Reducing the Length of Field-replay Based Load Testing

Yuanjie Xia

A Thesis

in

The Department

of

Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Software Engineering) at

Concordia University

Montréal, Québec, Canada

August 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Yuanjie Xia**

Entitled: **Reducing the Length of Field-replay Based Load Testing**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Name of the Chair

_____ External Examiner
Dr. Name of External Examiner

_____ Examiner
Dr. Name of Examiner One

_____ Supervisor
Dr. Weiyi Shang

Approved by _____
Lata Narayanan
Department of Software Engineering

_____ 2021

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Reducing the Length of Field-replay Based Load Testing

Yuanjie Xia

With the development of software, load testing have become more and more important. Load testing can ensure the software system can provide quality service under a certain load. Therefore, one of the common challenges of load testing is to design realistic workloads that can represent the actual workload in the field. In particular, one of the most widely adopted and intuitive approaches is to directly replay the field workloads in the load testing environment, which is resource- and time-consuming. In this work, we propose an automated approach to reduce the length of load testing that is driven by replaying the field workloads. The intuition of our approach is: if the measured performance associated with a particular system behaviour is already stable, we can skip subsequent testing of this system behaviour to reduce the length of the field workloads. In particular, our approach first clusters execution logs that are generated during the system runtime to identify similar system behaviours during the field workloads. Then, we use statistical methods to determine whether the measured performance associated with a system behaviour has been stable. We evaluate our approach on three open-source projects (i.e., *OpenMRS*, *TeaStore*, and *Apache James*). The results show that our approach can significantly reduce the length of field workloads while the workloads-after-reduction produced by our approach are representative of the original set of workloads. More importantly, the load testing results obtained by replaying the workloads after the reduction have high correlation and similar trend with the original set of workloads. Practitioners can leverage our approach to perform realistic field-replay based load testing while saving the needed resources and time.

Acknowledgments

I would like to express my special thanks of gratitude to my supervisor (professor Weiyi Shang) as well as my colleague in SENSE lab, for giving me opportunity to do research and providing priceless guidance.

Contents

List of Figures vii

List of Tables viii

1 Introduction 1

2 Background 4

3 Approach 6

3.1 Characterizing workloads . . . 6

3.1.1 Log abstraction . . . 6

3.1.2 Generating workload signatures . . . 8

3.2 Grouping time periods with similar workloads . . . 9

3.2.1 Distance calculation. . . . 9

3.2.2 Hierarchical clustering. . . . 10

3.2.3 Dendrogram cutting. . . . 10

3.3 Workload stability analysis . . . 11

3.3.1 Generating the performance vector set of each cluster . . . 11

3.3.2 Statistical analysis of performance stability . . . 11

4 Case study setup 14

4.1 Subject systems . . . 14

4.1.1 Data collection . . . 14

5 Case Study Results **17**

6 Related Work **30**

6.1 Load test reduction 30

6.2 User workload characterization 31

7 Threats to Validity **33**

7.1 Conclusion 35

Bibliography **36**

List of Figures

Figure 3.1 The overview of our approach. 7

Figure 5.1 The convergence speed of the workloads clusters. The red horizontal lines indicate the threshold of p-value (0.05) for determining statistical significance. . . 20

Figure 5.2 Prediction performance of OpenMRS. The shaded region represents the predicted performance data while the un-shaded area shows the performance data that are used to build the models (the workloads that are kept after the reduction). . . . 24

Figure 5.3 Comparison of the original performance and the predicted performance based on the replay data (after the Max-Min scaling) for OpenMRS. 29

List of Tables

Table 3.1 Our illustrative running example of execution logs and the extracted log events. 8
Table 3.2 Workload signatures of our running example. 9
Table 3.3 Performance vectors for the time periods in our running example (based on Table 3.2). 12
Table 3.4 The workload stability analysis for the workload clusters in our running example (based on Table 3.3). 13
Table 4.1 Overview of our subject systems. 15
Table 5.1 Workload reduction results for our studied systems. 19
Table 5.2 Comparing the original performance data and the performance predicted by the models built from the workloads-after-reduction. 23
Table 5.3 Comparing the performance predicted by the models that are built from the workloads-after-reduction and from the original set of workloads (i.e., the baseline). 23
Table 5.4 The hardware configurations of the original and the replay environments. . . 25
Table 5.5 Comparison between the performance data from replaying the workloads-after-reduction in the replay environment and the original performance data. Bold font indicates the best scaling methods. 28

Chapter 1

Introduction

Software performance is an essential measurement in software quality (Smith, 2006). Prior studies show that the failures of large software systems are often due to performance and load related issues rather than functional bugs (Dean & Barroso, 2013; Weyuker & Vokolos, 2000). Besides the catastrophic field failures, performance and load related software issues may also increase the cost of operations of the system and compromise the user experience. For example, due to the extraordinarily high load of online grocery shopping at the beginning of the pandemic in 2020 (Droesch, 2020 (accessed October 21, 2020)), some online purchasing systems of supermarkets crashed or had extremely slow responses (Bureau, 2020 (accessed October 21, 2020); Stevens, 2020 (accessed October 21, 2020)). Both the financial and reputational repercussions from these issues would be detrimental to the successes of software systems.

Load testing is one of the major activities for ensuring the quality of services provided by the system under load (Jiang & Hassan, 2015). However, due to the complex nature of software systems and the ever evolving user behaviours, load testing has become a challenging task. In particular, practitioners often aim to design load testing based on realistic workloads that can reflect the end users' behaviour while the software system is running in the field environment. However, these workloads are continuously evolving due to user base changes, feature changes (additions and removals), and user preference changes over time (Syer, Shang, Jiang, & Hassan, 2017). Thus, it is challenging to maintain the load test cases to reflect realistic workloads in the field. One of the most intuitive approaches to realistic load testing is to directly replay the workloads from the field (i.e.,

behaviours of real end users) in a load testing environment (Colle, Galanis, Wang, Buranawatanachoke, & Papadomanolakis, 2009).

Despite the advantages of the load testing that is driven by field-relay, practitioners still face the dilemma between realistic load tests and their costs. On one hand, the longer the duration of the replay, the more-representative the tested workloads. For example, a load test can replay a full day (24 hours) of the field workloads in order to reduce the bias caused by the variation of workloads within a day (e.g., peak workloads at a certain time of the day). However, the needed resource and time of such a lengthy replay may become an obstacle for the development of large-scale software systems, especially in a fast-paced release cycle of the modern software development process (Alghmadi, Syer, Shang, & Hassan, 2016). On the other hand, replaying a short duration of the field workloads may not suffice the goal of realistic load testing, as a short duration may not be representative of the actual field workloads.

In this thesis, we present our approach that can reduce the length of field-replay based load testing while preserving the realistic workloads. The intuition of our approach is that the lengthy field workloads typically contain repetitions in system behaviours. If we have obtained enough performance observations of the same system behaviour, we can skip the further replaying of this system behaviour to reduce the length of the field workloads. We first cluster the system behaviours, represented by the execution logs generated during system runtime, in order to identify similar user behaviours. Afterwards, we consider the stable software performance associated with the similar system behaviour as an indicator of having enough performance observations. We apply the Kolmogorov–Smirnov test (Stapleton, 2008) to determine whether adding additional testing time would have a significant influence on the distribution of the measured performance associated with each system behaviour. A statistically insignificant result of the Kolmogorov–Smirnov test is used as an indicator of stable performance. Since we only reduce the workloads if there already exist similar workloads with stable performance observations, the workloads-after-reduction are still representative of both the system behaviours and their associated performance.

We evaluate our approach on three open-source projects (i.e., *TeaStore*, *OpenMRS*, and *Apache James*) which are tested under field-like varying workloads. In particular, our study aims to answer three research questions (RQs):

RQ1: *How effectively can our approach reduce tested workloads?*

The field workloads can be drastically reduced by using our approach. Only 26%, 14% and 18% of the field workloads in *OpenMRS*, *TeaStore* and *Apache James*, respectively, are kept after reduction by our approach in the experiment. By examining the results of our experiments, we find that while the majority of the system behaviours achieve a stable performance distribution throughout in a short duration, there exist system behaviours that require a long testing time to achieve the stability.

RQ2: *How representative are the workloads-after-reduction produced by our approach?*

The workloads-after-reduction are representative of the original set of workloads. When using the workloads-after-reduction to build a performance model and use the model to predict the system performance of the entire workloads, the predicted system performance is similar to the original system performance (with a median absolute relative error lower than 6.51%). The performance model built from the workloads-after-reduction has similar prediction results to the performance model built from all the workloads with negligible effect sizes.

RQ3: *How representative are the workloads-after-reduction replayed in a different environment?*

By replaying the workloads-after-reduction, the performance of the systems in the replay environment has a high correlation with the performance of the systems under the original set of workloads. On the other hand, we encounter the challenge of using scaling methods to transform workloads and their performance data across different environments.

The evaluation results of our approach highlight the opportunities of automatically deriving and optimizing load tests of large-scale systems based on the operational data from the end users. Our results also illustrate the need for approaches that scale performance data between the operational and testing environment to better leverage the rich knowledge in the field operational data.

Thesis organization. The remainder of the thesis is organized as follows. Chapter 2 introduces the background of load testing. Chapter 3 presents our approach to reduce the length of the testing. Chapter 4 introduces the subject systems we used and how we collect the data. Chapter 5 presents the results of our case study, organized along our three RQs. Chapter 6 discusses prior research related to our work. Chapter 7 discusses the threads to the validity of our results. Finally, chapter 8 concludes the thesis.

Chapter 2

Background

Load testing is the process of assessing a system's behaviour under a workload (Jiang & Hassan, 2015). Typically, there are three phases in load testing: 1) defining a workload, 2) running a load test, and 3) analyzing the results of a load test. Load testing is a complicated and uncertain, but required process to ensure a system's quality under load (Gesvindr & Buhnova, 2016; Leitner & Cito, 2016). Prior studies propose techniques to design a proper workload (J. Chen, Shang, Hassan, Wang, & Lin, 2019; Vögele, van Hoorn, Schulz, Hasselbring, & Krcmar, 2018; P. Zhang, Elbaum, & Dwyer, 2011), determine test length (Alghmadi et al., 2016; He et al., 2019), analyze test results (Jiang, Hassan, Hamann, & Flora, 2009; Malik et al., 2010; Shang, Hassan, Nasser, & Flora, 2015; Syer et al., 2017), and detect performance issues (Ibidunmoye, Hernández-Rodriguez, & Elmroth, 2015; Xiao, Han, Zhang, & Xie, 2013). All these studies illustrate the value and importance of load testing.

One of the common approaches to conducting a load test is replaying historical field workloads. Although one may rely on the workloads that are specified in existing benchmarks for load testing, the benchmarks may not cover the unique workloads of a specific system. In addition, there exist special real-world cases where the field workload is completely different from other workloads. For example, the throughput of an online shopping system on Black Friday is much larger than the average daily workload[1]. To assess the system behaviour on the next Black Friday, the simplest approach

[1] https://www.triton.co.uk/black-friday-causes-seasonal-workload-spikes-how-did-you-cope

is to replay the exact user behaviours from the last year's Black Friday. However, such replay-based load testing is extremely time consuming and costly. For example, replaying the workloads from the previous year's Black Friday may cost at least 24 hours and many testing resources.

Prior research proposes automatic techniques to determine the length of load testing (Jiang, Avritzer, Shihab, Hassan, & Flora, 2010) or when to stop load testing (Alghmadi et al., 2016). Prior approaches are typically based on the repetitiveness of software logs (Busany & Maoz, 2016), or the naive comparison of raw performance counters (Alghmadi et al., 2016; Raz, 1992). However, prior approaches that are based on the repetitiveness of logs may not capture the performance variation of the system (e.g., caused by the variation of execution experiment) when producing similar logs. On the other hand, prior approaches that are based on comparison of raw performance counters may miss the difference of the system performance under different workloads (e.g., under spike vs. smooth workloads).

The above challenges in load testing motivate our study to not only reduce the lengthy load testing but also capture the representative system behaviours, i.e., covering diverse workloads while maintaining the accuracy of load testing results. The next chapter details our approach.

Chapter 3

Approach

In this section, we present our approach to reducing the length of load testing. The overview of our approach is shown in Figure 3.1. Our approach consists of three steps: 1) characterizing workloads, 2) grouping time periods with similar workloads, and 3) workload stability analysis.

3.1 Characterizing workloads

In order to reduce the workloads by extracting a representative subset of workloads, we first characterize workloads by system runtime behaviours. In particular, we use the execution logs that are generated during system runtime to represent the system workloads.

3.1.1 Log abstraction

Software execution logs are produced during software system execution, which usually record important system runtime behaviours. Generally, each line of execution logs contains valuable information, e.g., a log timestamp, a user event, and a server response message. We refer to the term user as any type of end user, such as IP address, email address. Such information can be used to recover workloads and then design proper load tests (Elnaffar & Martin, 2002). For example, prior work has found that events in logs are useful sources for workload recovery (Summers, Brecht, Eager, & Gutarin, 2016; Syer et al., 2017). Therefore, in this step, we parse the system execution logs to extract timestamps, user events and system responses.

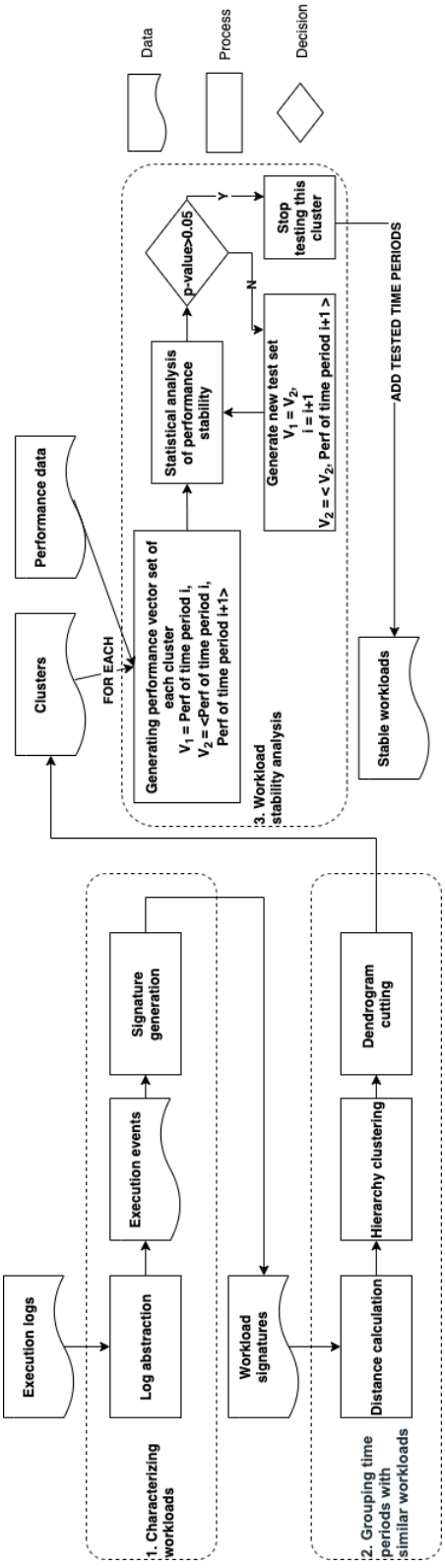


Figure 3.1: The overview of our approach.

We first extract the log timestamp of each line of execution logs. Second, we extract the user events. User events are typically a source of information to recover workloads. Table 3.1 is an illustrative example of execution logs and their corresponding log events. In our experiments, we use regular expressions to extract log events. However, in practice, one may adopt various automated log abstraction techniques (Zhu et al., 2019) for this step.

Table 3.1: Our illustrative running example of execution logs and the extracted log events.

Timestamp	Logs	Log events
00:02:00	update value a from 0 to 1 success	update success
00:05:04	search value t="jack" success	search success
00:06:17	add new value s1="hot" fail	add fail
00:07:16	add new value s2="cold" success	add success
00:11:31	update value b from 5 to "O" fail	update fail
00:59:57	update value c from 1 to 0 success	update success

3.1.2 Generating workload signatures

Workload signatures represent user behaviours in terms of their feature usage. Traditionally, one can represent a workload signature as the behaviour of one end user, or the behaviour of all aggregated users in a short period of time, e.g., 120 seconds (J. Chen et al., 2019). Since the performance of a system is mainly dependent on the workloads of aggregated users, in our study, we generate workload signatures by aggregating the log events from all users during the short period of time.

A workload signature for each time period can be represented by one data point in an n-dimensional space where n is the total number of unique log events. Each dimension represents the number of a log event in a time period.

Then, we specify the length of the time period. We find that the setting of the time period should be long enough to differentiate the workload signatures and create a representative and reliable clustering result. On the other hand, a too-long time period may cause fewer time periods to be reduced due to their higher diversity. In this thesis, we opt to use 10 minutes as the length of our time periods in order to capture more diverse workloads. Comparing to prior research (Syer et al., 2013) where 90 seconds to 150 seconds are chosen for the length of time periods, the conservative

choice of a 10 minutes time period is due to: 1) the field workloads are often longer than an in-house load testing, and 2) we want to provide a conservative evaluation result of our approach to ease its adoption in practice.

By setting the length of time periods and generating a workload signature for each time period, the entire field workloads are transformed into a time series, where each workload signature in the time series is an n-dimensional vector. Table 3.2 is an illustrative example where each workload signature is a vector of 5 dimensions. The workload signature of the time period from the beginning (0 sec) to the 600^{th} second is $< 1, 1, 1, 1, 0 >$. The entire set of workloads are represented by a time series of 6 data points.

Table 3.2: Workload signatures of our running example.

Time periods	**Log events**				
	update success	**update fail**	**search success**	**add success**	**add fail**
0 sec-600 sec	1	1	1	1	0
601 sec-1200 sec	1	2	1	2	3
1201 sec-1800 sec	2	1	2	1	1
1801 sec- 2400 sec	4	4	4	4	4
2401 sec- 3000 sec	4	4	5	5	5
3001 sec- 3601 sec	3	3	3	3	4

3.2 Grouping time periods with similar workloads

To study the performance stability of the workloads in different time periods, in this step, we apply a clustering algorithm on the time periods based on their workload signatures. Based on the clusters, we can group time periods with similar workload signatures.

3.2.1 Distance calculation.

The first step of the clustering is to calculate the distance between two time periods. We choose to use the Pearson distance(Fulekar, 2010) to calculate the cluster distance since the Pearson distance often produces a clustering that is a close match to the manually assigned clusters (Sandhya & Govardhan, 2012). The equations (1) and (2) present the calculation of the Pearson distance.

$$\rho = \frac{n\sum_i^n x_i \times y_i - \sum_i^n x_i \times \sum_i^n y_i}{\sqrt{(n\sum_i^n x_i^2 - (\sum_i^n x_i)^2) \times (n\sum_i^n y_i^2 - (\sum_i^n y_i)^2)}} \tag{1}$$

$$distance = \begin{cases} 1 - \rho & (\rho \geq 0) \\ |\rho| & (\rho < 0) \end{cases} \tag{2}$$

x_i and y_i in Equation (1) are the ith elements in the two vectors between which the distance is calculated. n is the length of the vectors.

3.2.2 Hierarchical clustering.

We apply an agglomerative hierarchical clustering to group workload signatures using a distance metrics based on the Pearson distance. We choose hierarchical clustering for the following reasons: 1) there is no need to determine the number of cluster beforehand; and 2) the hierarchical cluster result is intuitive and understandable. We then merge the most neighboring two clusters into a new cluster. Through this way, we can generate a dendrogram based on the distance.

3.2.3 Dendrogram cutting.

The result of a hierarchical clustering can be visualized using a dendrogram. Such a dendrogram must be cut at some height with a horizontal line. Each workload signature will be assigned to a cluster after cutting the dendrogram. To avoid human bias and make the cluster results more reliable, we use the Calinski-Harabasz stopping rule (Caliński & Harabasz, 1974) to cut the dendrogram. The Calinski-Harabasz index is a measure of the quality of a partition of a set of data. The Calinski-Harabasz stopping rule can often cut the dendrogram into the correct number of clusters (Milligan & Cooper, 1985). Prior research also reported that the Calinski-Harabasz stopping rule outperforms other stop rules when clustering workload signatures (Syer et al., 2017).

Applying the clustering method to our running example, we can obtain a clustering result for the workload signatures in Table 3.2. The time periods are divided into three clusters: X, Y, and Z. The time periods 0 seconds - 600 seconds, 1201 seconds - 1800 seconds and 2401 seconds - 3000 seconds belong to cluster X; the time periods 601 seconds - 1200 seconds and 3001 seconds - 3601

seconds are grouped into cluster Y. The time period 1801 seconds - 2400 seconds forms cluster Z.

3.3 Workload stability analysis

In the final step, we analyze each group of workloads from the last step to reduce the workloads with stable performance distributions.

3.3.1 Generating the performance vector set of each cluster

After clustering the workloads, the next step is to analyze the stability of the performance distributions of the workloads in each cluster. Firstly, we sort and group the performance data *P* in each time period t_x according to the time stamp to a vector $P_{tx} = < p_{x1}, p_{x2}, ..., p_{xn} >$, where each data point p_{xi} is the i^{th} recorded performance measurement in the time period. Table 3.3 shows the vector of performance data for each time period in our running example. After this step, we can obtain performance vector for each time period $S = < P_{t1}, P_{t2}, ..., P_{tn} >$ from t_1 to t_n. Then, based on the clustering result, we merge the set of the time periods belonging to each cluster. The time periods belonging to cluster x can be defined as $C_x = \{ t_{x1}, t_{x2}, ... , t_{xn} \}$. The correspond performance vector is $S_{Cx} = < P_{t_{x1}}, P_{t_{x2}}, ..., P_{t_{xn}} >$.

3.3.2 Statistical analysis of performance stability

To check the stability of each cluster's performance S_{Cx}, we start from the first two time period of each cluster. We form two performance distributions from the set S_{Cx}, which is vector $V_1 = < P_{t_{xi}} >$ and vector $V_2 = < P_{t_{xi}}, P_{t_{x(i+1)}} >$, where i starts from 1. After that, we apply the Kolmogorov–Smirnov statistical test and employ a statistical threshold of 0.05 for statistical test. The reason why we use Kolmogorov–Smirnov statistical test (Stapleton, 2008) is that we would like to examine whether the two distributions of the performance values are statistically different. A *p*-value lower than 0.05 means that V_1 and V_2 have different distributions in performance. In other words, $P_{t_{x(i+1)}}$ brings extra information to $P_{t_{xi}}$. Therefore, the performance of the corresponding cluster C_x is not stable. In contrast, if the p-value of the Kolmogorov–Smirnov test between V_1 and V_2 is larger than 0.05, the distributions of the vectors V_1 and V_2 do not have a statistically significant

difference. In other words, the performance of the cluster is stable.

If the performance of the workloads of C_x are not yet stable, we increase the value i and append another time period into the vectors V_1 and V_2. As an example, if the p-value from comparing V_1 = { $P_{t_{x1}}$ } and V_2 = { $P_{t_{x1}}$, $P_{t_{x2}}$ } is smaller than 0.05, we will further compare between V_1 = { $P_{t_{x1}}$, $P_{t_{x2}}$ } and V_2 = { $P_{t_{x1}}$, $P_{t_{x2}}$, $P_{t_{x3}}$ }. We keep increasing the value i until we observe a stable cluster of workloads, i.e., the p-value bigger than 0.05, or until all the data in P_{t_x} has been included in the comparison. We only keep the time periods in V_1 in the load tests for cluster C_x, while the rest time periods in the cluster will be excluded from the load tests.

We repeat the above process for every cluster. Finally, for all the time periods that are kept in all the clusters for load testing, we merge them together and sort them by their time stamps, to make the final workloads for the load testing. For example, we used the data from Table 3.3 to perform the workload stability analysis, and the result is shown in Table 3.4. For Cluster X, the performance distribution is not stable in the first comparison and it becomes stable in the second comparison. Cluster Y achieves a stable performance in the first comparison. For cluster Z, because it only has one time period, it does not have a stable performance. As a result, the first two time periods of Cluster X, the first time period of Cluster Y, and the only time period of Cluster Z are included in our load testing after reduction.

Table 3.3: Performance vectors for the time periods in our running example (based on Table 3.2).

Time periods (Cluster)	**CPU utilization**			
	t1	**t2**	**t3**	**t4**
0 sec-600 sec (X)	2%	1%	1%	1%
601 sec-1200 sec (Y)	33%	37%	41%	46%
1201 sec-1800 sec (X)	25%	20%	24%	34%
1801 sec-2400 sec (Z)	56%	47%	58%	23%
2401 sec-3000 sec (X)	23%	27%	2%	30%
3001 sec-3600 sec (Y)	34%	37%	43%	45%

Note: t1 to t4 indicate the recorded performance data during the 600-second time period, i.e., one recorded performance data per 150 seconds.

Table 3.4: The workload stability analysis for the workload clusters in our running example (based on Table 3.3).

Cluster	Time period A	Time period B	*p*-value	Stable?
X	0 sec-600 sec	0 sec-600 sec, 1201 sec-1800 sec	0.04	False
	0 sec-600 sec, 1201 sec-1800 sec	0 sec-600 sec, 1201 sec-1800 sec, 2401 sec-3000 sec	0.99	True
Y	601 sec-1200 sec	601 sec-1200 sec, 3001 sec-3600 sec	0.99	True
Z	1801 sec-2400 sec	N/A	N/A	False

Chapter 4

Case study setup

In this chapter, we present the setup of our case study.

4.1 Subject systems

We choose three open-source systems including *OpenMRS*, *Apache James*, and *TeaStore* as our subject systems. *OpenMRS* is a web system designed to support customized medical health care. *Apache James* is a Java-based mail system developed by the Apache Foundation. *TeaStore* (von Kistowski et al., 2018) is a basic web store for tea and tea supplies, which is a microservice-based test and reference application. All our subject systems have been studied in prior research (J. Chen et al., 2019; T. Chen, Shang, Hassan, Nasser, & Flora, 2016; Gao, Jiang, Barna, & Litoiu, 2016). The overview of the three subject systems is shown in Table 4.1.

4.1.1 Data collection

In this subsection, we describe our approaches for collecting system execution logs and performance data from the studied systems. In this work, we focus on the CPU usage performance, as the studied systems are CPU-intensive. Nevertheless, our approach could apply to other performance metrics (e.g., response time). In particular, we first deployed the systems in our experimental environment and conducted load tests to exercise the systems for an extended period of time. All the subject systems we studied are deployed on the Google Cloud Platform Compute Engine (*Compute*

Table 4.1: Overview of our subject systems.

Subjects	Version	SLOC (K)	# Users	# Lines of logs (K)
Apache James	2.3.2.1	37.6	2000	458
OpenMRS	2.0.5	67.3	1000	3019
TeaStore	1.3.4	29.7	99	4502

Engine: Virtual Machines (VMs) — Google Cloud, n.d.) with three separate virtual machines. Afterwards, we collected system execution logs and performance data during the system execution. For the system performance (i.e., CPU usage), we use the tool *Pidstat* (*pidstat(1): Report statistics for tasks - Linux man page*, n.d.) to monitor the process of the system for every ten seconds. We detail our data collection for each of our subject systems below. The details of our data can be found in our replication package[1].

OpenMRS: We setup our *OpenMRS* system with the *OpenMRS* demo database version 2.2.1 (*Demo Data - Resources - OpenMRS Wiki*, n.d.) in our load tests. The demo database contains various data for 5,000 patients. *OpenMRS* contains four typical requests: addition, deletion, search, and edition. We designed our load tests that are composed of various searches of patients, concepts, and observations, as well as addition, deletion, and edition of patient records.

We deployed *OpenMRS* on two virtual machines, each with 4-core vCPU, 15GB RAM, and 24GB persistent disk. One machine is deployed as the application server and the other machine as the MySQL server with the demo data. *OpenMRS* provides RESTFul services. Therefore, we used the RESTFul API of *OpenMRS* to simulate users sending requests to the application server. In particular, we used JMeter to perform a twenty-six hours duration of workloads to collect the system execution logs.

Apache James: We used JMeter to create load tests to exercise the *Apache James* server. We replicate the similar workloads as a prior study (Gao et al., 2016). In detail, we simulated 2,000 email users who send and receive different sizes of emails, with or without small and large sizes of attachments. In addition, we simulated the scenarios of users reading the email header or loading the entire email.

We deployed *Apache James* in a server machine with 2-core vCPU, 7.5 GB memory on a 1 TB persistent disk. We run JMeter on another machine with 4-core vCPU, 8GB memory and 24GB

[1] https://t.ly/lIJb

persistent disk. Finally, we execute one-day long workloads to load test the *Apache James* server using JMeter.

TeaStore: *TeaStore* has a few quintessential use cases, including login system, browsing the store, browsing user's profile, browsing products, shopping products, and logging out the system.

The experiments on *TeaStore* are performed with three separate virtual machines. These virtual machines have the same hardware configurations, including 4-core vCPU, 8GB memory, and 24GB persistent disks. We deployed the *TeaStore* web application and database on the first and second machines, respectively, while the third machine is used to run the JMeter load driver with varying workloads to simulate users operating the system with the above-mentioned use cases.

Chapter 5

Case Study Results

In this chapter, we present the case study results by answering our three research questions (RQs).

RQ1: How effectively can our approach reduce tested workloads?

Motivation

In order to achieve realistic workloads in load testing, practitioners often conduct load tests by simply replaying the field workloads that are obtained from the real usage scenarios of end users. However, as discussed in Chapter 2, determining the length of the field workloads is challenging. A set of workloads with too-small size may not contain representative workloads, while a too-large set of workloads would cause the load testing to be very expensive and may delay the release schedule of the software system, especially in a fast-paced release cycle (Jiang & Hassan, 2015). Therefore, in this RQ, we would like to examine how effective our proposed approach is in reducing the length of the load tests.

Approach

We apply our approach (cf. Chapter 3) on the three datasets obtained from our experiments on the studied subject systems. In particular, the datasets contain the logs and the performance metrics (CPU usages) that are collected when the subject systems are executed under random and varying

workloads. We consider these three datasets as the source of the system replay, i.e., the input of our approach. After applying our approach, we generate a new set of workloads which reduce the length of the original set of workloads. Therefore, we first measure the size of the reduction, i.e., how much shorter (in minutes) the new set of workloads is, compared with the original set of workloads.

To further understand the effectiveness of our approach, we calculate the total number of clusters of workloads and the number of workloads that achieve stable results after applying our approach, and the number of workloads that cannot achieve stable results. The more workloads that can achieve stable results, the more promising our approach is in practice.

Results

Our approach can effectively reduce the length of the original load testing workloads. Table 5.1 shows the results of our approach for reducing the performance testing workloads on our studied subject systems. We find that by applying our approach, the length of the load testing of our studied subject systems can be significantly reduced compared to the original set of workloads. In particular, for *TeaStore*, the time length of the workloads after reduced by our approach is only 420 minutes, whereas the original set of load testing workloads requires more than two days (3020 minutes) of execution (i.e., the reduction rate is 86%). When we compare the total number of clusters of workloads with the number of workloads that achieve stable results after applying our approach, we observe that the majority of the clusters of workloads can be reduced by our approach. For example, for *Apache James*, 85% of the clusters of workloads can be further reduced by our approach. For those clusters that cannot be reduced our approach, we consider the reason being size of those clusters, i.e., there are only one or two time periods in those clusters, which are difficult to further reduce.

When the system is under random and varying workloads, simply reducing the length of load test workloads by time can miss representative workloads. Figure 5.1 presents the convergent speed of each cluster, when applying our approach for reducing performance testing workloads on our studied subject systems, respectively. Each line in the figure shows p-values of each cluster of workloads during workload stability analysis (cf. Section 3.3), where dot above the red line (the threshold of p-value at 0.05) indicates that performance of the cluster of workloads is stable.

From those figures, we observe that some of the lines have quite low slopes, which means that the workloads cannot achieve a stable performance distribution throughout. For example, there is a flat line in both the figures of *TeaStore* and *Apache James* where only at the time around 2,600 and 1310 minutes, respectively, the workloads becomes stable, where the p-values for the clusters in *Teastore* and *Apache James* are 0.22 and 0.16, respectively. Such results also indicate that the length of the load testing workloads cannot be simply reduced by cutting down the time of the original set of workloads. If we simply reduce the length of original set of load testing workloads by time, for example, only keep a few hours at the beginning, some important field workloads would be missing and it would be hard to achieve stable performance.

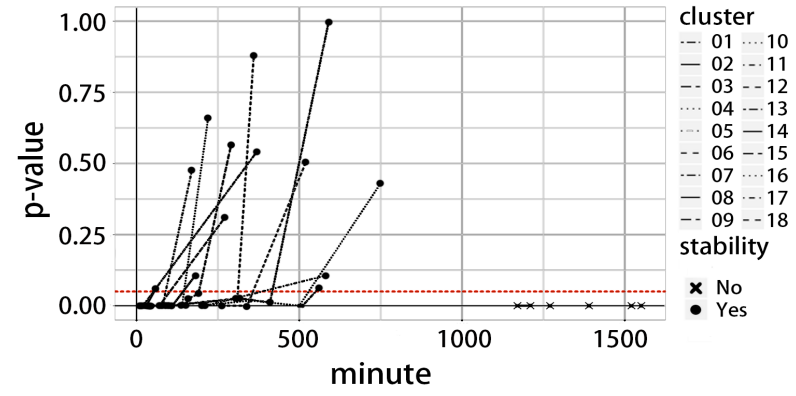
Table 5.1: Workload reduction results for our studied systems.

Project	**Time length (minutes)**		**# clusters**	
	Before reduction	**After reduction**	**# total clusters**	**# stable clusters**
OpenMRS	1,600	420	18	13
TeaStore	3,020	420	20	13
Apache James	1,440	260	14	12

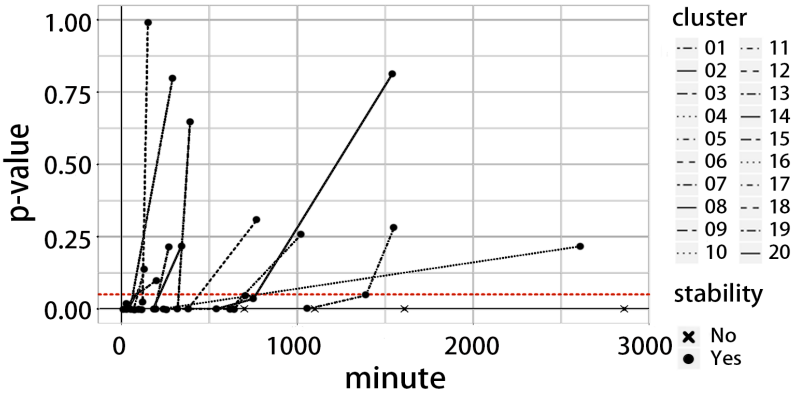
RQ2: How representative are the workloads-after-reduction produced by our approach?

Motivation

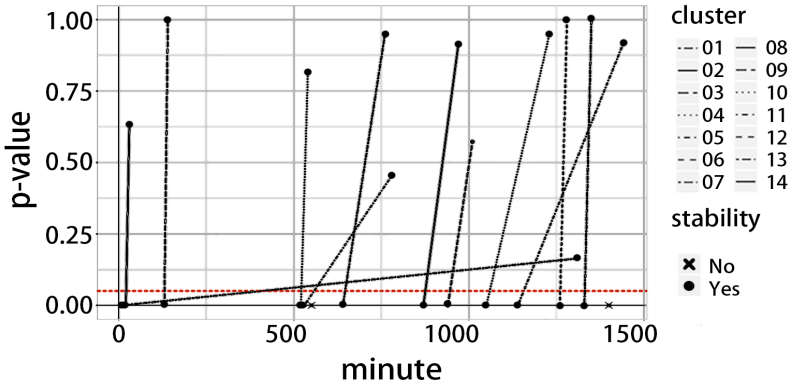
RQ1 shows that our approach can effectively reduce the field workloads into much shorter versions. However, if a workloads-after-reduction is not representative of the original set of workloads, the reduction from our approach is meaningless, since it would lead to unrealistic load tests, i.e., the testing workloads cannot represent the actual workloads from the end users in the field. Thus, the goal of this RQ is to assess the representativeness of the workloads-after-reduction that are generated by our approach.



OpenMRS



TeaStore



Apache James

Figure 5.1: The convergence speed of the workloads clusters. The red horizontal lines indicate the threshold of p-value (0.05) for determining statistical significance.

Approach

In order to assess the representativeness of the workloads-after-reduction, we examine whether we can use the workloads-after-reduction to extrapolate the original set of workloads before reduction. In particular, we build a performance model using only the data from the workloads-after-reduction that are generated by our approach, and use the model to predict the performance of the system under the original set of workloads. We call this model M_r in the rest of this chapter.

Measuring the performance model fit. We first evaluate the quality of M_r using the model fit. If M_r has a poor model fit, we cannot trust the data produced by this model, i.e., the workloads-after-reduction by our approach do not have the capability to model the performance of the software system. In particular, we construct M_r by training on the data that is in the workloads-after-reduction by our approach. To evaluate the model fit of M_r, we first apply M_r on the data that is in the original set of workloads but not in the workloads-after-reduction, and then calculate the *median absolute relative error* (MARE) which is used as a measurement for the model fit. Smaller values of MARE indicate better prediction accuracy.

Comparing the predicted and the actual system performance. In addition, we compare the system performance predicted by M_r with the actual observed system performance. Similarly, we apply M_r on the data that is in the original set of workloads but not in the workloads-after-reduction. If the workloads-after-reduction are representative of the original set of workloads, M_r should be able to predict the system performance based on the workloads that is in the original set of workloads. We perform statistical analysis to examine the deviation between the predicted and observed performance, in terms of the CPU usage. Specifically, we calculate the Pearson correlation(Fulekar, 2010) to measure the relationship between the predicted values generated by M_r and the observed performance.

Comparing the prediction error with a baseline. To further understand the representativeness of the performance model that is built from the workloads-after-reduction (M_r), we compare its prediction error with a baseline, i.e., a performance model that is built using all the original set of workloads, i.e., M_o. Our intuition is that if M_r is as good as M_o, we would be able to consider that the workloads-after-reduction have the same capability of modeling system performance as the

original set of workloads. Therefore, the workloads-after-reduction can be considered representative. In order to comprehend the difference between the two models(i.e., the model of baseline and M_r), we use the Kolmogorov–Smirnov test (Stapleton, 2008) to determine if there exists a statistically significant difference (i.e., p-value $<$0.05) between the prediction performance of baseline and M_r. We choose the Kolmogorov–Smirnov test because it does not enforce any assumptions on the distributions of the data. Reporting only the statistical significance may lead to erroneous results (i.e., if the sample size is very large, p-value can be small even if the difference is trivial). Thus, we further use Cohen's D (J. Cohen, 2009) to quantify the effect size between the predictions of two models. Through the statistical analysis, we can have a clear view of the differences between the error distributions of the two models.

In particular, for M_r, we train this model using the workloads-after-reduction, and apply the model on the data of workloads that is removed by our approach, i.e., data in the original set of workloads but not in the workloads-after-reduction. However, for M_o, we cannot directly calculate the prediction error, since applying a model to its training data leads to biased (overly optimized) results. To address this issue, we apply the throw-one approach that is used in prior research (Liao et al., 2020). For each time period in the original set of workloads, we remove its data from the training data to rebuild the model and apply the rebuilt model on the time period. We repeat the process until all time periods are used as test data once.

Results

The workloads after our approach's reduction can effectively represent the original set of workloads in terms of the corresponding performance. Table 5.2 presents the median relative error of the model built on workloads-after-reduction and the Pearson correlation between the original performance data and the predicted values. We find that for all the subject systems, after applying our approach, the model M_r built on the data from the workloads-after-reduction are of high quality, which achieves a median relative error of 6.51%, 4.37%, and 3.63%, respectively, for system performance prediction. Moreover, the relatively high Pearson correlations (0.91, 0.83 and 0.58) between the predicted values generated by M_r and the original performance data also show the representativeness of the workloads-after-reduction generated by our approach.

Table 5.2: Comparing the original performance data and the performance predicted by the models built from the workloads-after-reduction.

Project	MARE	Correlation
OpenMRS	6.51%	0.91
TeaStore	4.37%	0.83
Apache James	3.63%	0.58

Table 5.3: Comparing the performance predicted by the models that are built from the workloads-after-reduction and from the original set of workloads (i.e., the baseline).

Project	*p*-value	Cohen's D
OpenMRS	$\ll 0.001$	-0.18(negligible)
TeaStore	$\ll 0.001$	-0.11(negligible)
Apache James	0.38	-0.12(negligible)

The comparison results of the prediction error distributions between the performance model built on the workloads-after-reduction and the baseline model (i.e., model built using the original set of workloads) are shown in Table 5.3. The prediction errors of all three subject systems have either statistically insignificant or negligible differences between the two models (i.e., M_o and M_r), indicating that the workloads-after-reduction generated by our approach has the same capability of modeling system performance as the original set of workloads.

In addition, Figure 5.2 presents the trends of both the original and predicted performance data over time, in which we can have a clear view of how representative the workloads-after-reduction are. In particular, the shaded region represents the predicted performance data while the unshaded area shows the performance data that are used to build models which is also during the period of the workloads-after-reduction. Through the graph, we can observe that the trends of the original performance data and the prediction data are similar. For the purpose of comparing the prediction effects, we present two lines representing two prediction methods, i.e., M_r and the baseline. Although the baseline method is closer to the original data, the trend of predicted performance between the baseline method and M_r is similar. Such results also indicate the strong representativeness of the workloads-after-reduction generated by our approach for the original set of workloads. Due to space limitations, here we only provide the example of one system i.e., *OpenMRS*, the run charts of the other two subject systems are included in our replication package.

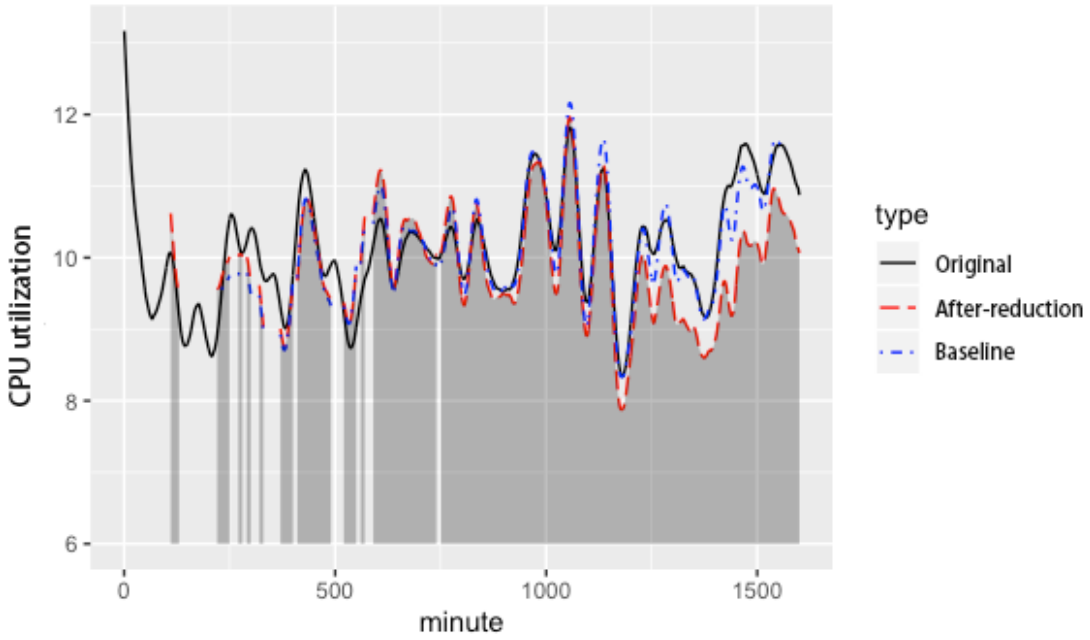


Figure 5.2: Prediction performance of OpenMRS. The shaded region represents the predicted performance data while the un-shaded area shows the performance data that are used to build the models (the workloads that are kept after the reduction).

RQ3: How representative are the workloads-after-reduction replayed in a different environment?

Motivation

In the previous RQ, we find that, with our approach, we can effectively use the workloads-after-reduction generated by our approach to extrapolate the original varying workloads. Such results could show that the workloads-after-reduction are representative of the original varying workloads, both of which are captured in the same field environment. However, in practice, the original set of workloads of end users are typically extracted from the field environments; while the load tests that replay the workloads are often conducted in a testing environment. If the workloads-after-reduction by our approach are sensitive to the runtime environment configuration, it may not be suitable to be used to replace the original performance testing workloads in practice. Therefore, the goal of this RQ is to examine whether the replaying results using the workloads-after-reduction generated by our approach in a different environment is still representative of the original workloads from the original field environment.

Table 5.4: The hardware configurations of the original and the replay environments.

Project	Hardware configuration	
	Original	**Replay**
OpenMRS	4vCPU, 15GB memory	8vCPU, 30GB memory
TeaStore	4vCPU, 8GB memory	8vCPU, 30GB memory
Apache James	2vCPU, 7.5GB memory	4vCPU, 30GB memory

Approach

To answer this research question, we re-deploy our studied subject systems in a new environment as the replay environment. The details of the difference of configuration between the replay environment and the original performance testing environment for all subject systems are shown in Table 5.4.

We generate load tests based on the workloads-after-reduction and use *JMeter* load test driver to replay the workloads-after-reduction. While testing, we collect the performance metrics (e.g., CPU) for every ten seconds by *Pidstat* (*pidstat(1): Report statistics for tasks - Linux man page*, n.d.). When replaying the workloads-after-reduction finished, we retrieve the execution logs from the web servers (e.g., Tomcat), which are used to provide the web server environment.

Similar to RQ2, we build performance models based on the replay of the workloads-after-reduction in the replay environment. We name this performance model M_{er}. We use M_{er} to predict performance of the original workloads in the original environment without reduction. By calculating the deviance of the predicted values and the actual performance at runtime, we can have a clear view of how our approach performs when replaying the workloads-after-reduction in a new environment. However, since there exist differences between the hardware configurations of the replay environment and the original environment, we would not directly compare the predicted performance metrics and the measured performance metrics. To better compare the two performance data distributions generated under different environments, we leverage the following scaling approaches:

- **Max-Min** scaling approach is a normalization method bringing all value into [0, 1] as the ratio of the value in the range between maximum and minimum. The formula of the approach is $Px_{scaled} = \frac{Px - Min(P)}{Max(P) - Min(P)}$, which the *Px* is the x^{th} value in the *P* is the vector needed to scale.

- **Median** scaling approach is used in prior research(Arif, Shang, & Shihab, 2018) to reduce the bias caused by different environment configurations. The result of the scaling is the ratio between the distance to the median and the value of median absolute deviation. In particular, the scaling follows the formula $Px_{scaled} = \frac{Px - Median(P)}{MAD(P)}$, where the *MAD* is the median absolute deviation of vectors.

- **Scaling by modeling** utilizes linear regression models to model the relationship between each log event's frequency and the system performance. We hypothesis that the R_α and R_β are the same dependent variable (i.e., CPU utilization) in two different datasets (i.e., the set of workloads in original environment and replay environment) while the α and β is the independent metrics. Then, the coefficient k_α, k_β and intercept h_α, h_β from $R_\alpha = k_\alpha \cdot \alpha + h_\alpha$ and $R_\beta = k_\beta \cdot \beta + h_\beta$. Finally, the normalize metric will be $\alpha_{Normalize} = \frac{k_\alpha \cdot \alpha + h_\alpha - h_\beta}{k_\beta}$. With transforming each independent metrics dimension, we can finally obtain values of the dependent variable with the same dimension. This approach is adopted from the work of Nguyen et al. (Nguyen et al., 2012).

- **Robust** scaling method is an advanced version of the median scaling method. The median absolute deviation is replaced by inter-quartile range (i.e., IQR), which is $Px_{scaled} = \frac{Px - Median(P)}{IQR(P)}$. IQR can be explained as the differences between the 25th percentile and the 75th percentile. The formula shows that the method receives less influence from the outlier and may ignore more information.

- **Quantile** scaling method uses the rank of the value in each metric. Firstly, through ranking, we can obtain the ranks of the values. Secondly, we can calculate the average value of the values that have the same rank in all vectors. Finally, the original value will be replaced by the average calculated. This method is widely used in cross-project modeling in software engineering (F. Zhang, Keivanloo, & Zou, 2017).

The implementation details of the scaling methods are included in our replication package.

We scale the prediction data based on M_{er} by applying the scaling approaches. Based on the scaled data, we measure the median relative error of M_{er} which is calculated as the difference

between the predicted performance and the measured performance, normalized by the measured performance. For example, by using Max-Min scaling method, we scale both two performance vectors(i.e., original performance, predicted performance based on model M_{er}). Then we can obtain the scaled data of these two vectors in the range from zero to one. Also, we calculate the Pearson correlation of the original performance and predicted performance value generated by model M_{er} to further capture the relationship of the original set of workloads and the replayed workloads after reduced by our approach in the different environments.

Results

The performance data from the replayed workloads is representative of the original set of workloads. Table 5.5 presents the median relative error of the performance models based on the re-playing the workloads-after-reduction in the new load testing environment and the Pearson correlation between predicted performance value and actual performance during the original execution. Figure 5.3 presents the trend of scaled original performance value and predicted data over time in *OpenMRS*. We can observe that there are strong correlations, ranging from 0.42 to 0.83, between the predicted performance value generated by model M_{er} and the measured performance data. The results indicate that the replaying results using the workloads-after-reduction generated by our approach from a different environment is still representative of the original workloads from the original field environment.

Future work on scaling the performance data from different environments is needed. Table 5.5 shows how the different scaling methods affect the quality of the performance model in terms of the median absolute relative prediction error. We find that, when re-playing the reduced workloads under a different environment, using all the selected scaling approaches, the performance model still has a relatively high MARE, ranging from 11.81% to 91.71%; while under the same environment, the maximum MARE is only 6.51% (cf. Table 5.2). Such results indicate the limitations of the current scaling approaches for analyzing performance data from different environments. Among all the scaling methods, we find that the simple Max-Min scaling achieves the best results, which may be explained by the fact that the Max-Min approach reserves the range and the linear relationship in the original data.

As performance data generated in the field contains many valuable information about how the system behaves in production, many field performance data are being analyzed by performance engineers to understand the system performance, e.g., detecting performance regressions (Liao et al., 2020). However, in these cases, the testing environment are often not completely identical to the production one, and if there is no optimal scaling approach to eliminate the bias from different environment configurations, performance data obtained in the field may be difficult to be used properly in a more reasonable way. Therefore, our findings also advocate the need for future research on how to better scale the performance data from different environments to reduce the bias caused by configuration differences.

Table 5.5: Comparison between the performance data from replaying the workloads-after-reduction in the replay environment and the original performance data. Bold font indicates the best scaling methods.

Project	**Correlation**	**MARE after scaling**				
		Max-Min	**Median**	**Model**	**Robust**	**Quantile**
OpenMRS	0.83	18.22%	52.84%	**11.81%**	42.66%	16.83%
Teastore	0.81	**19.57%**	52.52%	62.56%	78.94%	31.64%
Apache James	0.42	**23.17%**	91.71%	60.21%	90.26%	43.57%

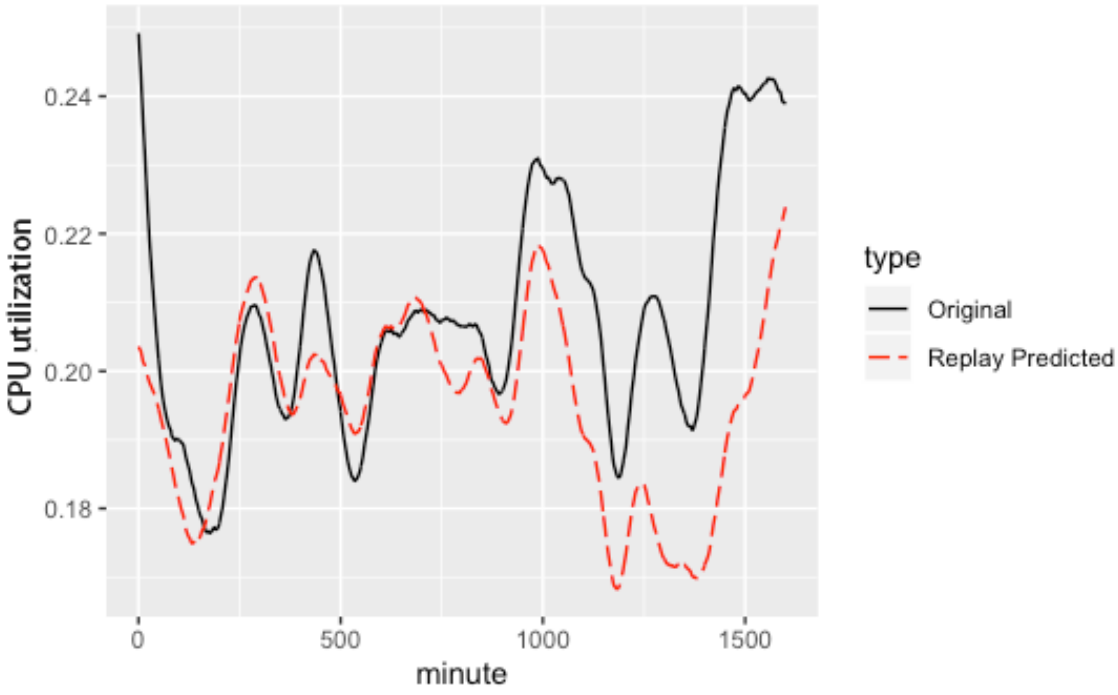


Figure 5.3: Comparison of the original performance and the predicted performance based on the replay data (after the Max-Min scaling) for OpenMRS.

Chapter 6

Related Work

In this chapter, we present the prior research related to our work.

6.1 Load test reduction

Several prior studies (Alghmadi et al., 2016; He et al., 2019; Schulz, Angerstein, & van Hoorn, 2018) share a similar goal to our work. He et al. (He et al., 2019) applied a statistics-based approach to investigate whether the distribution of a performance metric varies after the execution of one part of testing. Hammam et al. (Alghmadi et al., 2016) focused on searching the stop point of the performance testing. They used statistical methods to measure whether the performance metrics are repetitive during the testing. Jain (Jain, 1991) proposed to find the stop point of performance testing through applying a 5% threshold of the variance in response time. Daly et al. (Daly, Brown, Ingo, O'Leary, & Bradford, 2020) apply the Q statistic to detect changing point in testing for acknowledge the performance of the system. Busany et al. (Busany & Maoz, 2016) and Jiang et al. (Jiang et al., 2010) presented approaches of how to reduce the execution time of tests by tracking repetitive log traces. Approaches are also proposed to dynamically adapt the execution time of the load testing time (Ayala-Rivera, Kaczmarski, Murphy, Darisa, & Portillo-Dominguez, 2018; Shivam, Marupadi, Chase, Subramaniam, & Babu, 2008; Tchana et al., 2013). Apte et al. (Apte, Viswanath, Gawali, Kommireddy, & Gupta, 2017) arranged load testing by building a queueing model to achieve a higher effectiveness. These prior studies either consider the performance of the systems or their

workloads without building an association between the workloads and the performance. In our thesis, we consider both the workloads and the performance of various workloads to reduce the length of load testing.

There are several prior studies in load testing field that aim to reducing the system resources during testing a given workload. Shariff et al. (Shariff et al., 2019) demonstrated how they optimize system resources by running a browser-based load test with Selenium[1]. In their approach, the simulated users could share the browser instance, so that the total number of the browser instances decreased, which can improve the efficiency of load testing. Grano et al. (Grano, Laaber, Panichella, & Panichella, 2019) focused on generating performance-sensitive functional test suite with high coverage and low requirements on system resources. These approaches mainly tackle the problem about reducing the testing resources of load testing, while our approach focuses on producing reduced workloads that are representative of the original workloads in terms of performance measurements.

6.2 User workload characterization

The prior research from Cohen et al. (I. Cohen et al., 2005) emphasized the demand for considering varieties in system workload recovery. Specifically, they observed that it is inefficient to detect and identify system issues only by using the general recording of raw system metrics and to tackle such problem. Cohen et al. proposed an approach by clustering the system signature and the results show that the efficiency of issue detection can be improved by utilizing the clustering results. Later work followed Cohen et al.'s approach and applied it to large-scale systems. For example, Syer et al. (Syer et al., 2013) clustered the high viability users in a large software system to obtain the system workloads by counting the frequency of the log events associated with each user. In addition, instead of focusing on execution logs of the system, Shang et al. (Shang et al., 2015) utilized the physical performance metrics, like CPU and memory usage. In particular, Shang et al. clustered the performance metrics directly to capture the diversity and complexity in system workloads of large-scale systems. Motivated by prior work, our study also utilize clustering algorithms on system

[1] https://www.selenium.dev

workload signatures. However, our approach is different as we do not distinguish users in execution logs so that we can have a high-level view of the system performance.

To obtain a further understanding of the usage of system resources, workload recovery is a necessary step in load testing. Alireza et al. (Haghdoost, He, Fredin, & Du, 2017) implemented an I/O workloads replay tool named hfplayer and it aims to infer I/O dependencies and assist I/O performance evaluation. Neeraja et al. (Yadwadkar, Bhattacharyya, Gopinath, Niranjan, & Susarla, 2010) proposed to use the Profile Hidden Markov Models to analyze system workloads. Based on the patterns in the traces, this approach can classify workload patterns in a long sequence of NFS trace. Axel et al. (Busch et al., 2015) proposed an automatic workload characterization approach for I/O-intensive software in a virtual environment. Bumjoon et al. (Seo et al., 2014) defined twenty I/O related metrics to generate I/O workload signatures and clustered the I/O workloads. Eli et al. (Cortez et al., 2017) characterized Microsoft Azure's VMS workloads based on the VMs' size and lifetime.

Prior research mainly analyze physical performance metrics to recover workloads. In comparison, in our work, we consider the system performance metrics that are associated with the detailed events from users that are extracted from system execution logs. Our approach can complement existing approaches by combining user behaviours and the system performance to improve the effectiveness of system workload recovery. As a result, our approach are easier to be integrated into Dev-Ops(Bezemer et al., 2019).

Chapter 7

Threats to Validity

In this chapter, we discuss the threats to the validity of our findings.

External Validity. The subject systems used in our case study are three prevalent open-source systems (*OpenMRS*, *Apache James*, and *TeaStore*). These systems all have a long development history and have been studied in prior research (J. Chen et al., 2019; Liao et al., 2020). However, our subject systems may not represent all the software domains and our approach and results may not directly be applied to other systems. Future work may investigate the applicability in different systems.

Internal Validity. In our study, we build a prediction model to capture the relationship between the workloads and the system performance. The source of the workloads is the execution logs of the system. However, some runtime activities that have an impact on the system performance may not be recorded in the execution logs. Therefore, under such circumstances, the prediction accuracy of the performance model would be impaired. In addition, we do not consider the sequence of the actions during the workload signature generation, which may cause different workloads sorted into the same cluster. The design of the workload signatures is a direction of our future research.

Construct Validity. We use a traditional performance monitoring tool, *pidstat*, to collect the system runtime performance, instead of using a modern performance monitoring tool (e.g., application performance monitoring tools). Applying those tools may enhance the accuracy of the performance measurement. However, such systems may introduce more overhead to the monitored system. In our study, we only consider the CPU usage aspect of the system performance. Although CPU usage

is a main performance metric that reflects the system performance, other physical metrics (e.g., memory usage) are also important. Nevertheless, our approach can also apply to other performance metrics. Future work may extend our evaluation by considering other performance metrics.

7.1 Conclusion

In this thesis, we propose an automated approach to reducing the length of the field workloads that are used to drive load testing. By examining the stability of the system performance that is associated with similar system behaviours, our approach skips the execution of the workloads if the corresponding performance achieves a stable distribution. By evaluating our approach on three open-source systems, we find that our approach can significantly reduce the length of workloads for load testing while preserving the workloads that are representative of the entire original workloads. By replaying the workloads-after-reduction in a different load testing environment, we observe that the performance of the system has a high correlation to the performance from the original execution. This thesis provides the following contributions:

- We propose an approach that can automatically reduce the length of field-replay based load testing by skipping similar workloads with stable performance.
- Our approach can be leveraged in the replay of field workloads in a testing environment while significantly reducing the costs of such replay-based load tests.
- Our work sheds light on future work that leverages and optimizes the field workloads for cost-effective performance testing.
- We highlight the challenges of applying existing scaling methods to normalize the performance data produced in different environments (e.g., field vs. testing environments) and call for future work to address such challenges.

References

Alghmadi, H. M., Syer, M. D., Shang, W., & Hassan, A. E. (2016). An automated approach for recommending when to stop performance tests. In *2016 IEEE international conference on software maintenance and evolution, ICSME 2016, raleigh, nc, usa, october 2-7, 2016* (pp. 279–289). IEEE Computer Society.

Apte, V., Viswanath, T. V. S., Gawali, D., Kommireddy, A., & Gupta, A. (2017). Autoperf: Automated load testing and resource usage profiling of multi-tier internet applications. In *Proceedings of the 8th ACM/SPEC on international conference on performance engineering, ICPE 2017, l'aquila, italy, april 22-26, 2017* (pp. 115–126). ACM.

Arif, M. M., Shang, W., & Shihab, E. (2018). Empirical study on the discrepancy between performance testing results from virtual and physical environments. In *Proceedings of the 40th international conference on software engineering, ICSE 2018, gothenburg, sweden, may 27 - june 03, 2018* (p. 822). ACM.

Ayala-Rivera, V., Kaczmarski, M., Murphy, J., Darisa, A., & Portillo-Dominguez, A. O. (2018). One size does not fit all: In-test workload adaptation for performance testing of enterprise applications. In *Proceedings of the 2018 ACM/SPEC international conference on performance engineering, ICPE 2018, berlin, germany, april 09-13, 2018* (pp. 211–222). ACM.

Bezemer, C., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., ... Willnecker, F. (2019). How is performance addressed in devops? In *Proceedings of the 2019 ACM/SPEC international conference on performance engineering, ICPE 2019, mumbai, india, april 7-11, 2019* (pp. 45–50). ACM.

Bureau, E. (2020 (accessed October 21, 2020)). Bigbasket app, site crash on high demand [Computer software manual]. Retrieved from https://economictimes.indiatimes.com/small-biz/startups/newsbuzz/bigbasket-app-site-crash-on-high-demand/articleshow/74784753.cms?utm_source=contentofinterest&utm_medium=text&utm_campaign=cppst

Busany, N., & Maoz, S. (2016). Behavioral log analysis with statistical guarantees. In *Proceedings of the 38th international conference on software engineering, ICSE 2016, austin, tx, usa, may 14-22, 2016* (pp. 877–887). ACM.

Busch, A., Noorshams, Q., Kounev, S., Koziolek, A., Reussner, R. H., & Amrehn, E. (2015). Automated workload characterization for I/O performance analysis in virtualized environments. In *Proceedings of the 6th ACM/SPEC international conference on performance engineering, austin, tx, usa, january 31 - february 4, 2015* (pp. 265–276). ACM.

Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, *3*(1), 1–27.

Chen, J., Shang, W., Hassan, A. E., Wang, Y., & Lin, J. (2019). An experience report of generating load tests using log-recovered workloads at varying granularities of user behaviour. In *34th IEEE/ACM international conference on automated software engineering, ASE 2019, san diego, ca, usa, november 11-15, 2019* (pp. 669–681). IEEE.

Chen, T., Shang, W., Hassan, A. E., Nasser, M. N., & Flora, P. (2016). Cacheoptimizer: helping developers configure caching frameworks for hibernate-based database-centric web applications. In *Proceedings of the 24th ACM SIGSOFT international symposium on foundations of software engineering, FSE 2016, seattle, wa, usa, november 13-18, 2016* (pp. 666–677). ACM.

Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., & Fox, A. (2005). Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the 20th ACM symposium on operating systems principles 2005, SOSP 2005, brighton, uk, october 23-26, 2005* (pp. 105–118). ACM.

Cohen, J. (2009). *Statistical power analysis for the behavioral sciences*. Psychology Press.

Colle, R., Galanis, L., Wang, Y., Buranawatanachoke, S., & Papadomanolakis, S. (2009). Oracle

database replay. *Proc. VLDB Endow.*, *2*(2), 1542–1545.

Compute engine: Virtual machines (vms) — google cloud. (n.d.). https://cloud.google.com/compute. ((Accessed on 9/3/2020))

Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., & Bianchini, R. (2017). Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th symposium on operating systems principles, shanghai, china, october 28-31, 2017* (pp. 153–167). ACM.

Daly, D., Brown, W., Ingo, H., O'Leary, J., & Bradford, D. (2020). The use of change point detection to identify software performance regressions in a continuous integration system. In *ICPE '20: ACM/SPEC international conference on performance engineering, edmonton, ab, canada, april 20-24, 2020* (pp. 67–75). ACM.

Dean, J., & Barroso, L. A. (2013). The tail at scale. *Commun. ACM*, *56*(2), 74–80.

Demo data - resources - openmrs wiki. (n.d.). https://wiki.openmrs.org/display/RES/Demo+Data. ((Accessed on 10/3/2020))

Droesch, B. (2020 (accessed October 21, 2020)). Five charts: How coronavirus has impacted digital grocery [Computer software manual]. Retrieved from https://www.emarketer.com/content/five-charts-how-coronavirus-has-impacted-digital-grocery

Elnaffar, S., & Martin, P. (2002). Characterizing computer systems' workloads..

Fulekar, M. H. (2010). *Bioinformatics: applications in life and environmental sciences*. Springer.

Gao, R., Jiang, Z. M., Barna, C., & Litoiu, M. (2016). A framework to evaluate the effectiveness of different load testing analysis techniques. In *2016 IEEE international conference on software testing, verification and validation, ICST 2016, chicago, il, usa, april 11-15, 2016* (pp. 22–32). IEEE Computer Society.

Gesvindr, D., & Buhnova, B. (2016). Performance challenges, current bad practices, and hints in paas cloud application design. *SIGMETRICS Perform. Evaluation Rev.*, *43*(4), 3–12.

Grano, G., Laaber, C., Panichella, A., & Panichella, S. (2019). Testing with fewer resources: An adaptive approach to performance-aware test case generation. *CoRR*, *abs/1907.08578*.

Haghdoost, A., He, W., Fredin, J., & Du, D. H. C. (2017). On the accuracy and scalability of

intensive I/O workload replay. In *15th USENIX conference on file and storage technologies, FAST 2017, santa clara, ca, usa, february 27 - march 2, 2017* (pp. 315–328). USENIX Association.

He, S., Manns, G., Saunders, J., Wang, W., Pollock, L. L., & Soffa, M. L. (2019). A statistics-based performance testing methodology for cloud applications. In *Proceedings of the ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, ESEC/SIGSOFT FSE 2019, tallinn, estonia, august 26-30, 2019* (pp. 188–199). ACM.

Ibidunmoye, O., Hernández-Rodriguez, F., & Elmroth, E. (2015). Performance anomaly detection and bottleneck identification. *ACM Comput. Surv.*, *48*(1), 4:1–4:35.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons.

Jiang, Z. M., Avritzer, A., Shihab, E., Hassan, A. E., & Flora, P. (2010). An industrial case study on speeding up user acceptance testing by mining execution logs. In *Fourth international conference on secure software integration and reliability improvement, SSIRI 2010, singapore, june 9-11, 2010* (pp. 131–140). IEEE Computer Society.

Jiang, Z. M., & Hassan, A. E. (2015). A survey on load testing of large-scale software systems. *IEEE Trans. Software Eng.*, *41*(11), 1091–1118.

Jiang, Z. M., Hassan, A. E., Hamann, G., & Flora, P. (2009). Automated performance analysis of load tests. In *25th IEEE international conference on software maintenance (ICSM 2009), september 20-26, 2009, edmonton, alberta, canada* (pp. 125–134). IEEE Computer Society.

Leitner, P., & Cito, J. (2016). Patterns in the chaos - A study of performance variation and predictability in public iaas clouds. *ACM Trans. Internet Techn.*, *16*(3), 15:1–15:23.

Liao, L., Chen, J., Li, H., Zeng, Y., Shang, W., Guo, J., … Sajedi, S. (2020). Using black-box performance models to detect performance regressions under varying workloads: an empirical study. *Empir. Softw. Eng.*, *25*(5), 4130–4160.

Malik, H., Jiang, Z. M., Adams, B., Hassan, A. E., Flora, P., & Hamann, G. (2010). Automatic comparison of load tests to support the performance analysis of large enterprise systems. In *14th european conference on software maintenance and reengineering, CSMR 2010, 15-18*

march 2010, madrid, spain (pp. 222–231). IEEE Computer Society.

Milligan, G. W., & Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, *50*(2), 159–179.

Nguyen, T. H. D., Adams, B., Jiang, Z. M., Hassan, A. E., Nasser, M. N., & Flora, P. (2012). Automated detection of performance regressions using statistical process control techniques. In *Third joint WOSP/SIPEW international conference on performance engineering, icpe'12, boston, ma, USA - april 22 - 25, 2012* (pp. 299–310). ACM.

pidstat(1): Report statistics for tasks - linux man page. (n.d.). https://linux.die.net/man/1/pidstat. ((Accessed on 02/26/2020))

Raz, T. (1992). The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling (raj jain). *SIAM Review*, *34*(3), 518–519.

Sandhya, N., & Govardhan, A. (2012). Analysis of similarity measures with wordnet based text document clustering. In *Proceedings of the international conference on information systems design and intelligent applications 2012 (india 2012) held in visakhapatnam, india, january 2012* (pp. 703–714).

Schulz, H., Angerstein, T., & van Hoorn, A. (2018). Towards automating representative load testing in continuous software engineering. In *Companion of the 2018 ACM/SPEC international conference on performance engineering, ICPE 2018, berlin, germany, april 09-13, 2018* (pp. 123–126). ACM.

Seo, B., Kang, S., Choi, J., Cha, J., Won, Y., & Yoon, S. (2014). IO workload characterization revisited: A data-mining approach. *IEEE Trans. Computers*, *63*(12), 3026–3038.

Shang, W., Hassan, A. E., Nasser, M. N., & Flora, P. (2015). Proceedings of the 6th ACM/SPEC international conference on performance engineering, austin, tx, usa, january 31 - february 4, 2015. In *ICPE* (pp. 15–26). ACM.

Shariff, S. M., Li, H., Bezemer, C., Hassan, A. E., Nguyen, T. H. D., & Flora, P. (2019). Improving the testing efficiency of selenium-based load tests. In *Proceedings of the 14th international workshop on automation of software test, ast@icse 2019, may 27, 2019, montreal, qc, canada* (pp. 14–20). IEEE / ACM.

Shivam, P., Marupadi, V., Chase, J. S., Subramaniam, T., & Babu, S. (2008). Cutting corners:

Workbench automation for server benchmarking. In *Proceedings of the 2008 USENIX annual technical conference, boston, ma, usa, june 22-27,2008* (pp. 241–254). USENIX Association.

Smith, C. (2006, 04). Software performance engineering. In (p. 509-536). doi: 10.1007/BFb0013866

Stapleton, J. H. (2008). *Models for probability and statistical inference: theory and applications* (Vol. 277). Wiley-Interscience.

Stevens, B. (2020 (accessed October 21, 2020)). Ocado is relaunching its app after being forced to scrap it in march due to 1000% jump in traffic [Computer software manual]. Retrieved from https://www.chargedretail.co.uk/2020/07/09/ocado-is-relaunching-its-app-after-being-forced-to-scrap-it-in-march-due-to-1000-jump-in-traffic/

Summers, J., Brecht, T., Eager, D. L., & Gutarin, A. (2016). Characterizing the workload of a netflix streaming video server. In *2016 IEEE international symposium on workload characterization, IISWC 2016, providence, ri, usa, september 25-27, 2016* (pp. 43–54). IEEE Computer Society.

Syer, M. D., Jiang, Z. M., Nagappan, M., Hassan, A. E., Nasser, M. N., & Flora, P. (2013). Leveraging performance counters and execution logs to diagnose memory-related performance issues. In *ICSM* (pp. 110–119). IEEE Computer Society.

Syer, M. D., Shang, W., Jiang, Z. M., & Hassan, A. E. (2017). Continuous validation of performance test workloads. *Autom. Softw. Eng.*, *24*(1), 189–231.

Tchana, A., Dillenseger, B., Palma, N. D., Etchevers, X., Vincent, J., Salmi, N., & Harbaoui, A. (2013). Self-scalable benchmarking as a service with automatic saturation detection. In *Middleware* (Vol. 8275, pp. 389–404). Springer.

Vögele, C., van Hoorn, A., Schulz, E., Hasselbring, W., & Krcmar, H. (2018). WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction - a model-driven approach for session-based application systems. *Software and Systems Modeling*, *17*(2), 443–477.

von Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., & Kounev, S. (2018). Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In *26th IEEE international symposium on modeling, analysis, and simulation of computer and telecommunication systems, MASCOTS 2018, milwaukee, wi, usa, september 25-28, 2018* (pp. 223–236). IEEE Computer Society.

Weyuker, E. J., & Vokolos, F. I. (2000). Experience with performance testing of software systems: Issues, an approach, and case study. *IEEE Trans. Software Eng.*, *26*(12), 1147–1156.

Xiao, X., Han, S., Zhang, D., & Xie, T. (2013). Context-sensitive delta inference for identifying workload-dependent performance bottlenecks. In M. Pezzè & M. Harman (Eds.), *International symposium on software testing and analysis, ISSTA '13, lugano, switzerland, july 15-20, 2013* (pp. 90–100). ACM.

Yadwadkar, N. J., Bhattacharyya, C., Gopinath, K., Niranjan, T., & Susarla, S. (2010). Discovery of application workloads from network file traces. In *8th USENIX conference on file and storage technologies, san jose, ca, usa, february 23-26, 2010* (pp. 183–196). USENIX.

Zhang, F., Keivanloo, I., & Zou, Y. (2017). Data transformation in cross-project defect prediction. *Empir. Softw. Eng.*, *22*(6), 3186–3218.

Zhang, P., Elbaum, S. G., & Dwyer, M. B. (2011). Automatic generation of load tests. In P. Alexander, C. S. Pasareanu, & J. G. Hosking (Eds.), *26th IEEE/ACM international conference on automated software engineering (ASE 2011), lawrence, ks, usa, november 6-10, 2011* (pp. 43–52). IEEE Computer Society.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In *Proceedings of the 41st international conference on software engineering: Software engineering in practice, ICSE (SEIP) 2019, montreal, qc, canada, may 25-31, 2019* (pp. 121–130). IEEE / ACM.

Related publication

Yuanjie, X., Lizhi, L., Jinfu, C., Heng, L. , Weiyi, S. (2021). Reducing the Length of Field-replay Based Load Testing. *Under Review in TSE.*