs are reliable only when data is complete [19]. That is, for cloud performance testing results, the CIs are only reliable when the results covers all uncertainty factors. A narrow CI simply means that there are large amount of performance data obtained under the observed uncertainty factors, not that all factors are observed by the tests.

We compared our technique with a third methodology, which was a performance distribution prediction technique for cloud using Monte Carlo simulation [10]. This methodology can be applied to applications with multiple steps assuming the min and max performance of each step are known. Two of our benchmarks, YCSB and ft.C, have two steps. Thus, we applied this prediction technique on the 11 benchmark configurations involving these two benchmarks using the min and max performance obtained from the ground truth. For ft.C, the distributions predicted by this technique have accuracies of 10%, 11%, 16%, 21%, 8%, on the five VM configurations from CHM-2 to AWS-3. For YCSB, the accuracies are 42%, 22%, 20%, 39%, 29%, 16% for CHM-1 to AWS-3. The average accuracy for the 11 configurations is 21.3%. The low prediction accuracy is primarily because this technique assumes each step has a uniform performance distribution, which is not true on real public clouds. In fact, if the stable performance distribution for each step obtained with PT4Cloud is used for this method, the average prediction accuracy of this methodology can be increased to 53.9% (with one configuration’s accuracy increased to 94%). This increase in accuracy shows that not only does PT4Cloud benefit performance testing, it can also benefit performance prediction techniques by providing reliable training data set. Moreover, PT4Cloud can also benefit performance prediction techniques by providing reliable ground truth to evaluate the goodness of the prediction results.

6 REDUCING THE NUMBER OF TESTS

As performance tests on clouds incur cloud usage cost, a good performance testing methodology should also strive to minimize the testing costs. In this section, we explore the approaches to
reduce the length of the intervals and test runs per interval for PT4Cloud, which in turn can reduce the test costs.

6.1 Reducing Interval Length

In Figure 4, several benchmarks’ performance distributions obtained after the first week were already very similar to the final performance distributions (e.g., fi.C and IM), suggesting that some cloud applications can use intervals shorter than one week. Indeed, the proper interval length depends on the application and the VM configuration to be tested. For instance, a CPU-intensive application is less likely to be affected by the cloud performance uncertainty factors from I/O contentions. Therefore, the interval for this application can potentially be smaller than a week, as the performance test only needs to cover a smaller number of performance uncertainty factors. However, for applications with heavy usage of disk and network, the intervals must be longer to cover the fluctuations due to all uncertainty factors.

To help users conduct tests with shorter intervals, we modified the workflow of PT4Cloud to allow users to search for proper interval lengths while conducting tests. Figure 7 depicts a variant of PT4Cloud designed for short intervals. In this new variant, the user first conducts tests for two short intervals. Then our distribution comparison technique is used to determine if the performance distribution is stable after these two short intervals. If the distribution is stable, then it can be reported as the final results. Otherwise, the interval length is increased (Step 5), and a new round of tests starts from the beginning (Step 1) with the new longer interval. The testing interval gradually increases if the results do not stabilize. When the interval length reaches one week, the standard PT4Cloud methodology is employed.

In this new variant, we choose to increase the interval length instead of conducting tests for more intervals because our primary goal is to provide highly accurate testing results. Therefore, we aggressively assume that failure to stabilize is due to a short interval rather than that insufficient intervals are tested.

A disadvantage of shorter intervals is that the resulting distributions are more likely to be inaccurate than those acquired from longer intervals, as the impacts of cloud performance uncertainty factors are more likely to remain unchanged within a short period than a longer period. For example, an application’s performance may appear to be stable within two hours, although its actual performance over a week may be quite different. Consequently, in PT4Cloud for short intervals, we also included an optional validation step (Step 4 of Figure 7). More specifically, if the performance distribution is stable with two short intervals of tests, then an additional two intervals (with same interval length) of tests are executed. The performance distribution from the additional test runs is compared with the original distribution. If both distributions are the same with a probability higher than the objective probability (p₀), then the testing result is validated and can be reported. Otherwise, the interval length has to be increased. Based on the performance results we obtained from the experimental evaluation, we recommend the user to take the validation step if the testing result is stable within a week (i.e., the interval length is less than 3.5 days).

6.2 Reducing Test Runs with Sampling

To further reduce the testing cost, especially for the applications requiring 1-week intervals, we also explored an hourly sampling technique. So far, test runs have been executing consecutively for each interval with the goal to cover every change in the behavior of the cloud performance uncertainty factors. Similar to other statistical practices, this consecutive execution of tests can be replaced with sampling to reduce cost while providing similar accuracy. Here, we employed an hour-based sampling technique. More specifically, for each hour within an interval, tests are only executed for a portion of this hour. We sampled our performance testing data obtained for the evaluation in Section 5 with various portion sizes and found that our benchmarks can achieve more than 90% accuracy with portion sizes from 1/3 to 3/4. On average, a portion size of 1/2 (i.e., sampling one half of an hour) provides highly accurate results for the majority of our benchmarks.

6.3 Evaluation of Test Reduction Techniques

We conducted additional evaluations to examine the effectiveness and accuracy of our two test reduction techniques. By effectiveness evaluation, we mean evaluating whether our test reduction techniques can effectively reduce the number of test runs. We extracted smaller intervals of performance data from the tests conducted for the evaluation in Section 5 and performed sampling on these data. For interval reduction, we explored intervals of 1 hour, 12 hours, 1 day, 2 days, 3 days, etc. up to 6 days. For hourly sampling, we used a portion size of 1/2 (i.e., sampling one half of an hour).

**Effectiveness Evaluation** Table 4 shows the reduced interval lengths to achieve at least 90% stable probability for all benchmark configurations used in Section 5. As the table shows, the interval lengths ranged from 12-hours to one week. ML and HPC benchmarks that are CPU- and/or memory-intensive are more likely to use intervals less than a week. However, DB and web applications that are I/O-intensive often require an interval of a week or close
Table 4: Reduced interval lengths and the number of intervals it takes to stabilize, for all benchmark configurations using the two test reduction techniques ("D" stands for day).

<table>
<thead>
<tr>
<th>config</th>
<th>ft.C</th>
<th>ep.C</th>
<th>JPS</th>
<th>YCSB</th>
<th>TPC-C</th>
<th>JPS</th>
<th>YCSB</th>
<th>TPC-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHM-1</td>
<td>N/A</td>
<td>3D / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>N/A</td>
<td>4D / 2</td>
<td>1D / 2</td>
</tr>
<tr>
<td>CHM-2</td>
<td>3D / 2</td>
<td>4D / 2</td>
<td>1D / 2</td>
<td>12hrs / 2</td>
<td>4D / 2</td>
<td>3D / 2</td>
<td>4D / 2</td>
<td>1D / 2</td>
</tr>
<tr>
<td>CHM-3</td>
<td>12hrs / 2</td>
<td>1W / 3</td>
<td>1D / 2</td>
<td>1W / 3</td>
<td>2D / 2</td>
<td>4D / 2</td>
<td>1W / 3</td>
<td>2D / 2</td>
</tr>
<tr>
<td>AWS-1</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
</tr>
<tr>
<td>AWS-2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
</tr>
<tr>
<td>AWS-3</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
<td>12hrs / 2</td>
</tr>
</tbody>
</table>

Figure 8: Number of tests reduced by PT4Cloud and our test reduction techniques, compared to the ground truth tests.

7 THREATS TO VALIDITY

Execution environment changes. The performance results obtained with PT4Cloud are only valid when the execution environment, including the underlying hardware and multi-tenancy behavior, remains the same. When the execution environment is changed, new performance tests should be conducted.

Furthermore, while our results indicate that the performance of cloud applications has weekly or daily cycles, events that happen only a few times a year/month may still affect overall performance distributions. An example of such yearly events may be holiday shopping where all websites running in the clouds exhibit high resource demands. In general, we believe these events do not significantly change overall performance distributions as they happen infrequently. Additionally, cloud service providers can implement resource management policies to limit the impact of these events. Our experiments on AWS actually overlapped with Amazon 2018 Prime Day when Amazon’s own website experienced errors [16]. However, we observed nearly no impact on the performance of our benchmarks. Nonetheless, the potential impact of these yearly and monthly events should be acknowledged.

Other applications, workloads and VM configurations. Although we strive to evaluate PT4Cloud with all types of cloud applications, we can only evaluate a limited number of benchmarks and VM configurations, due to cost. Other cloud applications, workloads, VM configurations and clouds may exhibit different accuracy with PT4Cloud or require different interval lengths.

Other cloud uncertainty factors. Here, we focused on the cloud performance uncertainty factors caused by multi-tenancy and VM scheduling. Other factors, such as data center location, bursty VM types and hardware variation, may also affect performance. The VMs used for our evaluations are not affected by these factors. However, these factors do exist for other cloud services and VM
types. Although PT4Cloud methodology is generic enough to handle these factors, more tests may be required to cover these factors.

**Performance results other than distributions.** In some cases, users may only need to know the mean or a particular percentile of the performance of their applications. Although an accurate performance distribution can provide accurate mean and percentiles, there may be a cheaper testing methodology to obtain them directly. However, as the means and percentiles are still from the non-typical performance distributions of cloud applications, common parametric statistical tools still cannot be applied. We are currently working on a separate testing methodology for means and percentiles that can further reduce test costs.

8 RELATED WORK

A large body of work in performance testing focused on the generation and prioritization of test inputs [8, 13, 18, 60]. Burnim et al. designed an input generator that can produce worst-performing test inputs for any input sizes [12]. Zhang et al. proposed a methodology to generate test inputs given an input size and diversity based on symbolic execution [64]. Chen et al. extended this idea to employ probabilistic symbolic execution to generate test inputs with frequencies so that a performance distribution can be constructed [14]. PerfLearner can help test input generation by finding important parameters from bug reports [33]. Perfranker is a test input prioritization mechanism for performance regression testing [48]. There is also work on detecting performance bugs by analyzing source code, application behavior and traces [9, 15, 24, 32, 36, 40, 41, 50, 61].

These test input generation, prioritization and performance debugging studies are orthogonal to our work, as our work focuses on determining the accurate performance of a test input.

Several studies documented the importance of repeatedly executing a test input for performance testing [48, 53]. However, these studies did not provide means to properly determine the number of test runs required to get reliable performance with low testing cost. They either used extensively long test runs [48] or an arbitrary number of runs [53]. The most related work on performance test stopping condition is probably the study done by AlGhmadi et al. [3] and the CI-based methodology [39, 46]. However, our comparison experiments in Section 5.4 showed that these methodologies could not provide accurate performance testing results on the cloud. Guo et al. also employed bootstrap in their work for building performance models and cross-validation, instead of testing [30].

There is also research on predicting cloud application’s performance. The mostly related prediction work is proposed by Luke et al. to predict the performance distributions for applications with multiple steps [10]. Our comparison experiments in Section 5.4 showed that this work could provide predictions. Hsu et al. investigated predicting the best VM configuration using Bayesian Optimization with low-level metrics [35]. However, this work did not aim at predicting the actual performance of cloud applications. Wang et al. predicted the performance of CPU and memory-intensive applications [62]. PARIS is a model that could predict the performance of a cloud application on a VM configuration [63]. However, PARIS may have up to 50% (RMSE) error [63]. Gambi et al. employed the Kriging method to predict the average performance of a cloud application under different workloads and VMs [25]. It is worth noting that the existence of predictive work cannot eliminate the necessity of measurement-based performance testing approaches such as PT4Cloud. As shown in Section 5.4 and in prior work, measurement-based performance testing is still required to provide complete training set and accurate ground truths for predictive approaches [56].

9 CONCLUSION

Performance testing on clouds to obtain accurate testing results is extremely challenging due to cloud performance uncertainty, the inability to control cloud execution environments and the testing cost. In this paper, we present a cloud performance testing methodology, PT4Cloud. By employing non-parametric statistical tools of likelihood theory and bootstrapping, PT4Cloud can provide reliable stopping conditions to obtain highly accurate performance results as performance distributions with confidence bands. The evaluation results show that, with two test reduction techniques, PT4Cloud’s performance results are on average 91% similar with testing results from extensive test runs with 90.1% fewer tests compared to these extensive runs. This is considerably better than the state of the art as evidenced in our empirical comparisons.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation under grants CCF-1617390 and CCF-1618310. 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. We would also like to thank James Skripchuk, Tianyi Liu and Xin Nie for their valuable inputs.


