Testing Cloud Applications under Cloud-Uncertainty Performance Effects

Wei Wang*, Ningjing Tian†, Sunzhou Huang*, Sen He* , Abhijeet Srivastava† Mary Lou Soffa‡ and Lori Pollock†
*Department of Computer Science, University of Texas at San Antonio
†Department of Computer and Information Sciences, University of Delaware
‡Department of Computer Science, University of Virginia
{wei.wang, sunzhou.huang, sen.he}@utsa.edu, {ningjing, abhisri, pollock}@udel.edu, soffa@virginia.edu

***Abstract*—The paradigm shift of deploying applications to the cloud has introduced both opportunities and challenges. Although clouds use elasticity to scale resource usage at runtime to help meet an application's performance requirements, developers are still challenged by unpredictable performance, little control of execution environment, and differences among cloud service providers, all while being charged for their cloud usages. Application performance stability is particularly affected by multi-tenancy in which the hardware is shared among varying applications and virtual machines. Developers porting their applications need to meet performance requirements, but testing on the cloud under the effects of performance uncertainty is difficult and expensive, due to high cloud usage costs.**

This paper presents a first approach to testing an application with typical inputs for how its performance will be affected by performance uncertainty, without incurring undue costs of brute-force testing in the cloud. We specify cloud uncertainty testing criteria, design a test-based strategy to characterize the black-box cloud's performance distributions using these testing criteria, and support execution of tests to characterize the resource usage and cloud baseline performance of the application to be deployed. Importantly, we developed a smart test oracle that estimates the application's performance with certain confidence levels using the above characterization test results and determines whether it will meet its performance requirements. We evaluated our testing approach on both the Chameleon cloud and Amazon web services; results indicate that this testing strategy shows promise as a cost-effective approach to test for performance effects of cloud uncertainty when porting an application to the cloud.

Keywords – Cloud Applications; Software Testing on Cloud; Cloud Performance Uncertainty; Bootstrapping;

I. Introduction

A recent large survey reported that 95% of 1,060 surveyed technical professionals have adopted cloud computing in their organizations, with the majority (70%) of them employed an Infrastructure-as-a-Service (IaaS) cloud, such as Amazon EC2 and Microsoft Azure [1]. The main uses today include High Performance Computing (HPC), data analytics, web applications, and development and testing environments.

This paradigm shift of applications to a cloud has introduced both opportunities and challenges. Not only does the cloud application need to be tested for accuracy, but also for performance as cloud applications typically have both performance and cost requirements [2], [3], [4], [5]. Satisfying these requirements on public clouds is very challenging because the performance of applications running on a cloud suffers from considerable uncertainty. For example, Figure 1 shows the one week (3876 trials) performance trace of a PARSEC benchmark, streamcluster, on the research cloud Chameleon [6], [7]. The best performance is about 30% faster than the worst performance. This wide performance range, in turn, translates into highly uncertain cloud performance and usage cost.

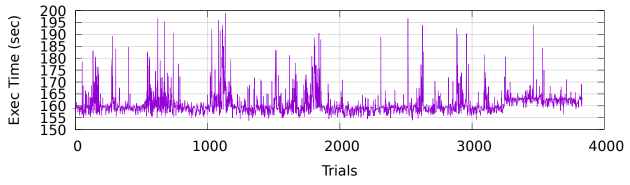


Fig. 1: The performance uncertainty of PARSEC benchmark streamcluster on Chameleon Cloud

To ensure that their applications meet performance requirements in the cloud, software engineers need to determine their performance and its fluctuation within the cloud. Traditionally, determining an application's performance requires performance testing, in which test cases are executed to cover the factors that may impact performance [8]. Under a cloud environment, additional factors must be considered, such as virtual machine (VM) scheduling and multi-tenancy (in which the hardware is shared among varying applications and VMs) [9], [10]. These factors randomly affect performance, causing fluctuations and uncertainties.

Unfortunately, cloud services are provided to the users as black boxes, where users have limited control over cloud execution environments. If traditional performance testing is employed in a cloud, extremely long tests must be conducted to extensively cover the random impacts of performance-affecting factors, which can be cost prohibitive in practice. For the example, for the benchmark in Figure 1, it requires roughly ten days of executions (about 3000 trials) to get stable average, maximum and minimum execution times.

Cloud elasticity, where users can increase resource allocation to improve performance, may be employed to improve performance satisfaction [11]. However, providing performance requirement satisfaction remains an open research question despite elasticity and other cloud management techniques [12], [13], [14]. In fact, for better results, many elasticity policies require prior knowledge of application performance [15], [16]. Consequently, effective cloud performance testing is a valuable enhancement to elasticity and many other cloud management techniques.

This paper describes a novel testing approach to help software engineers meet their performance and cost requirements when porting their applications to public clouds; that is, we present an approach to testing an application with typical inputs for how its performance will be affected by cloud uncertainty, without incurring undue costs of brute-force testing in the cloud. We address the problem of determining a cloud application's performance from an application developer's perspective: a black-box cloud that operates with usage cost. Specifically, we address the following challenges in testing applications amidst cloud uncertainty effects:

1) Performance result accuracy. Due to the uncertainty effects, the performance of a cloud application usually follows a distribution with a wide range of possible performance values. Therefore, performance statistics, such as mean execution time, should be reported with confidence intervals (CI). However, the performance distributions of cloud applications usually do not follow known theoretical distributions, making it very challenging to establish reliable confidence intervals. Without a reliable confidence interval, it is nearly impossible to accurately determine whether a cloud application can meet its performance goals.
2) High testing cost. When performance tests of an application are conducted on a cloud, they also incur cloud usage cost. A testing technique should fully cover the impacts of various performance factors. However, a poorly designed technique may have high cloud usage cost due to the black-box issue discussed above.
3) Cloud service generality. Different cloud service providers have different VM configurations and hardware. These differences may cause the behavior of an application to vary with cloud services. A truly practical cloud testing technique should work properly on any cloud service despite these differences.

To the best of our knowledge, there has been no prior research that guides software engineers through these challenges. Our approach, which we call *CAPT* (Cloud Application Performance Tester), consists of three major components. First, a test-based approach to characterize a cloud's per-resource performance distributions given performance uncertainty testing criteria, using common hardware resources that are generally available in all cloud services. Second, a reduced cost approach to test case execution to characterize an application in terms of its resource usage profile and baseline performance on the cloud. Third, a smart test oracle that estimates the application's performance on the target cloud using the test results from the first two components and determines whether performance requirements can be met. This smart oracle employs a resource-based performance model and a statistical approach called bootstrap to provide highly accurate estimations with developer-selected confidence levels [17]. The testing cost mainly comes from the first component, whose test results can be reused over multiple application inputs and multiple applications, reducing the overall testing cost.

We conducted a proof-of-concept implementation of *CAPT* and evaluated it on two public clouds with configurations using one VM of multiple VM types. The results show that *CAPT* can accurately estimate the performance of various application with an average error of 4.9%. *CAPT* can also reduce testing cost by 66.9% on average. These results strongly suggest that *CAPT* considerably reduces testing cost by conducting limited tests of the target application on a cloud and employing a smart test oracle.

This paper presents the following novel contributions:

- The definition of testing coverage criteria for cloud performance uncertainty,
- A novel performance testing approach, *CAPT*, that employs cloud/application characterizations and a smart oracle to determine whether an application can meet its performance requirements on public clouds with high accuracy and low testing cost.
- The use of bootstrapping in performance testing to process the performance results of an application with unknown theoretical performance distribution, and
- A feasibility study and evaluation of *CAPT* for applications executing with cost and performance requirements on two IaaS clouds.

The rest of this paper is organized as follows: Section II provides a use case scenario of *CAPT*. Section IV describes the *CAPT* approach in detail, followed by an example presented in Section V. Section VI describes the experimental evaluation; Section VII discusses related work, and conclusions are presented in Section VIII.

II. Usage Scenario

Assume that Cloud X is a public cloud that is available for users to run their applications. Like most clouds, Cloud X offers various types of VMs, such as small, medium and large instances. Cloud X has dynamic scheduling policies and allows for multi-tenancy. A software engineer can configure the VM sizes and numbers, but otherwise, the operation of Cloud X is not explicitly made available to a given user, and Cloud X is shared among many users, sometimes with more users during some days and times than others.

Now assume that an organization wants to port applications (e.g., HPC, data analysis, simulation, etc.) which have performance requirements (e.g., execution time, response time) and cost requirements to Cloud X. For a given application, a software engineer, Eve, needs to determine what configurations of VM meet the organization's performance requirements 95% of the time with the least cost. She provides her application to be ported (App_{port}), its test suite and an initial specification to use a large VM, because it is the most powerful configuration. The *CAPT* approach starts by running the test suite through

App_{port} on one of her local machines to determine App_{port}'s resource usage profile, for example, how App_{port} uses CPU and memory bandwidth. *CAPT* then runs App_{port} on Cloud X using a large VM for a few trials to get App_{port} cloud baseline performance, which is computed as the average execution time of the few trials on Cloud X.

Using *CAPT*'s pre-determined performance distributions of Cloud X, App_{port}'s resource profile and baseline performance, *CAPT*'s smart oracle estimates the 95th percentile performance of App_{port} under the given large VM configuration with a confidence level selected by Eve and determines whether the large VM configuration meets Eve's performance requirements. Eve is also provided information that will help in determining the cost. Depending on the estimates, Eve can change the VM configuration and repeat the process until the "best" configuration is determined to meet the performance requirement with the lowest cost.

III. Brief Introduction to Bootstrap

As discussed in Section I, the performance of an application on a cloud is distributed across a wide range of possible values. Therefore, any testing to examine performance is equivalent to sampling the performance population. Thus, it is more accurate to report performance results from these tests using statistical estimations with confidence intervals (CI). However, because the application performance distribution does not always follow known theoretical distributions, it becomes quite challenging to establish reliable CIs. To overcome this challenge, we employed non-parametric bootstrap, which allows establishing reliable CIs without the need to know theoretical distributions [17].

Here, we demonstrate the basic idea of bootstrapping with an example. Suppose a set of tests is executed with an application on a cloud to acquire a sample which has N execution times. With this sample, the population's mean execution time, μ, can be estimated using the sample average, t. Bootstrap is then employed to establish a CI of μ. To obtain a bootstrap sample, N execution times are randomly drawn from the original sample with replacement. This bootstrap sample is essentially a resample of the original sample, treating the original sample as its population. With the bootstrap sample, a bootstrap average is computed. Repeating the bootstrap sampling for R times, we can then acquire R bootstrap averages, which produces a histogram of bootstrap averages. By the definition of CI, the endpoints of the area that cover 90% of this histogram provide a CI of the original sample average t with 90% confidence level, if R is sufficiently large (usually $R \geq 1000$). With large N and R, this CI can be used as a reliable estimate of the CI of the original population mean μ with 90% confidence level.

Similar processes can be applied to other statistical estimates, such as median or quantiles. The confidence level can also be adjusted based on needs by choosing endpoints that cover different sizes of the area of the histogram. Intuitively, bootstrap works correctly assuming re-sampling on the original sample behaves similarly as the original sampling on the population. This assumption is usually true when N and R are sufficiently large and the original sample does not contain missing data and internal dependencies [18].

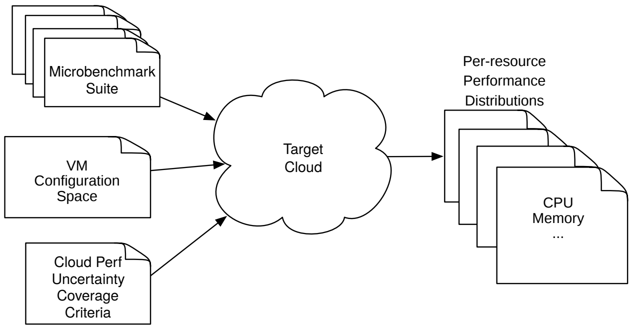

Microbenchmark Suite
VM Configuration Space
Cloud Perf Uncertainty Coverage Criteria
Target Cloud
Per-resource Performance Distributions
CPU
Memory
...

Fig. 2: Test-based characterization of the cloud as black box

Bootstrapping is employed by our smart oracle when it needs to establish a CI for a quantile using the performance distributions collected from the cloud characterization step.

IV. The *CAPT* Methodology

This section describes the three major components of the *CAPT* approach:

1) A test-based characterization to determine the black-box *cloud's performance distributions* under the cloud testing criteria. The characterization tests are executed for each resource type, such as CPU and memory, to acquire the performance distribution of individual resource types. The test coverage criteria usually cover the performance factors that are of concern to software engineers, including multi-tenancy and VM scheduling.
2) The execution of a small set of performance tests to characterize the *resource usage profile* of the target application and the *baseline performance* of the application executing on a cloud. Resource profiling tests are executed outside the cloud (e.g. on a local machine) to eliminate cloud usage cost. Baseline performance tests are executed on the cloud and are designed to include only a small number of executions to minimize cost.
3) A smart test oracle that determines whether a given application with a certain VM configuration can meet its performance requirements. With a resource-based performance model, the smart oracle combines cloud performance distributions, application resource profile and baseline performance, to estimate performance statistics (e.g., mean execution time and 95th percentile execution time) of the target application. An estimated performance statistic is expressed as a confidence interval (CI) with a confidence level (CL) selected by software engineers. With the estimated performance statistics, the smart oracle can then determine whether the performance requirement can be met with certain CL.

A. Characterizing the Performance Distribution of a Cloud

Figure 2 depicts our overall approach to uncovering the performance characteristics of the cloud, which is a black box to the software engineer porting an application to it. The cloud characterization process takes three inputs: *a micro-benchmark suite*, *a VM configuration space*, and *desired cloud performance testing criteria*.

The **micro-benchmark suite** consists of micro-benchmarks that are able to reveal the performance with respect to each common hardware resources in the cloud. For example, we use a benchmark that is purely computation-bound to determine the performance distribution of the CPU and a benchmark that is purely memory-bound to determine the memory performance distribution. We choose to characterize a cloud based on resources for three reasons. First, application performance fluctuations are fundamentally caused by fluctuations in hardware resource usages. Second, different resources have different impacts on performance fluctuations. Third, a characterization based on common hardware resources makes this characterization step applicable to any cloud services. Although we focus on CPU and memory in this paper, the idea of per-resource characterization can be extended to other common types of resources, such as storage and network, when testing applications with heavy usage of these resources.

The **VM configuration space** denotes the types and numbers of VM instances on which the application executes, as specified by the software developer. Cloud services usually provide various types of VMs, with certain CPU count and memory capacity. In this paper, we focus on the configurations with one VM of different VM types and leave configurations with varying numbers of VMs for future work.

The **testing coverage criteria** target the factors that cause cloud performance uncertainty. In this paper, we focus on the coverage of multi-tenancy and VM scheduling. Each factor also has its unique coverage criteria associated with it, which is explained as follows:

- *Coverage criteria for multi-tenancy.* Multi-tenancy is the primary cause of performance uncertainty. Sharing hardware resources causes resource contention, which degrades the performance of cloud applications. The actual performance degradation fluctuates with the types and count of contending applications. Consequently, covering multi-tenancy requires covering the potential types and count of contending applications. Moreover, different types of cloud applications have different probabilities of executing in the cloud. Therefore, multi-tenancy coverage criteria should be covering the potential types and count of cloud applications with their associated probability. However, because a cloud is provided as a black-box, it is impossible to control the types of applications executing together.

  Fortunately, because the impact of multi-tenancy usually exhibits seasonality (e.g., application types and count vary daily/weekly/monthly), multi-tenancy coverage can be translated into covering the types and count of cloud applications within a seasonal period or cycle [19]. Whether to cover a daily, weekly, monthly or a custom cycle, depends on the actual behavior of the cloud system and testing budget. Ideally, all potential types of applications should be covered. However, as an initial attempt to address the cloud performance testing problem, we strive to cover all application types, instead of providing a theoretical guarantee of complete coverage. This theoretical guarantee requires in-depth statistical analysis which is beyond the scope of this paper and approach.
- *Coverage criteria for VM scheduling.* The CPU scheduling policy of virtual machines may also cause performance fluctuations [10]. The impact of scheduling policy is extremely visible if the underlying hardware is Non-Uniform Memory Access (NUMA) architecture where thread-to-processor placement may produce considerable remote memory access overhead. The coverage criteria for VM scheduling should be covering the potential thread-to-processor placements.

In traditional performance testing, test cases should be executed with the application-to-port to meet the above coverage criteria. However, as stated previously, to reduce testing cost, we partition the test cases into two sets, with each set characterizing either the cloud or application. The set of test cases for cloud characterization is designed to meet the above testing coverage criteria.

To satisfy the *coverage criteria for multi-tenancy*, the test cases are designed to run micro-benchmarks repeatedly for one multi-tenancy (daily/weekly/monthly or custom) cycle with the VM types and VM count specified by the VM configuration. To obtain performance distributions for both CPU and memory, one test case is executed with a CPU micro-benchmark while one test case uses a memory micro-benchmark. After execution, a test case will give the performance distribution of one resource type for one cycle. Note that, similar to other time series data, multi-tenancy impacts might include random bursts, besides seasonality. To mitigate the effect of these random bursts, software engineers may want to conduct additional test cases for additional cycles. Combining data from multiple cycles provide more accurate performance distributions which are less susceptible to the noise of random bursts.

To satisfy the *coverage criteria for VM scheduling*, several tests are needed to achieve a high probability of covering all thread-to-processor placements. With the knowledge of the core count of the VM and the processor count of the server, it is easy to compute the number of possible thread-to-processor placements. Based on this number, software engineers can easily determine how many times a micro-benchmark should be executed to achieve a high probability of covering all placements. That is, a test case can be viewed as running a micro-benchmark repeatedly until all placements are covered with high probability. Note that, because test cases covering multi-tenancy usually provide enough micro-benchmark runs to cover VM scheduling, and because the black-box cloud does not offer methods to separate the impacts of multi-tenancy

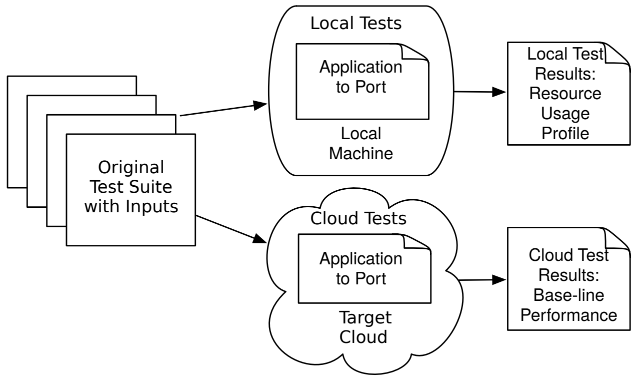

Local Tests
Application to Port
Local Machine
Local Test Results: Resource Usage Profile
Original Test Suite with Inputs
Cloud Tests
Application to Port
Target Cloud
Cloud Test Results: Base-line Performance

Fig. 3: Tests to characterize application-to-port.

and VM scheduling, it is usually not necessary to separately run test cases for VM scheduling coverage. The results from multi-tenancy test cases can provide performance distributions covering the combined impacts of both multi-tenancy and VM scheduling, meeting both testing coverage criteria.

In summary, a test is defined as executing a micro-benchmark repeatedly for one multi-tenancy cycle, using VM configurations specified by software engineers, to meet the coverage criteria of multi-tenancy and VM scheduling. Each additional micro-benchmark requires an additional test. The results of the tests include the performance (e.g., execution times) of the micro-benchmarks, which give the per-resource-type performance distributions of the cloud. Note that, for each VM configuration, the cloud characterization needs to be done only once. But its results can be reused for testing multiple application inputs or multiple applications, thus reducing cost.

B. Determining an Application's Resource Usage Profile

Similar to the cloud characterization step, our application resource usage profile contains an application's usage for both CPU and memory. Fig. 3 shows the *CAPT* approach to determining an application's resource usage profile, with inputs that consist of the application to be ported and real user workloads for the application. Note that, similar to traditional performance testing, we assume that software engineers provide testing inputs that resemble real inputs.

A resource usage profile is comprised of the processor cycles that an application spends in CPU computation and memory accesses. To reduce the testing cost, tests to profile the application's resource usage are executed on a local machine instead of the cloud, using hardware performance monitoring units (PMU) [20]. Note that, if the resource usage profiling is executed on the cloud, the results may be skewed due to the cloud's performance uncertainty, and not all clouds support hardware PMU monitoring needed for profiling.

We denote an application's resource usage running on a local machine using App_{res}. More specifically,

- App_{cpu} denotes the percentage of the total cycles that are spent on CPU computations
- App_{mem} denotes the percentage of the total cycles that are spent on memory accesses

To determine the resource usage profile, each test is to execute an application on a local machine with its typical input. During the test case execution, PMU readings are also collected. The total cycles consumed by an application, denoted by $Cycles_{total}$, is obtained with one PMU event. For instance, on Intel platforms, the event is typically named "UNHALTED_CORE_CYCLES." This event includes both CPU computation and memory access cycles. Consequently, CPU computation cycles can be computed by subtracting memory cycles from the total cycles. Because memory accesses are usually slow and would block the execution of the processor [20], they can be estimated using stalled CPU cycles. For instance, the PMU event to read these stalls on Intel processors is called "RESOURCE_STALLS." Some memory accesses can overlap with computation and do not stall execution. Since these memory accesses can overlap with computation, their fluctuation has limited impact on overall execution time. Therefore, we do not consider the cycles of overlap-able memory accesses as part of memory access cycles. Let $Cycles_{mem}$ be the memory accesses/stall cycles. Once total cycles and memory cycles are acquired, the percentage of CPU and memory cycles can be computed as:

$$\begin{aligned} App_{cpu} &= \frac{Cycles_{total} - Cycles_{mem}}{Cycles_{total}} \times 100 \\ App_{mem} &= \frac{Cycles_{mem}}{Cycles_{total}} \times 100 \end{aligned} \tag{1}$$

C. Establishing an Application's Cloud Baseline Performance

As Figure 3 shows, application characterization also includes obtaining the baseline performance of the application on the cloud, which is later used by the smart oracle to estimate the real performance of the application on cloud.

A test is an application executing on the cloud with its typical input. The test is repeatedly executed for a few times to reduce cost, and the average performance (e.g., average execution time) is computed as the application's baseline performance. Let $App_{baseline}$ denote this baseline performance. The number of test executions is determined by software engineers based on their testing budget. More executions may increase performance estimation accuracy with increased cost. Note that, for each new input, an application resource profile and baseline performance should be collected.

D. The Smart Oracle

Performance Estimation. Figure 4 depicts how the smart oracle estimates the performance of the application-to-port on a cloud. The smart oracle takes four inputs: the *cloud performance distributions*, the *application resource usage profile* and *application baseline performance* from previous steps, as well as the *desired performance statistic and confidence level* from the software engineer. The *performance statistic* is the performance attribute that the software engineer wants to know. For instance, this statistic may be any quantile of the execution time. The *confidence level* (CL) is used by the smart oracle to establish the CI for the performance statistic.

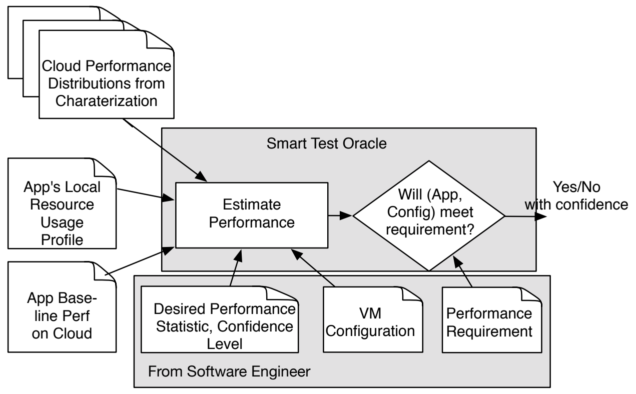

Cloud Performance Distributions from Charaterization
Smart Test Oracle
App's Local Resource Usage Profile
Estimate Performance
Will (App, Config) meet requirement?
Yes/No with confidence
App Base-line Perf on Cloud
Desired Performance Statistic, Confidence Level
VM Configuration
Performance Requirement
From Software Engineer

Fig. 4: Smart Test Oracle

For example, if a software engineer wants to know whether an application will meet the performance goal 95% of the time with 90% confidence, the performance statistic should be 95th percentile execution time, and CL should be 90%.

Knowing the desired performance statistic and CL, the smart oracle first processes the cloud performance distributions (from cloud characterization) to obtain the CI of the performance statistic for CPU and memory resources using bootstrap. For instance, with the above example statistic and CL, the smart oracle processes the performance distributions to obtain 90% CI of the 95th percentile execution times of the CPU and memory micro-benchmarks.

As the goal of the smart oracle is to determine whether the performance goal can be met, we only use the upper bound of the CI.[1] Let this upper bound for CPU and memory be $Cloud_{cpu}^{stat}$ and $Cloud_{mem}^{stat}$. Additionally, let the average execution times of CPU and memory micro-benchmarks be $Cloud_{cpu}^{avg}$ and $Cloud_{mem}^{avg}$. The smart oracle then computes the fluctuation ranges of the performance statistic relative to average performance for CPU and memory using Equation (2).

$$\begin{aligned} Cloud_{cpu}^{range} &= \frac{|Cloud_{cpu}^{stat} - Cloud_{cpu}^{avg}|}{Cloud_{cpu}^{avg}} \\ Cloud_{mem}^{range} &= \frac{|Cloud_{mem}^{stat} - Cloud_{mem}^{avg}|}{Cloud_{mem}^{avg}} \end{aligned} \quad (2)$$

To estimate the corresponding performance statistic for the application with the same CL, the smart oracle applies the respective fluctuation range to the CPU and memory portions of its baseline performance, using the performance model of Equation (3).

$$\begin{aligned} App_{stat} =& App_{baseline} \times [App_{cpu} \times (1 + Cloud_{cpu}^{range}) + \\ & App_{mem} \times (1 + Cloud_{mem}^{range})] \end{aligned} \quad (3)$$

By comparing the estimated performance statistic with the performance goal, the smart oracle can then determine if the performance goal can be met with the same CL. For instance, with the above example statistic and CL, if the estimated 95th percentile execution time is smaller than the performance goal, the smart oracle determines that the performance goal can be met with at most 90% confidence.

Result Interpretation. CI's interpretation is confusing in classic (frequentist) statistics. However, another benefit of non-parastat bootstrap is that it can be viewed as an approximation of Bayesian model [21]. Therefore, we can interpret the CIs from bootstrap similarly as credible intervals. That is, when the smart oracle determines that the performance goal can be met with X confidence level, it can be approximately interpreted as: with at most X probability, the performance goal can be met. Note that, because the performance is estimated using a performance model in Equation (3), the estimation has errors due to the model's inaccuracy. Therefore, X is the upper bound of the confidence/probability. The lower bound of the confidence/probability is determined by the estimation's error. The experiment results in Section VI show that the maximum error is 14% and the average error 5%. Therefore, the experimental lower bound of the confidence/probability is approximately $X - 14\%$ at most and $X - 5\%$ on average.

V. Example

Continuing with the example from Section II, assume that Eve wants to port the application *ferret* from PARSEC onto the Chameleon cloud [7]. She wants to use a single large VM instance with a performance requirement of 120 secs as the maximum execution time for 95% of the time. The application will be executed with inputs similar to PARSEC's "native" input, assuming Chameleon cloud's per-resource performance distributions have already been characterized.

Eve considers whether a large VM instance is a good choice. She executes *ferret* on a local machine. Table V indicates that the resource usage profile for *ferret*, which spends 76% cycles on CPU (App_{cpu}) and 24% cycles on memory (App_{mem}). Eve also executes *ferret* on the Chameleon cloud with a large VM configuration for 100 iterations, and finds the average baseline execution time is 99 seconds ($App_{baseline}$).

Eve then inputs *ferret*'s resource profile and baseline execution time into the smart oracle. Eve also specifies that she wants to test for the 95th percentile execution time with 90% confidence level. Based on Eve's performance requirements, *CAPT* provides the feedback to Eve that the estimated 95th percentile execution time of *ferret* is less than 104.7 seconds with at most 90% confidence, and thus the performance requirement can be met with a large VM instance. Since the cloud charges users by resource usage, she also wants to know the associated costs. As a large instance meets Eve's performance requirement, Eve may consider using a large instance and calculates the budget using the large instance's hourly rate. She could also try a medium VM to see if it meets the performance requirement.

Assuming an hourly rate for a large VM instance on the Chameleon cloud is \$0.1865 (based on AWS t2.xlarge pricing), the process of collecting 100 sample executions of *ferret* on Chameleon for determining the baseline performance

[1]Note that, although the lower bound of the CI is not used in *CAPT*, it could be useful to analyze other performance properties in the future.

costs $\lceil 2.75 \rceil hours \times 0.1865 = \$0.56$. If Eve instead were to run *ferret* on a large VM instance for two weeks instead of using *CAPT*, the cost to determine whether the large instance will meet the performance requirements would be $336\ hours \times 0.1865 = \$62.67$. Thus, Eve saves money using *CAPT* to determine whether the large VM instance is adequate for her performance requirements. It is also worth noting that, by using *CAPT*, Eve can get an estimation within 3 hours, compared to two weeks wait without *CAPT*.

VI. Evaluation

Our evaluation is designed to answer the following questions: 1) RQ1-Effectiveness: Are *CAPT*'s performance estimations accurate compared to the results acquired from extensive performance testing? 2) RQ2-Cost Benefit: What is the relative cost of the *CAPT* approach versus determining the performance of an application directly in the cloud through extensive performance testing?

A. Experimental Methodology

To answer the above questions, we used *CAPT* to conduct testing to estimate the 95th percentile execution times of several applications with 90% confidence level on two public clouds. We also conducted extensive performance tests for these applications on the same clouds to acquire the experimental 95th percentile execution times. We refer to this extensive performance testing as the *ground truth tests* and their results as the *ground truth*. We then compared *CAPT*'s results and testing costs with the ground truth results and ground truth testing cost to evaluate *CAPT*'s effectiveness and cost benefit, respectively.

Micro-benchmarks suite. We selected two well known micro-benchmarks, where each benchmark heavily utilizes only one type of resource. These benchmarks are used to obtain the performance fluctuation distributions of each type of resource. Although the idea of *CAPT*'s cloud characterization can be extended to most common types of resources, as a proof of concept, we focus on the CPU and memory resources in this paper. Table I describes the micro-benchmarks and their corresponding resources.

Micro-benchmark	Type	Origin
Sysbench-CPU	CPU	Sysbench [22]
STREAM	memory	STREAM [23], [24]

TABLE I: Micro-benchmarks for cloud characterization

Subject Applications. We selected seven applications from the PARSEC benchmark suite as the subjects of study for evaluation [6]. These applications primarily utilize CPU and memory for computation, with limited usage of disk and network. Table II lists these applications with their primary domains. These subjects cover a variety of domains that may be ported to and benefit from cloud systems, including data mining, financial analysis, simulation and system optimizations. The "native" input sets from PARSEC, which are designed for evaluations on real systems, are used as inputs to all of our subject applications.

Application	Domain
blackscholes (BS)	Financial analysis
swaption (SW)	Financial analysis
streamcluster (SC)	Data mining
ferret (FE)	Content-based similarity search server
bodytrack (BT)	Body tracking of a person
canneal (CN)	Chip design optimization
fluidanimate (FL)	Fluid dynamics simulation

TABLE II: Subject applications

Cloud Infrastructure and VM configurations. Our evaluation was conducted on the research cloud Chameleon and Amazon Web Services (AWS) [7]. We selected the Chameleon cloud because it is free for researchers and a well maintained cloud infrastructure. Chameleon cloud offers configurations that are similar to the popular commercial clouds such as AWS and Microsoft Azure. AWS was selected because it is one of the leading service providers in cloud computing. The applications running on Chameleon and AWS are affected by cloud performance uncertainty factors, including virtual machine (VM) scheduling and multi-tenancy [9], [19]. In particular, through our correspondence with a Chameleon maintainer, we confirmed that multi-tenancy does exist in Chameleon. Figure 1 also shows that application performance indeed fluctuates on Chameleon. Both Chameleon and AWS offer several types of VM instances. For Chameleon, we used all its VM types in this evaluation, except the "Tiny" instances which are too small for our subject applications. While AWS offers more choices of VMs, we chose to use three general purpose instances that are also not too expensive for our large scale evaluation (our evaluation on AWS costs about $2000 in total) The instance types used in this evaluation are listed in Table III and Table IV.

As stated in Section IV, we focus on the configurations with a single VM. Chameleon instances are created in the data center at the University of Chicago, while the AWS instances are created in the AWS service region of US East (Ohio). When executing micro-benchmarks and subject applications on VMs with multiple CPU cores, all micro-benchmarks and applications are configured to execute with multiple threads to utilize all available cores.

Type	Core Count	Memory(GB)	Pricing
Small (Sm)	1	2	free
Medium (Md)	2	4	free
Large (Lg)	4	8	free

TABLE III: VM instance types of Chameleon in this study

Type	Core Count	Memory(GB)	Pricing
t2.small (t2s)	1	2	$0.023/hr
t2.medium (t2m)	2	4	$0.0464/hr
t2.xlarge (t2l)	4	16	$0.1856/hr

TABLE IV: VM instance types of AWS in this study

Coverage Criteria. In this evaluation, we chose the multi-tenancy coverage criteria to cover both one daily cycle and one weekly cycle, because they are the most common cycles reported by prior work [19]. Daily and weekly cycles are also more affordable to our evaluation than monthly cycles. To cover both one daily cycle and one weekly cycle, a cloud characterization test case must be executed for a week. However, due to the bursty issue discussed in Section IV-A, additional test cases were executed for another week to mitigate the impact of random bursts. That is, for both cloud characterization tests and ground truth tests, a micro-benchmark or an application was executed repeatedly for two weeks. Although executing for more weeks may provide more accurate performance data, it is too expensive to conduct. Recall that, as discussed in Section IV-A, the tests for covering multi-tenancy coverage criteria also provide coverage for VM scheduling criteria.

Performance Statistic and Confidence Level. The performance statistic used in this evaluation is the 95th percentile execution time, with the CL as 90%. Intuitively, with these two parameters, *CAPT* aims at determining if a subject application can meet its performance goal for 95% of the time with at most 90% confidence. Recall that *CAPT* estimates the upper bound of the CI of the performance statistic to determine the satisfaction of performance goals.

B. Characterizing Cloud Performance Distributions

Procedure: Given there are two micro-benchmarks and six types of VMs in total for the two clouds, a total of 24 tests of micro-benchmark executions were conducted. Performance results for the micro-benchmarks are collected to construct per-resource performance distributions.

Results: Figure 5 gives the distributions (with histograms) of the performance of CPU and memory micro-benchmarks. Due to space limitations, only the medium VM types from each cloud are shown. As Figure 5 shows, the performance distributions of both clouds have large fluctuation ranges and vary considerably depending on the resource type and cloud. With statistic software, we also found that the distributions in Figure 5 do not follow common theoretical distributions. These characteristics make it very difficult to process the distribution to acquire important performance statistics.

C. Determining Applications' Resource Usage Profiles

Procedure: To determine the applications' resource usage profile, we executed each subject application on a local Ubuntu server with Intel processor i7-2600 and followed the approach in Section IV-B to calculate App_{cpu} and App_{mem}.

Results: Table V presents the results of the resources used by the applications. Among all applications, blackscholes and swaptions are CPU-intensive applications with little memory access time. Streamcluster streams in large amounts of data, thus it is a memory-intensive application that spends most of its time in memory accesses. The rest of the applications, including ferret, bodytrack, canneal and fluidanimate, shows a mixture of both CPU computation and memory accesses.

Application	App_{cpu}	App_{mem}
swaption	99.8%	0.2%
blackscholes	99.2%	0.8%
streamcluster	2.4%	97.6%
ferret	76%	24%
bodytrack	64%	36%
canneal	79%	21%
fluidanimate	31%	69%

TABLE V: Subject applications' resource usage profiles

D. Establishing Application Cloud Baseline Performance

Procedure: To establish application baseline performance, we executed each application for 100 times on each VM configuration, following the approach in Section IV-C. 100 runs are chosen because we found they provide a good balance between accuracy and cost for most applications.

Results: Table VI shows the baseline execution time of our applications. The rows represent each subject application, while each column represents a VM type. These results suggest that the subject applications cover a variety of applications in terms of execution times.

	Chameleon			**AWS**		
	Sm	Md	Lg	t2s	t2m	t2l
SW	228.7	118.2	61.8	2618.3	1329.8	576.8
BS	130.9	65.5	37.1	740.0	371.3	161.2
SC	513.7	303.5	159.6	2359.7	1203.2	524.0
FE	364.9	174.6	99.1	2084.4	1075.2	472.8
BT	124.0	69.8	47.0	765.6	384.9	168.4
CN	126.9	70.4	39.0	1010.0	436.5	146.5
FL	323.4	189.0	104.5	1715.3	873.3	419.9

TABLE VI: Application baseline results in seconds

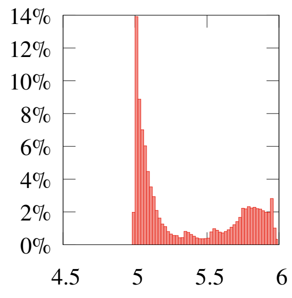
E. CAPT *Effectiveness*

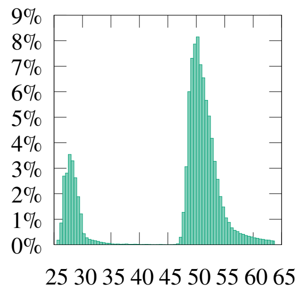
We next examined the research question: *Are* CAPT*'s performance estimations accurate compared to the results acquired from extensive performance testing?*

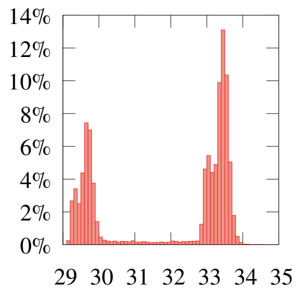
Procedure: We compare *CAPT* estimations with ground truth results. More specifically, we used *CAPT* to estimate the upper bound of the CI of the 95th percentile execution times (Per_{est}) of the subject applications with 90% confidence level for each VM configuration. We also conducted ground truth tests with these applications to acquire the experimental 95th percentile execution times as the ground truth results ($Perf_{exp}$). Then we computed the percentage error using Equation (4).

$$Error = \frac{|Perf_{est} - Perf_{exp}|}{Perf_{exp}} \times 100\% \qquad (4)$$

Results: Figure 6 presents the percentage errors for *CAPT*'s estimations on two clouds. As the figure shows, *CAPT*'s estimations have high accuracy. The average error on Chameleon is 5.1%, with a maximum error of 13.8%. The average error on AWS is 3.0%, with a maximum error of 5.9%. The average error on two clouds is 4.9%. These low errors suggest that *CAPT* can provide reliable estimations to replace traditional performance testing.


(a) Chameleon Med VM CPU Dist.


(b) Chameleon Med VM Mem Dist.


(c) AWS t2.medium VM CPU Dist.

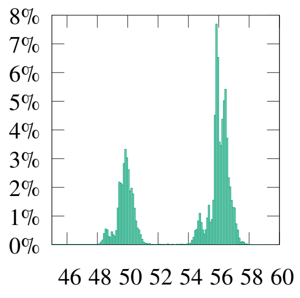
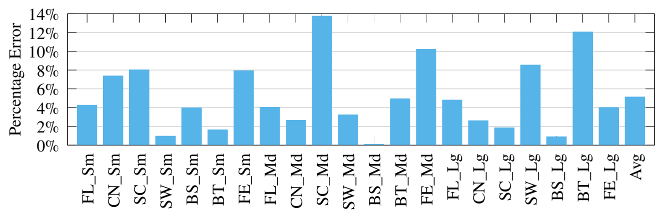

(d) AWS t2.medium VM Mem Dist.

Fig. 5: CPU and memory performance distributions based on cloud characterization tests with micro-benchmarks. X axis: execution times in seconds; Y axis: percentages of executions with particular execution times.


(a) Error on Chameleon for each application-VM pair

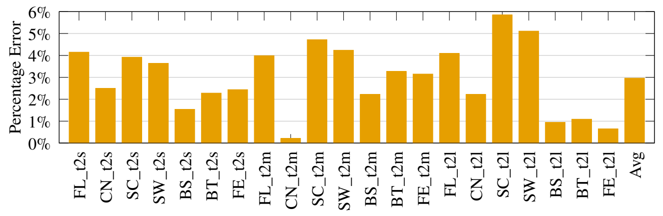

(b) Error on AWS for each application-VM pair

Fig. 6: Percentage errors of *CAPT*'s performance estimations

F. CAPT *Cost Benefits*

Lastly, we address the research question: *What is the relative cost of the* CAPT *approach versus determining the performance of an application directly in the cloud through extensive performance testing?*

Procedure: We compared the total testing cost of *CAPT* with the total testing cost of ground truth tests for all subject applications. The total testing cost of *CAPT* includes the costs of cloud characterization tests and application baseline tests. The total testing cost of ground truth tests includes the cost to execute all applications for two weeks. Because the bills from AWS do not include per-VM cost, we estimate the testing costs using the total test execution time and AWS's pricing model listed in Table IV. The sum of the computed test costs is roughly the same as the bill we received from AWS. For Chameleon, there is no charge for VM usage. However, by employing the pricing model from AWS, it is possible to determine the relative cost benefits from using *CAPT*. Note that, storage costs are not included in our calculations.

	t2.small	t2.medium	t2.xlarge
ground truth tests	\$54.10	\$109.13	\$436.53
CAPT total costs	\$22.75	\$38.51	\$137.79
CAPT cost reduction	58.0%	64.7%	68.4%

TABLE VII: Testing costs for all subject applications on AWS

Results: Table VII shows the costs of conducting tests on AWS. As the table shows, for the seven subject applications, *CAPT* can reduce testing cost by 64% on average, compared to traditional testing (ground truth tests). For a single application, if the cloud characterization cost is not considered, the cost of using *CAPT* can be 62 times smaller than traditional performance testing. For Chameleon, using a similar pricing model as AWS, the average cost reduction of *CAPT* is 69.1%, 70.2% and 70.7% for small, medium and large VMs, respectively. The average cost reduction on both clouds is 66.9%. The primary cost reduction comes from the fact *CAPT* only executed two two-week (cloud characterization) test cases for each VM configuration and reuse their results for all applications, while the extensive performance tests executed seven applications each for two weeks for each VM configuration.

Note that another benefit of *CAPT* is its time saving. Application characterization tests usually take a few hours. If the cloud is already characterized, the smart oracle can provide performance estimations instantaneously, meaning the performance estimations are available in only several hours. However, without *CAPT*, running the applications to cover all cloud uncertainty factors typically takes weeks. Therefore, *CAPT* can potentially reduce the time cost for cloud performance testing. Software engineers may even use *CAPT* as a screening tool to quickly determine the VM configurations that may meet performance goals, then conduct longer performance tests to get more accurate performance results.

G. Threats to Validity

Our evaluation focused on the cost effectiveness of the *CAPT* approach for a set of seven applications using a suite of two micro-benchmarks for the cloud characterization. The results may not generalize to other applications. To address this threat, we selected well known benchmarks for CPU and memory characterization and a variety of applications for evaluation. Our evaluation is conducted on Chameleon and AWS; results may differ on other clouds. For more accurate resource usage profiles, applications should be tested on local machines with similar processors to those in the target cloud.

We observed that there are two main sources of errors in *CAPT*. The first source is an inaccurate baseline performance. Because baseline performance is acquired as the average from only 100 runs, it may be different from the average acquired from two-week executions. We observe that longer baseline runs may mitigate the impact of this type of error. The second source is the random bursts in a cloud. The burst issue is more severe on Chameleon than AWS, causing the relatively higher errors on Chameleon. To mitigate this error, more than two weeks of tests should be conducted for both cloud characterization and ground truth tests. However, research is required to investigate the proper number of tests to obtain stable performance distributions on clouds.

Additionally, limited by space and cloud execution costs, in this proof-of-concept evaluation, we do not consider configurations with multiple VMs. Other common resources, such as storage and network, are not considered as well. Cloud performance may also be affected by other factors, such as hardware heterogeneity. We plan to extend this research in the future to consider these issues.

VII. Related Work

A number of studies in performance testing have been focusing on the proper selection of test inputs. Common approaches for input selection include static code analysis [25], [26], [27], dynamic analysis and profiling [28], [29], and probability symbolic execution [30], [31], [32]. There are also studies focusing on performance regression detection through proper test case selection and performance data analysis [33], [34], [35], [36]. Performance models are also employed to predict performance and its fluctuations due to variations in system configurations, program model and inputs [37], [38], [39]. However, performance testing on cloud presents a new problem, where the performance of a single application with a single input under a single system configuration still experiences significant fluctuations and uncertainty due to uncontrollable external issues (e.g., multi-tenancy). Therefore, by focusing on the performance uncertainty issue of the cloud, *CAPT* is complementary to these performance testing studies.

There are studies investigating testing on cloud. Gambi et al. proposed to reduce testing cost on cloud by reusing VMs and interleaving tests and VM reimaging [40]. Malik et al. presented an approach to detect performance variations in large scale system [41]. Núñez and Hierons proposed a cloud system model to facilitate cloud testing on simulators [42]. Rose et al. investigated employing cloud system for probability testing [43]. Several studies investigated automating cloud testing [44], [45], [46]. There is also research on testing auto-scaling policies and elasticity [47], [48]. Others proposed strategies for testing the performance of cloud applications under various loads [49], [50], [51], [52], [53], [54], [55], [56]. Although these studies investigated various testing issues on cloud, none of them addresses the testing challenges imposed by cloud performance uncertainty.

Many studies have observed and analyzed the performance variation and uncertainty on cloud [57], [58], [59], [60], [61], [9], [62], [63], [19]. Our work is inspired by these prior studies on cloud performance uncertainty. Elasticity may help meet performance requirements by scaling up resource usages [16], [64], [65], [66]. However, as stated in Section I, providing performance guarantee is still an open question even with elasticity and many other system management techniques [12], [13], [14]. An effective cloud testing technique such as *CAPT* may even help the design of elasticity by providing better estimations of performance [15], [16]. There is also work on predicting the performance of cloud applications [67], [68], [69], [70], [71]. These studies predicted how performance changed with input/workload sizes. They focused on applications whose performance has a strong correlation with workload sizes, such as database, map-reduce and high performance computing applications. They also required reliable training sets of performance data. However, *CAPT*, as a performance testing approach, is generic in terms of application types. Moreover, *CAPT* can also be employed to provide reliable training sets to these studies.

VIII. Conclusions and Future Work

Setting appropriate VM configurations for applications on clouds is critical for both performance and cost. However, this task can be challenging to software engineers due to the performance uncertainty of the cloud systems. In this paper, we present a proof-of-concept approach to testing an application for how its performance will be affected by cloud uncertainty, without incurring undue testing costs. We specify cloud uncertainty testing criteria, design a test-based strategy to characterize the black-box cloud's performance distributions using the testing criteria, and support execution of tests to characterize the application to be deployed as both a resource usage profile and a cloud baseline performance using sampling. We conducted a proof-of-concept implementation of *CAPT*, and our evaluation results for two public clouds show that *CAPT* can accurately estimate the performance of the considered cloud applications with 4.9% error on average and reduce the testing cost of 66.9% on average. Our future work will include increasing the number and type combinations of VMs for the configuration, more cloud service providers, as well as additional types of resources and applications.

Acknowledgments 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 Jinay Jani and Xin Nie for their valuable inputs.

References

[1] RightScale, "RightScale 2015 State of the Cloud Report," 2015.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, 2008, pp. 5–13.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software Practice & Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[4] B. Narasimhan and R. Nichols, "State of Cloud Applications and Platforms: The Cloud Adopters' View," *Computer*, vol. 44, no. 3, pp. 24–28, March 2011.

[5] P. Gupta, A. Seetharaman, and J. R. Raj, "The usage and adoption of cloud computing by small and medium businesses," *International Journal of Information Management*, vol. 33, no. 5, pp. 861 – 874, 2013.

[6] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, 2011.

[7] "A configurable experimental environment for large-scale cloud research," https://www.chameleoncloud.org/, [Online; accessed 2015-10-22].

[8] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, *Performance Testing Guidance for Web Applications: Patterns & Practices*. Redmond, WA, USA: Microsoft Press, 2007.

[9] P. Leitner and J. Cito, "Patterns in the Chaos: A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, pp. 15:1–15:23, Apr. 2016.

[10] 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 *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015.

[11] P. M. Mell and T. Grance, "SP 800-145. The NIST Definition of Cloud Computing," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

[12] T. Zhu, M. A. Kozuch, and M. Harchol-Balter, "Workloadcompactor: Reducing datacenter cost while providing tail latency slo guarantees," in *2017 ACM Symposium on Cloud Computing*, 2017.

[13] P. Janus and K. Rzadca, "Slo-aware colocation of data center tasks based on instantaneous processor requirements," in *2017 ACM Symposium on Cloud Computing*, 2017.

[14] T. Chen and R. Bahsoon, "Self-Adaptive Trade-off Decision Making for Autoscaling Cloud-Based Services," *IEEE Transactions on Services Computing*, vol. 10, no. 4, July 2017.

[15] A. Ilyushkin, A. Ali-Eldin, N. Herbst, A. V. Papadopoulos, B. Ghit, D. Epema, and A. Iosup, "An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017.

[16] M. Mao and M. Humphrey, "Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[17] B. Efron, *The Jackknife, the bootstrap and other resampling plans*. SIAM, 1982.

[18] A. C. Davison and D. V. Hinkley, *Bootstrap Methods and Their Application*. New York, NY, USA: Cambridge University Press, 2013.

[19] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011.

[20] Intel, "Intel 64 and IA-32 architecture software developer's manual," 2009.

[21] D. B. Rubin, "The bayesian bootstrap," *The Annals of Statistics*, vol. 9, no. 1, pp. 130–134, 1981.

[22] A. Kopytov, "SysBench: A System Performance Benchmark," https://github.com/akopytov/sysbench0, 2017, [Online; accessed 23-Feb-2017].

[23] J. D. McCalpin, "The STREAM Benchmark," https://www.cs.virginia.edu/stream/, [Online; accessed 2017-02-22].

[24] ——, "A survey of memory bandwidth and machine balance in current high performance computers," *IEEE TCCA Newsletter*, vol. 19, p. 25, 1995.

[25] S. Chattopadhyay, L. K. Chong, and A. Roychoudhury, "Program Performance Spectrum," in *Proceedings of the 14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, 2013.

[26] P. Zhang, S. Elbaum, and M. B. Dwyer, "Automatic generation of load tests," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011.

[27] J. Burnim, S. Juvekar, and K. Sen, "WISE: Automated Test Generation for Worst-case Complexity," in *Proceedings of the 31st International Conference on Software Engineering*, 2009.

[28] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson, "Measuring Empirical Computational Complexity," in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2007.

[29] L. Zhang, D. R. Bild, R. P. Dick, Z. M. Mao, and P. Dinda, "Panappticon: Event-based tracing to measure mobile application and platform performance," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2013.

[30] B. Chen, Y. Liu, and W. Le, "Generating Performance Distributions via Probabilistic Symbolic Execution," in *Proceedings of the 38th International Conference on Software Engineering*, 2016.

[31] A. Filieri, C. S. Pasareanu, and G. Yang, "Quantification of Software Changes Through Probabilistic Symbolic Execution (N)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015.

[32] A. Filieri, C. S. Păsăreanu, W. Visser, and J. Geldenhuys, "Statistical Symbolic Execution with Informed Sampling," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.

[33] K. C. Foo, Z. M. J. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, 2015.

[34] A. Cavalli, S. Maag, and G. Morales, "Regression and Performance Testing of an e-Learning Web Application: dotLRN," in *2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, 2007.

[35] C. Yilmaz, A. Porter, A. S. Krishna, A. M. Memon, D. C. Schmidt, A. S. Gokhale, and B. Natarajan, "Reliable Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems," *IEEE Transactions on Software Engineering*, 2007.

[36] S. Mostafa, X. Wang, and T. Xie, "Perfranker: Prioritization of performance regression tests for collection-intensive software," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017.

[37] M. Kowal, M. Tschaikowski, M. Tribastone, and I. Schaefer, "Scaling Size and Parameter Spaces in Variability-Aware Software Performance Models (T)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015.

[38] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016.

[39] H. Gautama and A. J. C. van Gemund, "Low-cost static performance prediction of parallel stochastic task compositions," *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[40] A. Gambi, A. Gorla, and A. Zeller, "O!Snap: Cost-Efficient Testing in the Cloud," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2017.

[41] H. Malik, H. Hemmati, and A. E. Hassan, "Automatic Detection of Performance Deviations in the Load Testing of Large Scale Systems," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013.

[42] A. Núñez and R. M. Hierons, "A Methodology for Validating Cloud Models using Metamorphic Testing," *Annals of Telecommunications*, vol. 70, no. 3, Apr 2015.

[43] L. M. Rose, S. Poulding, R. Feldt, and R. F. Paige, "Towards A Scalable Cloud Platform for Search-Based Probabilistic Testing," in *2013 IEEE International Conference on Software Maintenance*, 2013.

[44] Y. Magid, R. Tzoref-Brill, and M. Zalmanovici, "Coverage-based metrics for cloud adaptation," in *Proceedings of the 2Nd International Workshop on Quality-Aware DevOps*, 2016.

[45] A. Gambi, S. Kappler, J. Lampel, and A. Zeller, "CUT: Automatic Unit Testing in the Cloud," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017.

[46] B. Garcia, F. Gortazar, L. Lopez-Fernandez, M. Gallego, and M. Paris, "Webrtc testing: Challenges and practical solutions," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 36–42, 2017.

[47] A. Gambi, W. Hummer, and S. Dustdar, "Automated Testing of Cloud-based Elastic Systems with AUToCLES," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 714–717.

[48] S. Dustdar, A. Gambi, W. Krenn, and D. Nickovic, "A pattern-based formalization of cloud-based elastic systems," in *Proceedings of the 2015 IEEE/ACM 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*, 2015.

[49] Z. M. Jiang, "Automated Analysis of Load Testing Results," in *Proceedings of the 19th International Symposium on Software Testing and Analysis*, 2010.

[50] L. Riungu-Kalliosaari, O. Taipale, and K. Smolander, "Testing in the Cloud: Exploring the Practice," *Software, IEEE*, vol. 29, no. 2, pp. 46–51, 2012.

[51] N. Snellman, A. Ashraf, and I. Porres, "Towards Automatic Performance and Scalability Testing of Rich Internet Applications in the Cloud," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, 2011.

[52] P. Mohagheghi and T. Sæther, "Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project," in *Services (SERVICES), 2011 IEEE World Congress on*, 2011.

[53] J. H. Kim, S. M. Lee, D. S. Kim, and J. S. Park, "Performability Analysis of IaaS Cloud," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, 2011.

[54] D. Jayasinghe, G. Swint, S. Malkowski, J. Li, Q. Wang, J. Park, and C. Pu, "Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012.

[55] Z. Ganon and I. Zilbershtein, "Cloud-based Performance Testing of Network Management Systems," in *Computer Aided Modeling and Design of Communication Links and Networks, 2009. CAMAD '09. IEEE 14th International Workshop on*, 2009.

[56] "Elastest: an elastic platform for testing complex distributed large software systems." [Online]. Available: http://elastest.eu/

[57] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *VLDB Endownent*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[58] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," *IEEE Transcations on Parallel Distributed System*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[59] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, "Performance Evaluation of Amazon EC2 for NASA HPC Applications," in *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, 2012.

[60] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, 2012.

[61] A. Gandhi, P. Dube, A. Karve, A. Kochut, and H. Ellanti, "The Unobservability Problem in Clouds," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, 2015.

[62] D. Shue, M. J. Freedman, and A. Shaikh, "Performance Isolation and Fairness for Multi-tenant Cloud Storage," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, 2012.

[63] Z. Ou, H. Zhuang, A. Lukyanenko, J. Nurminen, P. Hui, V. Mazalov, and A. Yla-Jaaski, "Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds," *Cloud Computing, IEEE Transactions on*, 2013.

[64] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010.

[65] M. Malawski, G. Juve, E. Deelman, and J. 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.

[66] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven Auto-scaling of Green Cloud Computing Infrastructure," *Future Generation Compututer System*, vol. 28, no. 2, pp. 371–378, Feb. 2012.

[67] R. Mian, P. Martin, F. Zulkernine, and J. L. Vazquez-Poletti, "Towards Building Performance Models for Data-intensive Workloads in Public Clouds," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[68] B. Mozafari, C. Curino, A. Jindal, and S. Madden, "Performance and Resource Modeling in Highly-concurrent OLTP Workloads," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013.

[69] I. K. Kim, J. Steele, Y. Qi, and M. Humphrey, "Comprehensive Elastic Resource Management to Ensure Predictable Performance for Scientific Applications on Public IaaS Clouds," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014.

[70] A. Matsunaga and J. A. B. Fortes, "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[71] F. Tian and K. Chen, "Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011.