2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

An Exploratory Study on Assessing the Impact of Environment Variations on the Results of Load Tests

Ruoyu Gao* and Zhen Ming (Jack) Jiang†
Software Construction, AnaLytics and Evaluation (SCALE) Lab
York University, Toronto, ON, Canada
{rgao*, zmjiang†}@cse.yorku.ca

***Abstract*—Large-scale software systems like Amazon and healthcare.gov are used by thousands or millions of people every day. To ensure the quality of these systems, load testing is a required testing procedure in addition to the conventional functional testing techniques like unit and system integration testing. One of the important requirements of load testing is to create a field-like test environment. Unfortunately, this task is often very challenging due to reasons like security and rapid field updates. In this paper, we have conducted an exploratory study on the impact of environment variations on the results of load tests. We have run over 110 hours load tests, which examine the system's behavior under load with various changes (e.g., installing an anti-virus program) to the targeted deployment environment. We call such load tests as environment-variation-based load tests. Case studies in three open source systems have shown that there is a clear performance impact on the system's performance due to these environment changes. Different scenarios react differently to the changes in the underlying computing resources. When predicting the performance of the system under environment changes that are not previously load tested, our ensemble models out-perform (24% - 94% better) the baseline models.**

I. Introduction

Load testing is an important type of non-functional testing technique to ensure the performance and the scalability of large-scale software systems like AT&T's telecommunication systems [1] and Microsoft's web services [2]. These systems are used by millions of users concurrently everyday. The failure to process high volumes of user traffic could result in catastrophic consequences and unfavorable media coverage (e.g., the botched launch of US Government's Health Care website [3] and the crashes of the Consensus websites for Statistics Canada [4] and Bureau of Statistics in Australia [5]).

Load testing in general refers to the process of assessing system behavior under load in a field-like environment [1]. The load drivers are used to generate hundreds or thousands of concurrent requests, which mimic the realistic user behavior, to the system under test (SUT). During the course of the load test, execution logs (e.g., the printf and System.out statements) and performance counters (e.g., CPU, memory and disk utilizations) are recorded for further analysis.

Although some load tests are conducted in the field, it is often very difficult and costly to access the actual field environment [6], [7]. Most of the load testing is done in a lab environment [8]. The computing hardware can be purchased in-house [6] or by renting virtual machines from cloud computing service providers [9], [10]. However, extra care should be taken to ensure the lab environment closely resembles various characteristics of the field environment. For example, if the database and the web sever are far apart (e.g., in two different continents), network delays should be added when deploying the SUT in the lab to simulate cross-the-continent network latency using Network Virtualization tools like Shunra [11]. However, re-creating the field-like environment in the lab is very challenging due to the following three reasons:

1) **Additional Tools**: System operators may install additional tools in the field for security [12] and monitoring purposes [13] without informing the performance analysts about these changes.
2) **Shared Environment**: Many of the systems are installed in a shared environment (e.g., public cloud or multi-tenant), in which one or multiple resources are shared by different users [10], [14]. Deploying in a shared environment can cause instability of the performance behavior of the SUT [9], [15] and degrading user experience [16].
3) **Rapid Changes**: Nowadays, many companies adopt the Continuous Integration/Continous Delivery (CI/CD) process, in which software changes and environment configuration changes are constantly pushed to the field. Companies like Netflix are constantly updating their various components in the field [17]. It is not feasible for them to re-create an up-to-date test environment.

In this paper, we have conducted an exploratory study on assessing the impact of the environment variations on the results of load tests. In addition to running load tests in an ideal environment (a.k.a, a test environment with no additional software processes running), we have also run various environment-variation-based load tests. Examples of the environment-variation-based load tests are running the SUT in a system deployment environment (SDE) where other processes (e.g., anti-virus) are active, or running load tests in an SDE with limited computing resources (e.g., constraining CPU resource). Our objective is to compare and understand the performance behavior of the SUT under these tests and try to model the impact of changes in the SDE on the results of the load tests. We have proposed and investigated the following three research questions (RQ):

- **RQ1 (Impact Analysis)** *How different can the results of a load test be under realistic changes to the SDE?* We

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.22

have found that there is a clear difference in terms of the SUT's performance behavior under load when deploying in these different environments.

- **RQ2 (Sensitivity Analysis)** *Which test scenarios are sensitive to the variations in the SDE?* We have found that there is no one-size-fits-all answers to this problem. Varying the amount of computing resources would have different performance impact to different scenarios.
- **RQ3 (Model-based Analysis)** *Can we effectively model the performance impact of the SUT due to the variations in the SDE?* We have developed an ensemble-based modeling approach, which are trained with the results of various environment-variation-based load tests, to better predict the system performance for environment changes which are not previously assessed.

The contributions of this paper are:

1) This is the first work to assess the impact of the load testing results with respect to the variations in the SDE. Compared to the existing field testing techniques like Chaos Monkeys [17], which assess the SUT behavior by randomly turning off some processes or machines, our environment-variation-based load testing can assess the behavior of the SUT in a more fine-grained manner. The results of our testing can provide useful suggestions to the performance analysts and the system operators (e.g., process A cannot be co-deployed with the SUT due to high contention of the CPU resources).
2) We have proposed an ensemble-based performance model to predict the system performance for changes to the SUT, which have not been previously tested. Case studies show that the performance of our ensemble models is 24% - 94% better than the baseline models.
3) In this paper, we have conducted various environment-variation-based load tests on three open source systems. We have made this data (over 110 hours of load testing data) available online [18] to support replications and to encourage further research on this topic.

Paper Organization

The rest of the paper is organized as follows: Section II explains the background and the related work. Section III provides an overview of our case study setup. Sections IV, V and VI study the three research questions. Section VII discusses the threats to validity. Section VIII concludes the paper and discusses some future work.

II. Background and Related Work

There are three areas of prior works which are related to this paper: (1) load testing setup, (2) performance monitoring and field testing, and (3) performance modeling.

A. Load Testing Setup

There are many techniques proposed in the area of designing, execution and analyzing load tests. For details, please refer to the survey by Jiang and Hassan [8]. In this subsection, we are only going to discuss one particular related area of load testing research: setting up a load testing environment.

If possible, it is always recommended to perform load tests in the actual field [6], [7] to gain the most accurate assessment of the SUT's behavior under load. Otherwise, load testing practitioners need to create a test environment, which closely mimics the deployment field [19]. There are many works devoted to mimicking various aspects of the field environment (e.g., creating the realistic network latency [11], and creating realistic database [2], [20], [21], [22]). However, it is still not clear whether all the aspects of the field environment have been properly captured in the test environment. Furthermore, the modeling and the configuration of a field-like test environment becomes increasingly harder nowadays due to the rapid changes in the field environment introduced by the CI/CD process [17].

B. Performance Monitoring and Field Testing

Detailed monitoring has a huge performance overhead, which may slow down the system execution or even alter the system behavior [23]. It is important to measure the system's performance multiple times to avoid measurement noise and errors [24]. If the SUT is deployed in a shared environment (e.g., public cloud), the monitored system performance can be unstable [15]. It is still a challenging area for properly monitoring and analyzing the results of an SUT in such shared environment [14]. This paper differs from the above works, as it tries to empirically assess and model the performance impact due to the changes in the SDE.

Due to the complexity and the rapid changes of their field environment, large companies like Amazon [25], Netflix [17] and Microsoft [26], [27] have recently started to conduct a new type of field testing, called "Chaos Engineering". During testing, system operators or programs, called Chaos Monkeys, randomly turn off some processes or even machines in the field environment. The goal of the Chaos Engineering is to assess the system behavior under various extreme conditions. As these systems are used by millions of people worldwide, they have very high requirements in terms of reliability and robustness. These systems are expected to be still available for service even when some of their processes got terminated or even some machines are unresponsive. The environment-variation-based load testing introduced in this paper is different from Chaos Engineering as they have different goals. The goal of our environment-variation-based load tests is to assess the system's performance behavior under various realistic changes to the SDE. The goal of chaos engineering is not performance assessment, but rather assessing the reliability and the robustness of the SUT.

C. Performance Modeling

In general, there are three categories of performance modeling techniques: analytical queuing-based models, data mining-based models, and hybrid models.

- **Analytical Queuing-based Models:** Analytical queuing-based models apply queuing theory [28] to model the

performance behavior of a system. Queuing models mimic system resources (e.g., CPU and memory) as queues, predict the time that different scenarios spend on different queues and estimates the resource utilizations and throughput information. In addition to hardware resources, there are also various software resources (e.g., database or middleware) in a system. The single layer queuing models can only be used to model the performance of hardware resources. To model the performance of both hardware and software resources, Layered Queuing Models (LQN) are required [29]. Since queuing models require the knowledge of the system internals (e.g., the deployment topology, the queues and locks involved in a scenario, and whether the call is synchronous or asynchronous), they are also called *white-box models* [30].

- **Data Mining-based Models:** Compared to LQN, data mining models treat the system under test as a black box and do not require the knowledge of the system internals. Hence, they are also called *black-box models* [30]. Data mining models rely on various statistical and AI-based techniques to model the system's performance behavior. Most of the data mining-based performance models discussed in the research literature are regression-based models (e.g., multiple linear regression-based models [31], [32], regress tree-based models [32], multivariate adaptive regression splines (MARS)-based models [33], and quantile regression-based models [34].
- **Hybrid Models:** Recently, there are works (e.g., [35], [30]) which combine the analytical queuing and the data mining-based models. Studies studies show that the resulting hybrid models yield shorter training time than the data mining-based models and have better performance compared to the analytical queuing-based models.

Unfortunately, there are very few empirical studies on assessing the performance of various performance modeling techniques (except [36]). In [36], Gao et al. empirically evaluated the effectiveness of various performance modeling techniques in terms of detecting load-related problems. However, there are no works which try to empirically assess the impact of environment variations on the performance modeling techniques. This is one of the main objectives of this paper.

III. Case Study Setup

In this section, we will introduce the setup of our case studies. In particular, we will describe the motivation and the descriptions of three research questions, the case study systems and their testing scenarios, and the load test setup.

A. Research Questions

We have proposed the following three RQs to assess the system's performance under different variations to the SDE:

- **(RQ1) Impact Analysis**: *How different can the results of a load test be under realistic changes to the SDE?*
  Although many efforts have been made to mimic the lab environment close to the actual field, in many cases it is not possible to completely capture all the environment characteristics in the field. For example, the system operators might decide to install additional monitoring probes or anti-virus software in the field for purposes like monitoring and security without informing the performance analysts. Hence, in this RQ, we want to examine the degree of performance deviation for SUT when running under an ideal environment and under an environment with some realistic changes.
- **(RQ2) Sensitivity Analysis**: *Which test scenarios are sensitive to the variations in the SDE?*
  The system consumes different amount of computing resources, while processing different scenarios. For example, the scenario of compressing a file would probably consume more CPU and disk resources, instead of memory; whereas uploading an image would probably consume more disk and network resources, rather than CPU. In this RQ, we want to perform a sensitivity analysis on the performance impact of different scenarios with respect to the amount of computing resources allocated.
- **(RQ3) Model-based Analysis**: *Can we effectively model the performance impact of the SUT due to the variations in the SDE?*
  In practice, each software system consumes various computing resources. The resource consumption patterns would not be uniform throughout the test. For example, the amount of the computing resources consumed would be much higher when an anti-virus program is actively scanning than the idle period. In addition, due to the complexity of the field environment, in many cases, it would not be feasible to exercise all the variations in the SDE. Some enterprise systems (e.g., SAP's ERP systems) are deployed in thousands of customers' site, each of which has different field configurations set by the system operators. Hence, in this RQ, we want to research into modeling techniques which can effectively predict the SUT's performance behavior for changes to the SDE that are not previously assessed.

TABLE I: Case Study Systems

System	Domain	Category	# of Test Scenarios	SLOC
DS2	Benchmark application	JSP	4	$\sim$ 100
PET	E-commerce	Hibernate	15	$>$ 2000
JMS	Mail Server	Mailet Container	6	$>$ 4,000

B. Target Systems and Test Scenarios

In this paper, we use three open source systems to conduct our case study: Dell DVD Store (DS2) [37], Petclinic (PET) [38] and James Mail Server (JMS) [39]. Each system has different types of components, configurations and uses different development technologies. Table I shows the summary of these three systems. Due to space limitations, please refer to our replication package [40] for the details of our testing scenarios and testing loads.

- **DS2**: DS2 is an open source e-commerce website made by Dell to benchmark the quality of their hardware. It has two components: web server and database. It is deployed on the Tomcat as web server, and uses MySQL as database. The system has four different scenarios: login, browsing DVD, purchasing items and registrations for new customers.
- **PET**: Petclinic is a web application based on Java Spring framework and Hibernate. Similar to DS2, it also uses Tomcat as web server and MySQL as database. Petclinic has 15 different test scenarios related to pet owners (e.g., view pet owners, edit owners information), pet (e.g., view pet, add visit log to pet), and veterinarians (e.g., view list of veterinarians).
- **JMS**: The Apache James Mail Server is an open source mail server written in Java. Different from DS2 and PET, JMS is a stand-alone system and does not need any other supporting components. Based on [41], [42], we define six scenarios with two from each the following three categories: (1) sending emails with or without attachments; (2) receiving emails with or without attachments; and (3) reading only the mail headers or the whole emails.

C. Test Setup

We use two machines to execute our load tests. One machine is used to deploy the SUT. Its configuration is Intel® Core™ 2 CPU 2.40 GHz, 2 GB memory and a 160 GB 7200 RPM hard drive. The other machine is used as a load generator to mimic the hundreds of users using the system concurrently. The load generator machine has Intel® Core™ i7-4790 CPU 3.60 GHz, 16 GB memory and a 2 TB 7200 RPM hard drive. The SUT and the load generator are deployed on separate machines, as this is the standard practice to eliminate the performance interference caused by the load generator [8].

We use an open source tool, called Apache JMeter [43], as our load generator and pidstat [44] to monitor and record the performance behavior during the course of a load test. We use the constant load profile, which generates stable request rate, to test the target systems. In addition, we also record the response time logs from JMeter and the access logs from Tomcat for each load test. The logs are used to extract the workload and the response time information.

To simulate the variations of the SDE, we have used the following three tools:

- **rsync**: Many of the field machines perform period data backup to avoid potential data losses due to machine failures or security breaches. In this paper, we use a data backup utility, called rsync [45], which is used to perform automated remote data backup during a load test.
- **ClamAV**: We use an open source anti-virus program, ClamAV [46] to evaluate the performance impact of SUT under virus scan.
- **stress-ng**: The above two applications are used to simulate realistic changes to the SDE. However, there can be more variations to the SDE in reality. Hence, we need to have a way to assess the SUT behavior under more variations to the SDE (e.g., a different computing resource usage pattern compared to rsync and ClamAV). To address this problem, we use an open source application, stress-ng [47] to control the amount of available computing resources in a more fine-grained manner. Stress-ng can be configured to consume different amount of computing resources like CPU, memory and disk, and hence, constraining the SDE that SUT is running under.

While running load tests, all the configurations (e.g., the load profile and the pidstat setup) are kept the same for the same target systems, except activating or deactivating the above the three tools. In total, we have run more than 110 hours of load tests and collected about 150 performance counters and approximately 10 GB logs from three target systems. For the details on the experiments, please refer to the next two sections (Sections IV and IV).

IV. RQ1 - Impact Analysis

In many cases, a load test is running in an ideal environment, in which only the SUT is running. The reasons are two-fold. First, it is much easier to monitor and analyze the performance of the SUT under load in a cleaner environment. Second, the performance analysts are not clear about the actual characteristics of the field environment. Hence, in this research question, we want to assess the degree of performance deviations between load tests running in an ideal environment and in an realistic environment (e.g., an environment in which the anti-virus is running along with the SUT).

A. Experiment

The same load tests are executed with varying environment setup. We have conducted the following four environment-variation-based load tests for all three target systems:

- *Ideal test*: First, two load tests, $ideal_1$ and $ideal_2$, are run in an ideal environment, where no additional processes are activated. The reasons for executing the *ideal test* twice is to check whether the performance behavior in the ideal setting is similar to each other.
- *Backup test*: We run one load test in an environment, in which rsync is running along with the SUT. This setup is to simulate the field environment in which remote data back-up is enabled.
- *SScan test*: We run another load test in an environment, in which the anti-virus application ClamAV is running along with the SUT. The ClamAV is configured to perform anti-virus scan using only one thread. Both this and the *DScan test* below are to simulate the field environment in which the anti-virus process is running.
- *DScan test*: This load test setup is the same as the *SScan test*, except ClamAV is configured to perform anti-virus scan using two threads. The goal here is to simulate the scenario that the operators want to perform a faster anti-virus scan.

Collectively, we call the Backup test, the SScan test, and the DScan test as the "realistic runs". In total, five load tests are run for each system.

TABLE II: Percentage of test scenarios which have different performance behavior between the ideal and the realistic runs.

Systems		Test Runs				
		$Ideal_1$	$Ideal_2$	Backup	DScan	SScan
DS2	$Ideal_1$	-	0.00%	25.00%	50.00%	75.00%
	$Ideal_2$	0.00%	-	25.00%	100.00%	100.00%
PET	$Ideal_1$	-	0.00%	73.33%	93.33%	100.00%
	$Ideal_2$	0.00%	-	73.33%	100.00%	100.00%
JMS	$Ideal_1$	-	0.00%	100.00%	100.00%	100.00%
	$Ideal_2$	0.00%	-	100.00%	100.00%	100.00%

B. Result Analysis

In order to investigate the impact of variations in the SDE on model accuracy, we perform the following two types of analyses: statistical comparison and performance model assessment.

1) Statistical Comparison: We statistically compare the response time between the ideal runs and the three realistic runs using the Wilcoxon Rank Sum test and Cliff's Delta (WC). Wilcoxon Rank Sum (WRS) test is a non-parametric test which compares the distributions of the two datasets. For example, if two sets of response time are about one minute (1 minute vs. 1.0006 minute) and the WRS test shows statistical significance between these two datasets. However, the differences between the two sets are so small (36 milliseconds), humans would hardly notice the differences. Hence, to quantify the strength of the differences between the two distributions, we use Cliff's Delta (CD). CD is a non-parametric technique to calculate the effect size between two distributions [48]. CD quantifies the level of the differences as follows:

$$\text{effect size} = \begin{cases} \text{trivial} & \text{if } CD < 0.147 \\ \text{small} & \text{if } 0.147 \leq CD < 0.33 \\ \text{medium} & \text{if } 0.33 \leq CD < 0.474 \\ \text{large} & \text{if } 0.474 \leq CD \end{cases}$$

Table II shows the number of scenarios which are impacted by realistic changes to the SDE: each cell indicates how many scenarios have different performance behavior when comparing one ideal run and one other run. We consider the differences are significant when the WRS test shows statistical significance and CD shows medium to large effect sizes. For example, in DS2, when comparing the $ideal_1$ and the backup runs, there is one out of the total four tested scenarios which have different performance behavior. The cell shows an "-" when it is compared against itself (e.g., $ideal_1$ comparing against $ideal_1$). In general, the performance behavior is the same for all the scenarios when comparing two ideal runs. However, there is at least one scenario from all three systems whose performance behavior is different when comparing against the ideal and the realistic runs.

2) Performance Model Assessment: As performance requirements are seldom available and up-to-date [49], [50], performance analysts usually adopt the no-worse-than previous principle. In other words, performance analysts build performance models based on the load testing data from previous runs and predict on the current runs. There can be performance problems, if there is a large deviation between the actual measurements and the predicted results. Hence, we also build performance models using the load testing results from the ideal runs and compare the prediction results on the ideal runs as well as the realistic runs. Since queuing models cannot easily model environment variations, in this paper, we are only going to evaluate the effectiveness of data mining-based performance models. In particular, we build the following three data mining-based performance models:

- **Generalized Linear Model (GLM)** represents the system behavior using a global function of workload mixes and resource usage utilization [51]. Extra care is needed to ensure the assumptions associated with the GLM model are met.
- **Multivariate Adaptive Regression Splines (MARS)** is a model that can split data with different kinds of behavior into different sub models using one or more "hinge function" [33]. Each hinge function is a function indicating two different behaviors based on certain cut-offs. All the hinge functions can be automatically produced by the model, modeling system in different period rather than as a whole like GLM. We want to use MARS to model the different phases in a load test (e.g., when anti-virus program is active vs. idle).
- **Regression Tree (RegTree)** builds a decision tree model by splitting the data points into certain leaves. Compared to GLM, RegTree does not have many assumptions associated with the model [32]. Similar to MARS, RegTree can also model the behavior of the system in different phases, but in a tree-like manner.

We build the performance models using the results from the ideal runs. Then we either predict the performance of another ideal run or the performance of one realistic run. Then we perform the WC analysis between the predicted values and the actual measured results. If they are statistically significantly different by the WRS analysis and have medium to large effect sizes, this scenario is considered to have large performance deviations between the predicted values and the actual values. Table III shows the percentage of scenarios from each system where there large deviations between the predicted and the measured values. For example, in DS2, when using RegTree as the prediction model and trained on the data from the ideal run, there are 12.50% of the test scenarios which are different when predicting on another ideal run. However, when predicting on the realistic runs, 50% of the scenarios have large deviations between the actual and the predicted values.

TABLE III: Percentage of test scenarios in which there are differences between the predicted and the actual values.

Models	DS2		PET		JMS	
	Ideal	Realistic	Ideal	Realistic	Ideal	Realistic
GLM	37.50%	37.50%	43.33%	38.89%	50.00%	66.67%
MARS	12.50%	37.50%	40.00%	45.56%	16.67%	72.22%
RegTree	12.50%	50.00%	3.33%	28.89%	8.33%	94.44%

In general, the prediction results on the ideal runs are better than the realistic runs when using MARS or RegTree as the modeling techniques. This is the opposite when using the GLM as the modeling technique. We believe this is mainly due to the performance of the modeling techniques, as the model performance of the GLM is significantly worse than the other two techniques. For example, when training on the ideal run and predicting on another ideal run in DS2, the performance of GLM is more than three times worse than the performance of MARS and RegTree. For RegTree, although it can predict the performance from another ideal run with the highest performance, the prediction results on the realistic runs are much worse.

C. Summary

Findings: There is a clear performance impact when incorporating realistic changes to the SDE. Consequently, the performance models built using the ideal runs cannot be used to accurately predict the behavior of the realistic runs.
Implications: As it can be unavoidable to have variations to the SDE, there could be large performance deviations between the load tests running in an ideal environment and the field. It is worthwhile to conduct environment-variation-based load testing to assess the impact of the SDE changes.

V. RQ2 - Sensitivity Analysis

In the previous RQ, we have demonstrated the need to conduct environment-variation-based load testing. When the system contains abundant computing resources, the SUT and the other applications can run without interfering each other. However, when the resources are limited, the performance of the SUT can be impacted if some of its scenarios are competing for the same computing resources against another application. In this RQ, by leveraging the environment-variation-based load tests, we want to conduct a sensitivity analysis on the performance impact of different scenarios with respect to the amount of available computing resources.

A. Experiment

If one scenario consumes much CPU, limiting the CPU resource would result in a large increase in its response time. Otherwise, its performance would not be impacted. Hence, in order to find out which scenario is sensitive to what kinds of computing resources, we run the load tests with limited computing resources. We run load tests along with stress-ng multiple times. Each time, stress-ng is configured to consume a different amount of computing resources, which forces the SUT to be run under different constrained conditions. For each system, we run ten additional environment-variation-based load tests apart from the two ideal tests. The details are as follows:

- *Ideal test*: We run the load test under the ideal environment twice. We call these two runs as ideal_1 and ideal_2. These two runs are used as our baseline in this RQ.
- $CPUStr$ *test*: We run the load tests with stress-ng configured to consume different amount of CPU resources. Two CPUStr tests (CPUStr_1 and CPUStr_2) are run in this RQ with with stress-ng configured to consume 30% and 70% of one particular CPU core, respectively.
- $DISKStr$ *test*: Similar to the CPUStr tests, we run the load tests with stress-ng configured to perform writes to the disk with one or two worker threads. We call these two runs as DISKStr_1 and DISKStr_2, respectively.
- $MEMStr$ *test*: Similar to the CPUStr and the DISKStr tests, we ran the load tests with stress-ng configured to consume 512 MB or 1 GB of memory. We call these two runs as MEMStr_1 and MEMStr_2, respectively.
- *Multiple Resource Stress test*: In order to assess the system behavior under more than one types of resource stresses, we also performed tests which combined two to three types of low-level resource stresses. For example, we combine stress-ng with 30% CPU consumption and stress-ng with 512 MB memory corruption as $CPUMEMstr_1$. We call the runs with all types of resource (CPU, memory and disk) stressed as $ALLStr_1$.

Collectively, we call the CPUStr test, the DISKStr test, and the MEMStr test and the Combination Stress test as "lab stress runs". To distinguish from the multiple resource stress tests, we refer to the CPUStr test, the DISKStr test, and the MEMStr test, as the "single resource stress test".

B. Result Analysis

We use the WC analysis to compare the response time between the ideal runs and a particular type of lab stress runs. For example, if WC result of the login scenario shows a large difference when comparing the ideal run and the CPUStr runs, the performance of the login scenario is sensitive to the amount of available CPU resource. The overall WC results are shown in Table IV. Each row shows the percentage of scenarios from that subject system which has different response time from the ideal runs. For example, the third row in the table shows that 75% of the scenarios in DS2 are impacted when running with lower level CPU stress (CPUStr_1).

As expected, the two ideal runs are similar to each other. Hence, no scenarios are flagged. Generally speaking, lower level of stress will have less effect to the scenario performance. But if the scenario heavily relies on certain type of resource, limiting this resource will have significant a impact on this scenario. DS2 and PET both use database for information storage, and JMS needs to access the disk to read and write emails. Hence, both levels of disk stresses show significant impact for all three systems. JMS consumes large amount of memory to store the users' states and PET uses Hibernate, which caches the database information to optimize database performance. Hence, the scenarios from these two systems are impacted by memory stresses. DS2 is a simple benchmarking system using the JSP technology. The scenarios from DS2, which do not consume much memory, are not impacted by the changes in the memory resource. Compared to DS2 and PET, the impact of the CPU stress is smaller in JMS.

Compared to single resource stress runs (a.k.a., the CPUStr, the DISKStr, and the MEMStr runs), multiple resource stress

runs demonstrate a much bigger difference from the ideal runs. Almost all the scenarios are impacted in all the multiple resource stress runs. The only exception is DS2 under the CPUDISKStr test, in which the performance of one scenario (registrations for new customers) is not impacted. The same scenario's performance is not impacted in the CPUStr_1 and MEMStr_1 runs, either.

C. Summary

Findings: Varying the amount of computing resources can impact the performance behavior of the SUT. However, since individual scenarios consume varying amount of computing resources, their performance may or may not be impacted by the underlying changes to the SDE.
Implications: There is a need to create better performance models to assess the performance implications of different scenarios with respect to the changes in the SDE.

VI. RQ3 - Model-based Analysis

RQ2 shows that there is no one-size-fits-all answer to the relationship between the performance of individual scenarios and the amount of available computing resources. In this RQ, we will propose an ensemble-based modeling approach to systematically assess the performance implications of different scenarios with respect to the changes in SDE.

A. Ensemble Modeling

Our proposed ensemble models consist of many small performance models, built using the data from segments of environment-variation-based load tests. These small performance models are called sub-models, which perform their own predictions. The overall ensemble model output is based on how close the state of the current test environment and the environment enclosed in the sub-models. The more similar the state of the two environments are, the higher weight it should be given to the prediction results of that sub-model. Figure 1 illustrates the overall process of our ensemble modeling. It consists of three steps: model building, distance calculation and model prediction.

Step 1 - Model Building: As the available computing resources can fluctuate during the course of a load test due the state of different software processes (e.g., anti-virus actively scanning vs. anti-virus being idle). Hence, in order to accurately capture the SUT's behavior under different environment conditions, we split the load testing data with a moving window. The window contains data within a fixed time window size *i*. Hence, the first window (Training window_1) contains the data from time 0 to time $i-1$; the second window (Training window_2) contains the data from time 1 to time i, and so on. For the data within that window, we built performance models using the data from that window as the training data. One performance model is built for the response time of each scenario. For example, the model $M_{scenarioA,i}$ corresponds to the sub-model built to model the response time of scenarioA using the training data from Training window_i.

This process is applied on all our executed environment-variation-based load tests (a.k.a., the ideal runs, the realistic runs, and the lab stress runs). For the data from each window, we build three kinds of performance models: GLM-based models, MARS-based models and RegTree-based models.

Step 2 - Distance Calculation: In the previous step, we have built many sub-models, which capture the performance behavior of SUT under different environment variations. The next step is to defined how to determine the final output from the ensemble model.

When the ensemble model is used (a.k.a, during the testing stage), each sub-model would give a prediction value. We need an approach, which automatically selects the most suitable sub-model(s) and use their prediction result(s) as the final ensemble output. Our distance function is defined to measure the similarity between the training and the testing environment. If the SUT is run in a test environment similar to the current target environment, their performance behavior should be similar. Hence, the prediction results from these sub-models should be given more trust than other sub-models in the final ensemble output.

We will explain the calculation of our distance function with an example. Figure 2 shows an example of the resulting sub-model built using the regression tree technique. In this case, the sub-model contains two independent variables: CPU and disk. Hence, our distance function measures the degree of similarity of these two counters between the training window and the current target environment.

Similar to the *Training window*, we define the *Test window* with a fixed time window with a fixed time window size j. For example, at current time t, the window (Test Window_t) consists of the data from $t-j-1$ to t (a.k.a., the data from the past j minutes). Note that the size of the training and the testing windows are configurable and they are not necessarily the same. For each prediction, we conduct the WC analysis on the CPU data between the current (*Test Window*$_t$) and all the (*Training Windows*). For the details of the WC analysis, please refer to Section IV. Similarly, the WC analysis is also applied on the disk data. Hence, for each of the performance models, we have two WC values: one for the CPU counter and the other one for the disk counter. The value of the distance function for this model is the average of the two WC values. For example, if the WC values for the CPU and the disk counters are 0.4 and 0.6. The resulting distance function for this performance model is 0.5 ($\frac{0.4+0.6}{2}$). Different performance models would have different types of performance counters. For example, if a MARS-based sub-model contains only one independent variable: the disk. The distance function for this sub-model is just the WC values for the disk counter.

Step 3 - Model Prediction: Once we have calculated the distance functions of all the sub-models, we need to have a way to combine the prediction results from all these sub-models into one final output for the ensemble model. In this paper, we have experimented with the following three ensemble strategies:

- **Mean** calculates the average of the prediction results from

TABLE IV: Percentage of test scenarios which are different between the ideal and the lab runs.

Systems		Ideal Runs		CPU Stress Runs		Disk Stress Runs		Memory Stress Runs		Combination Stress Runs			
		$Ideal_1$	$Ideal_2$	$CPUStr_1$	$CPUStr_2$	$DISKStr_1$	$DISKStr_2$	$MEMStr_1$	$MEMStr_2$	$CPUDISKStr_1$	$CPUMEMStr_1$	$DSKMEMStr_1$	$ALLStr_1$
DS2	$Ideal_1$	-	0.00%	75.00%	100.00%	100.00%	100.00%	0.00%	0.00%	100.00%	75.00%	100.00%	100.00%
	$Ideal_2$	0.00%	-	75.00%	100.00%	100.00%	100.00%	0.00%	0.00%	100.00%	75.00%	100.00%	100.00%
PET	$Ideal_1$	-	0.00%	73.33%	93.33%	100.00%	100.00%	0.00%	93.33%	100.00%	100.00%	100.00%	100.00%
	$Ideal_2$	0.00%	-	66.67%	80.00%	100.00%	100.00%	0.00%	93.33%	100.00%	100.00%	100.00%	100.00%
JMS	$Ideal_1$	-	0.00%	16.67%	50.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
	$Ideal_2$	0.00%	-	33.33%	66.67%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

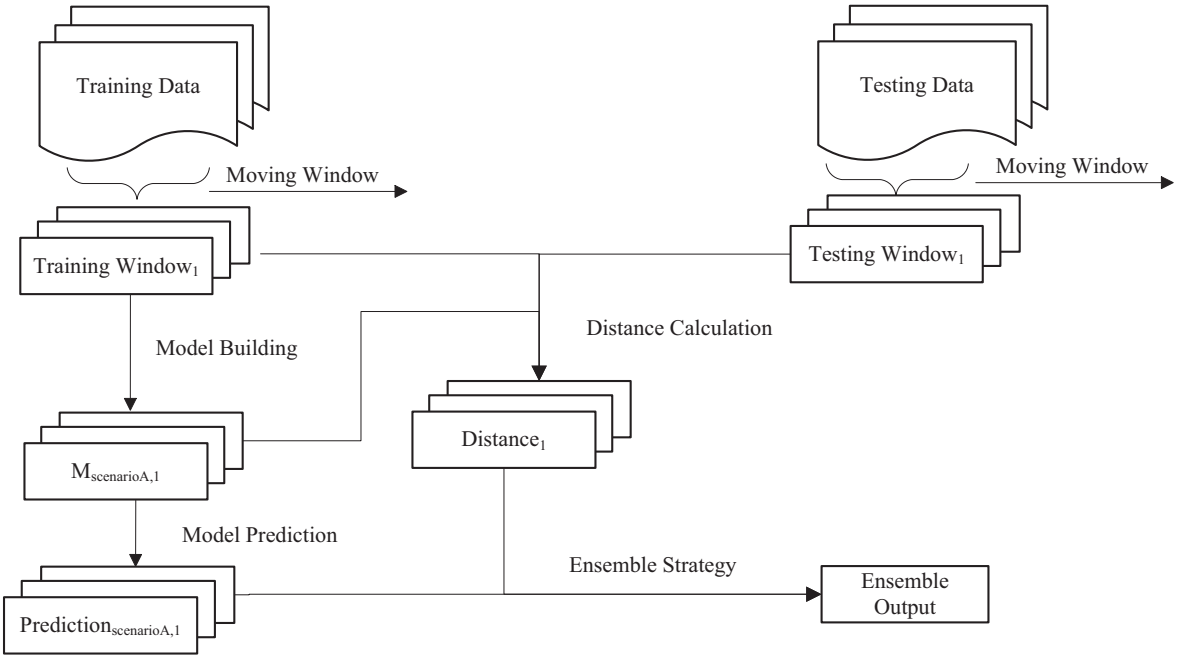

Fig. 1: Our ensemble-based modeling approach.

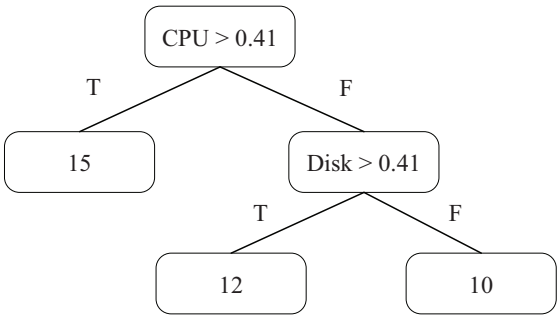

Fig. 2: An example of a regression tree-based performance model.

all the sub-models. Note that in this case, the values calculated from the distance functions are not used.

- **Winner-takes-all** uses the prediction results with the smallest distance values. If there is a tie, the median of the prediction results from all the best sub-models is used.
- **Weighted Average** provides a weight for each of the prediction results and aggregates them. For example, there are three sub-models, which have WC values 0.2, 0.4, 0.6, respectively. The prediction results from these three sub-models are: 0.6, 0.3, and 0.4. The final ensemble output would be $\frac{\frac{1}{0.2+1}}{\frac{1}{0.2+1}+\frac{1}{0.4+1}+\frac{1}{0.6+1}} \times 0.6 + \frac{\frac{1}{0.4+1}}{\frac{1}{0.2+1}+\frac{1}{0.4+1}+\frac{1}{0.6+1}} \times 0.3 + \frac{\frac{1}{0.6+1}}{\frac{1}{0.2+1}+\frac{1}{0.4+1}+\frac{1}{0.6+1}} \times 0.4 = 0.44$. We take the inverse of WC values, as bigger WC values indicate larger differences. We also add one to each of the WC values to avoid the division by zero error.

For each test window, the ensemble model is only going to predict the last point (a.k.a., the prediction value for the current time).

B. Evaluation

We evaluate our ensemble models by training on the results of environment-variation-based load tests and predicting on the results of three realistic runs (the Backup, the SScan, and the DScan tests). In order to systematically evaluate our ensemble modeling technique, we vary the configurations of our ensemble models in the following three dimensions:

- **The size of the training window**: we use 10, 15, or 20 minutes as three different window sizes. We fix the testing window size as 10 minutes in order to compare the effect of the training window sizes.

- **The training dataset**: We evaluate our ensemble models with these five options: (1) ideal runs only, (2) ideal runs and single resource stress runs, (3) ideal runs and all the lab stress runs (a.k.a., single resource stress runs and multiple resource stress runs), (4) ideal runs, single stress runs, and realistic runs, and (5) all the runs. There are in total six different kinds of single resource stress runs as described in Section V: two CPU stress runs ($CPUStr_1$ and $CPUStr_2$), two memory stress runs ($MEMStr_1$ and $MEMStr_2$), two disk stress runs ($DISKStr_1$ and $DISKStr_2$). There are four kinds of multiple resource stress runs ($CPUDISKStr_1$, $CPUMEMStr_1$, $DISKMEMStr_1$, and $ALLStr_1$). For options (4) and (5), the realistic runs refer to the two out of the three realistic runs. For example, if we are predicting the results of the Backup test, the two realistic runs used in option (4) and (5) would be the DScan test and the SScan test. We call options (1) - (5) as "Ideal", "Lab", "LabCmb", "All", "AllCmb" in the rest of this section and in Table V.
- **Ensemble strategies**: We use using mean, winner-takes-all and weighted average as three options to calculate the final ensemble results.

To demonstrate the effectiveness of our ensemble-based modeling technique, we compare our results against the baseline technique, which is the single-model based approach. The single-model based approach only uses the data from the ideal run and builds just one performance model. Similar to the other two RQs, we calculate the percentage of scenarios whose predicted values and the measured values are different. In other words, the two sets of data are statistically different by the WRS test and the CD results shows medium to large effect sizes. In this case, we consider the performance model produces "bad prediction results".

Table V shows the results of the RegTree models. The bottom row with row name "Baseline" shows the prediction of baseline model. For example, half of the scenarios in DS2 suffer from bad prediction results when using window size as 20, the weighted average as the ensemble strategy and only the ideal runs as the training data. Overall, we can see the results from our ensemble models are the same or better than the baseline. Among the three ensemble strategies, winner-takes-all always outperforms the other two under the same window size and training dataset. If we only compare within the winner-takes-all strategy, the performance is similar to the baseline model when simply using the ideal runs as the training dataset. Comparing among different window sizes, Win_{15} and Win_{20} generally perform the same or better than Win_{10}.

Generally, adding more single resource stress runs into the training set would greatly improve the prediction performance. However, the effect of further adding more realistic runs or more multiple resource stress runs are mixed. This makes sense, as adding more single resource stress runs into the training set will provide more variations of the SUT behavior due to changes to the SDE. However, since most of the multiple resource stress runs would overload the system and do not closely resemble the realistic field behavior, the gain due to the addition of multiple resource stress runs is not obvious. Similarity, different realistic runs would have different impact to the underlying resources (e.g., data backup and anti-virus scanning consume different amount of computing resources), the additional of these runs would not further improve the model performance.

We have also experimented the ensemble-based modeling techniques using the GLM and the MARS as our modeling techniques. We analyze RegTree results for the purpose of illustrating the effectiveness of ensemble modeling approach compared with baseline approach. Besides, due to space limitations, we can not included all the results here. Please refer to [18], if you want the evaluation results for the GLM and the MARS-based ensemble models.

C. Summary

Findings: When predicting on realistic runs, our ensemble models generally perform better than the single models trained using the ideal runs only. Adding results from the single resource stress runs into the training dataset would improve the model performance. However, the effect of adding other environment-variation-based load tests is mixed. The model built using the larger window sizes (e.g., 15 or 20 minutes window sizes) are usually better than the models using smaller window sizes.
Implications: We can better model the performance behavior under realistic changes to the SDE by leveraging the results from the environment-variation-based tests. However, different types of environment-variation-based load tests would have different impact on the model performance. This motivates the need for further research into other kinds of environment-variation-based load tests (e.g., stressing the database component at various levels) and better performance modeling techniques.

VII. Threats To Validity

In this section, we will discuss the threats to validity.

A. Construct Validity

1) Performance Monitoring Overhead: We monitor the SUT's resource usage using an open source performance monitoring tool, called pidstat. This tool is very low overhead ($< 1\%$ CPU, $< 0.1\%$ memory, and very few disk writes). In addition, we also collect the logs from JMeter and Tomcat to calculate the workload and the response time. The access logs for Tomcat are enabled by default. JMeter and the SUT are run on different machines to avoid resource contention.

2) Evaluation Method: In this paper, we use the WC method to check whether the predicted results match with the actual measurements. We flag a prediction model to be bad if the WRS test shows statistically significant and CD test shows medium to large effect sizes between the predicted and the actual measured values.

Mean Percentage Absolute Error (MAPE) is another popular method to evaluate the quality of the predicted results [52]. The higher the MAPE values, the worse the prediction models

TABLE V: Percentage of scenarios which are different between the predicted results and the actual measurements. To conserve space, the three model selection techniques, mean, winner-takes-all, and weighted average, are shown as "Mean", "Winner" and "Weighted" in the table. Win_{10}, Win_{15} and Win_{20} refer to the three different training window sizes.

Window size	Ensemble strategy	DS2					PET					JMS				
		Ideal	Lab	Labcmb	All	Allcmb	Ideal	Lab	Labcmb	All	Allcmb	Ideal	Lab	Labcmb	All	Allcmb
Win_{10}	Mean	58.33%	41.67%	41.67%	41.67%	50.00%	51.11%	33.33%	48.89%	84.44%	95.56%	100.00%	66.67%	66.67%	66.67%	66.67%
	Winner	66.67%	16.67%	33.33%	33.33%	58.33%	55.56%	6.67%	15.56%	4.44%	13.33%	100.00%	33.33%	22.22%	22.22%	22.22%
	Weighted	58.33%	41.67%	41.67%	41.67%	41.67%	51.11%	15.56%	42.22%	66.67%	88.89%	100.00%	66.67%	66.67%	66.67%	66.67%
Win_{15}	Mean	58.33%	41.67%	41.67%	41.67%	50.00%	51.11%	33.33%	48.89%	84.44%	95.56%	100.00%	66.67%	66.67%	66.67%	66.67%
	Winner	66.67%	33.33%	8.33%	16.67%	25.00%	48.89%	6.67%	13.33%	6.67%	13.33%	100.00%	5.56%	0.00%	0.00%	0.00%
	Weighted	58.33%	41.67%	41.67%	41.67%	41.67%	51.11%	22.22%	42.22%	75.56%	88.89%	100.00%	66.67%	66.67%	66.67%	66.67%
Win_{20}	Mean	50.00%	41.67%	41.67%	41.67%	41.67%	44.44%	13.33%	44.44%	51.11%	80.00%	100.00%	66.67%	66.67%	66.67%	66.67%
	Winner	66.67%	16.67%	33.33%	25.00%	25.00%	64.44%	4.44%	13.33%	6.67%	11.11%	100.00%	72.22%	5.56%	27.78%	5.56%
	Weighted	50.00%	41.67%	41.67%	41.67%	41.67%	44.44%	11.11%	37.78%	44.44%	68.89%	100.00%	61.11%	66.67%	66.67%	66.67%
Baseline		(		50.00%		)	(		28.89%		)	(		94.44%		)

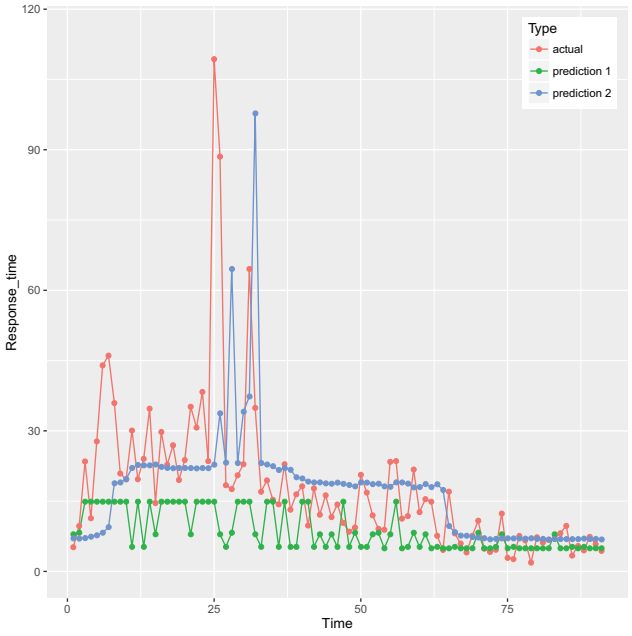

Fig. 3: Comparison of the prediction results from two performance models.

are. We have tried both the WC and MAPE as our evaluation methods and have found in certain scenarios MAPE values cannot highlight the strength and weaknesses of different models. Figure 3 shows one such example. Prediction 1 (the model trained with the ideal runs) and prediction 2 (the ensemble model trained with more tests) have MAPE values of 0.452 and 0.507, respectively. This indicates that the baseline model is better than the ensemble model. However, by looking at the graph, the baseline model did not accurately capture the trend of the data. The reason that the baseline model has a smaller MAPE is because the baseline model always gives a small prediction value than the ensemble model. When we use the WC analysis on these data, the CD values for the baseline and the ensemble models are 0.435 (medium effect size) and 0 (trivial effect size). This clearly shows that the prediction results from the baseline model are worse than the ensemble model's results.

B. Internal Validity

There are four configurable parameters in our ensemble models: the ensemble strategy, the training window size, the test window size, and the type of training data. In RQ3, we have kept the test window size fixed (10 minutes) and systematically assess the impact of three other parameters.

C. External Validity

In this paper, we assess the performance impact of environment changes using three open source systems. These three open source systems are picked because they are from different application domains, implemented with different technology, and have different deployment topologies. However, our findings may not be generalizable to other systems. In addition, although we have proposed various kinds of environment-variation-based load tests (e.g., realistic runs and lab stress runs), there can be more. For example, if the SUT is deployed in a multi-tenant environment, another type of realistic load tests can be running the SUT while the database is doing additional unrelated processing tasks. However, the goal of this paper is to explore this rarely studied but very important area and motivates further research into this subject.

VIII. Conclusion and Future Work

In this paper, we have evaluated the impact of the SDE variations on the results of load tests. We have run over 110 hours of environment-variation-based load tests on three different target systems. Our analysis shows that there is a clear performance impact on the SUT due to SDE changes. However, not all scenarios react the same to the changes of the computing resources. We have developed an ensemble-based modeling technique, which leverages the results of existing environment-variation-based load tests. Case studies show that our ensemble models perform better than the baseline models when predicting the performance of the realistic runs.

In the future, we plan to extend this study by researching into more types of environment-variation-based load tests and better performance modeling techniques. In addition, we also want to investigate techniques which can dynamically adjust the test environment (e.g., adding or removing computing resources) based on the performance of the SUT.

REFERENCES

[1] A. Avritzer and E. J. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software," *IEEE Transactions on Software Engineering*, 1995.

[2] M. D. Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann, "Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007.

[3] S. G. Stolberg and M. D. Shear, "Inside the Race to Rescue a Health Care Site, and Obama," 2013, http://www.nytimes.com/2013/12/01/us/politics/inside-the-race-to-rescue-a-health-site-and-obama.html, visited 2017-02-09.

[4] D. Beeby, "Design flaws crashed StatsCan's census website: documents," http://www.cbc.ca/news/politics/census-statistics-canada-computers-online-webpage-1.3649989, visited 2017-02-09.

[5] G. Mitchell, "Census 2016: IT experts say Bureau of Statistics should have expected website crash," http://www.smh.com.au/national/census-2016-it-experts-say-bureau-of-statistics-should-have-expected-website-crash-20160809-gqosj7.html, visited 2017-02-09.

[6] G. M. Leganza, "The Stress Test Tutorial," in *Proceedings of the 1991 Computer Management Group Conference (CMG)*, 1991.

[7] A. Savoia, "Web Load Test Planning: Predicting how your Web site will respond to stress," *STQE Magazine*, 2001.

[8] Z. M. Jiang and A. E. Hassan, "A Survey on Load Testing of Large-Scale Software Systems," *IEEE Transactions on Software Engineering*, 2015.

[9] J. Zhou, S. Li, Z. Zhang, and Z. Ye, "Position Paper: Cloud-based Performance Testing: Issues and Challenges," in *Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services (HotTopiCS)*, 2013.

[10] M. Yan, H. Sun, X. Wang, and X. Liu, "Building a TaaS Platform for Web Service Load Testing," in *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER)*, 2012.

[11] "Shunra Network Virtualization for HP Software," http://media.shunra.com/datasheets/ShunraNVforHP.pdf, visited 2017-02-09.

[12] "BlackBerry fixes critical Enterprise Server flaw," http://www.itpro.co.uk/630034/blackberry-fixes-critical-enterprise-server-flaw, visited 2017-02-09.

[13] "Identifying and resolving performance-related issues caused by the Behavior Monitoring and Device Control," https://success.trendmicro.com/solution/1056425-identifying-and-resolving-performance-related-issues-caused-by-the-behavior-monitoring-and-device-co, visited 2017-02-09.

[14] A. B. Bondi, "Challenges with Applying Performance Testing Methods for Systems Deployed on Shared Environments with Indeterminate Competing Workloads: Position Paper," in *Companion Publication for ACM/SPEC on International Conference on Performance Engineering (ICPE)*, 2016.

[15] P. Leitner and J. Cito, "Patterns in the Chaos&Mdash;A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM Transactions on Internet Technology (TOIT)*, 2016.

[16] X. Sun and A. May, "A Comparison of Field-based and Lab-based Experiments to Evaluate User Experience of Personalised Mobile Devices," *Advances in Human-Computer Interaction*, 2013.

[17] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos Engineering," *IEEE Software*, 2016.

[18] "Replication Package," https://goo.gl/vGk1HV, visited 2017-02-09.

[19] S. Barber, "Creating Effective Load Models for Performance Testing with Incomplete Empirical Data," in *Proceedings of the Sixth IEEE International Workshop on the Web Site Evolution (WSE)*, 2004.

[20] D. Bainbridge, I. H. Witten, S. Boddie, and J. Thompson, *Research and Advanced Technology for Digital Libraries*. Springer, 2009, ch. Stress-Testing General Purpose Digital Library Software.

[21] M. Grechanik, C. Csallner, C. Fu, and Q. Xie, "Is Data Privacy Always Good for Software Testing?" in *Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE),*, 2010.

[22] Y. Wang, X. Wu, and Y. Zheng, *Trust and Privacy in Digital Business*. Springer, 2004, ch. Efficient Evaluation of Multifactor Dependent System Performance Using Fractional Factorial Design.

[23] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Evaluating the accuracy of Java profilers," *Proceedings of the ACM SIGPLAN 2010 Conference on Programming Language Design and Implementation (PLDI)*, 2010.

[24] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically Rigorous Java Performance Evaluation," in *Proceedings of the 22nd International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 2007.

[25] J. Robbins, K. Krishnan, J. Allspaw, and T. Limoncelli, "Resilience Engineering: Learning to Embrace Failure," *Queue*, 2012.

[26] M. Acharya and V. Kommineni, "Mining Health Models for Performance Monitoring of Services," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2009.

[27] H. Nakama, "Inside Azure Search: Chaos Engineering," 2015, https://azure.microsoft.com/en-us/blog/inside-azure-search-chaos-engineering/, visited 2016-9-12.

[28] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[29] C. Barna, M. Litoiu, and H. Ghanbari, "Autonomic load-testing framework," in *Proceedings of the 8th ACM international conference on Autonomic computing*, 2011.

[30] D. Didona, F. Quaglia, P. Romano, and E. Torre, "Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2015.

[31] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[32] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[33] M. Courtois and M. Woodside, "Using Regression Splines for Software Performance Analysis," in *Proceedings of the 2nd International Workshop on Software and Performance (WOSP)*, 2000.

[34] A. B. de Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Why you should care about quantile regression," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[35] D. Didona and P. Romano, "Performance modelling of partially replicated in-memory transactional stores," in *Proceedings of the 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2014.

[36] R. Gao, Z. M. Jiang, C. Barna, and M. Litoiu, "A Framework to Evaluate the Effectiveness of Different Load Testing Analysis Techniques," in *Proceedings of the 9th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2016.

[37] "Dell DVD Store Database Test Suite," http://linux.dell.com/dvdstore/, visited 2017-02-09.

[38] "A sample Spring-based application," https://github.com/spring-projects/spring-petclinic, visited 2017-02-09.

[39] "Apache James Project," http://james.apache.org/, visited 2017-02-09.

[40] R. Gao, Z. M. J. Jiang, C. Barna, and M. Litoiu, "Replication Package," https://www.dropbox.com/s/1xzfxc28pf2h284/rep_package.zip?dl=0.

[41] "Exchange Server 2003 MAPI Messaging Benchmark 3," https://technet.microsoft.com/en-us/library/cc164328%28v=exchg.65%29.aspx?f=255&MSPPError=-2147217396, visited 2017-02-09.

[42] "Playing with Exchange in a Sandbox," https://technet.microsoft.com/en-gb/magazine/2006.08.exchangesandbox.aspx, visited 2017-02-09.

[43] "Apache JMeter," http://jmeter.apache.org/, visited 2015-10-23.

[44] "Performance monitoring tools for Linux," https://github.com/sysstat/sysstat, visited 2017-02-09.

[45] "rsync(1) - Linux man page," http://linux.die.net/man/1/rsync, visited 2017-02-09.

[46] "ClamAV," https://www.clamav.net/, visited 2017-02-09.

[47] "Stress-ng," http://kernel.ubuntu.com/~cking/stress-ng/, visited 2017-02-09.

[48] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other

surveys?" in *Annual meeting of the Florida Association of Institutional Research*, 2006.
[49] C.-W. Ho, L. Williams, and A. I. Anton, "Improving Performance Requirements Specification from Field Failure Reports," in *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE)*, 2007.
[50] A. Avritzer and A. B. Bondi, "Resilience Assessment Based on Performance Testing," in *Resilience Assessment and Evaluation of Computing Systems*, K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, Eds. Springer Berlin Heidelberg, 2012.
[51] P. McCullagh and J. A. Nelder, *Generalized linear models*. Chapman and Hall/CRC, 1989.
[52] A. Heiat, "Comparison of artificial neural network and regression models for estimating software development effort," *Information and software Technology*, 2002.