Reducing the Length of Field-Replay Based Load Testing

Yuanjie Xia, Lizhi Liao, *Graduate Student Member, IEEE*, Jinfu Chen, Heng Li, and Weiyi Shang

Abstract*—As software systems continuously grow in size and complexity, performance and load related issues have become more common than functional issues. Load testing is usually performed before software releases to ensure that the software system can still 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. However, replaying a lengthy, e.g., 48 hours, field workloads is rather resource- and time-consuming, and sometimes even infeasible for large-scale software systems that adopt a rapid release cycle. On the other hand, replaying a short duration of the field workloads may still result in unrealistic load testing. 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. Our approach sheds light on future research that aims to reduce the cost of load testing for large-scale software systems.

***Index Terms*—Load testing, workload reduction, workload replay, software performance.**

Manuscript received 22 January 2024; accepted 28 May 2024. Date of publication 31 May 2024; date of current version 15 August 2024. Recommended for acceptance by M. Pradel. *(Corresponding author: Lizhi Liao.)*

Yuanjie Xia, Lizhi Liao, and Weiyi Shang are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: y35xia@uwaterloo.ca; lizhi.liao@uwaterloo.ca; wshang@uwaterloo.ca).

Jinfu Chen is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: jinfuchen@whu.edu.cn).

Heng Li is with the Department of Computer and Software Engineering, Polytechnique Montreal, Montreal, H3T 1J4, Canada (e-mail: heng.li@polymtl.ca).

Digital Object Identifier 10.1109/TSE.2024.3408079

I. Introduction

Software performance is an essential measurement in software quality [62]. Prior studies show that the failures of large software systems are often due to performance and load-related issues rather than functional bugs [25], [72]. 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 [26], some online purchasing systems of supermarkets crashed or had extremely slow responses [13], [64]. Both the financial and reputational repercussions from these issues would be detrimental to the success of software systems.

Load testing is one of the major activities for ensuring the quality of services provided by the system under load [40]. 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 [67]. 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 [22].

Despite the advantages of load testing that is driven by field-relay, practitioners still face the dilemma between realistic load tests and their costs. On the 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 resources and time for 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 [6]. 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 paper, we present our approach that can reduce the length of field-replay based load testing while preserving 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 similar system behaviour as an indicator of having enough performance observations. We apply the Kolmogorov–Smirnov test [63] 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.

Paper organization. The remainder of the paper is organized as follows. Section II introduces the background of load testing. Section III presents our approach to reduce the length of the testing. Section IV introduces the subject systems we used and how we collect the data. Section V presents the results of our case study, organized along our three RQs. Section VII discusses prior research related to our work. Section VI discusses the sensitivity analysis on the different configurations of our approach. Section VIII discusses the threads to the validity of our results. Finally, Section IX concludes the paper.

II. Background

Load testing is the process of assessing a system's behaviour under a workload [40]. 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 [32], [45]. Prior studies propose techniques to design a proper workload [17], [69], [76], determine test length [6], [35], analyze test results [41], [48], [59], [67], and detect performance issues [36], [73]. 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 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 [39] or when to stop load testing [6]. Prior approaches are typically based on the repetitiveness of software logs [14], or the naive comparison of raw performance counters [6], [55]. 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

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

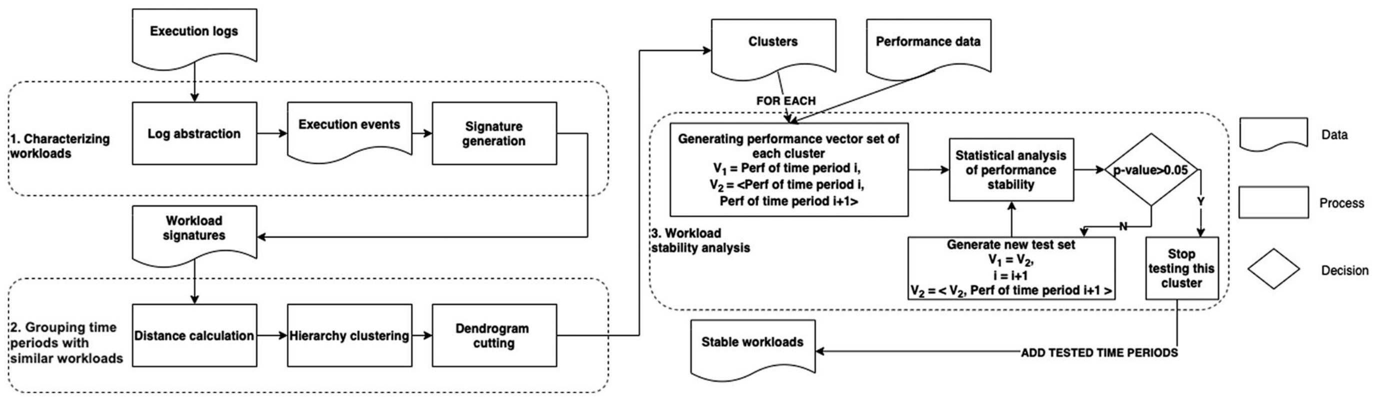

Execution logs
1. Characterizing workloads
Log abstraction
Execution events
Signature generation
Workload signatures
2. Grouping time periods with similar workloads
Distance calculation
Hierarchy clustering
Dendrogram cutting
Clusters
Performance data
FOR EACH
Generating performance vector set of each cluster
V1 = Perf of time period i,
V2 = <Perf of time period i, Perf of time period i+1>
Statistical analysis of performance stability
p-value>0.05
N
Y
Generate new test set
V1 = V2,
i = i+1
V2 = < V2, Perf of time period i+1 >
Stop testing this cluster
3. Workload stability analysis
Stable workloads
ADD TESTED TIME PERIODS
Data
Process
Decision

Fig. 1. The overview of our approach.

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 section details our approach.

III. Approach

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

A. 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.

1) Log Abstraction: Software execution logs are produced during software system execution, which usually records 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 [27]. For example, prior work has found that events in logs are useful sources for workload recovery [65], [67]. Therefore, in this step, we parse the system execution logs to extract timestamps, user events and system responses.

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 I is an illustrative example of execution logs and their corresponding log events. In our experiments, we use regular expressions to

TABLE I
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

extract log events. However, in practice, one may adopt various automated log abstraction techniques [77] for this step.

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 [17]. 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 an n-dimensional vector (i.e., each element value in the vector represents the number of appearances of a unique log event during that 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 contain a mixture of different types of workload behaviors (i.e., behaviors that do not belong to the same cluster), which can degrade the performance of our approach that leverages clustering to reduce workloads. Intuitively, the workload behaviors within a time period or a cluster should present a similar pattern to allow effective clustering. In this paper, we opt to use 10 minutes as the length of our time periods in order to capture more diverse workloads. Comparing to prior research [66] 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.

TABLE II
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

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 data point in the time series is an n-dimensional space. Table II 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.

B. 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.

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 [29] to calculate the cluster distance since the Pearson distance often produces a clustering that is a close match to the manually assigned clusters [56]. The equations (1) and (2) present the calculation of the Pearson distance [28].

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

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

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.

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.

Hierarchical clustering starts by defining each sample as one cluster, so at the beginning we have a cluster set $X = \{X_1, X_2, ..., X_n\}$, which n represents the number of samples. By using the distance calculation that we defined in Section III-B1 (Equation (1) and (2)), we can obtain the distance between two clusters, which is the linkage distance $\Delta(X_i, X_j)$.

TABLE III
PERFORMANCE VECTORS FOR THE TIME PERIODS IN OUR RUNNING EXAMPLE (BASED ON TABLE II)

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.

Thus, we could start to build a binary merge tree [52] by merging the pair of clusters that are closest to each other. The merge does not stop until the binary merge tree covers all samples and merges to one single cluster at the top. Finally, the resulting binary merge tree represents the hierarchical relationships between the clusters.

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 [16] 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 [50]. Prior research also reported that the Calinski-Harabasz stopping rule outperforms other stop rules when clustering workload signatures [67].

Applying the clustering method to our running example, we can obtain a clustering result for the workload signatures in Table II. 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.

C. 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.

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 timestamp 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 III shows the vector of performance data for each time period in our running example. After this step, we can obtain a 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

TABLE IV
THE WORKLOAD STABILITY ANALYSIS FOR THE WORKLOAD CLUSTERS IN OUR RUNNING EXAMPLE (BASED ON TABLE III)

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

to each cluster. The time periods belonging to cluster x can be defined as $C_x = \{t_{x1}, t_{x2}, ..., t_{xn}\}$. The corresponding performance vector is $S_{Cx} = < P_{t_{x1}}, P_{t_{x2}}, ..., P_{t_{xn}} >$.

2) Statistical Analysis of Performance Stability: To check the stability of each cluster's performance S_{Cx}, we start from the first two time periods 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 [63] 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 III to perform the workload stability analysis, and the result is shown in Table IV. 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. In our practice, we chose to monitor performance every ten seconds. Since we mentioned that the length of the time period is 600 seconds above (cf. Section III-A), the performance vector in each time period has 60 elements.

As we describe above, we compare the performance vectors in the time period set C_x, in which all the time periods share the similar workload behaviors and are grouped into cluster x. Without injecting performance bugs or adding new features, we can ensure the stability of performance of the workload behaviors in each cluster. In another word, after we confirm the stability of a workload cluster, the performance of the workloads belonging to this cluster are predictable, and the measured performance at different time periods can converge after a limited number of measurements.

TABLE V
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

IV. Case Study Setup

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

A. 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* [70] 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 [17], [18], [30]. The overview of the three subject systems is shown in Table V.

In our experiments, for a more precise evaluation, we should clarify that we make sure that the database of our subject systems runs in normal conditions without saturation, and we assume that the database of the subjects only has a negligible effect on the subject system's performance.

B. 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. 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 [1] 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* [4] to monitor the process of the system every ten seconds. Because we mainly focus on the performance of the host servers in the experiment, we reduce the influence of the database (i.e., avoid rapid growth of the database size) by the design of our workload. We detail our workload design for each of our subject systems below. The details of our data can be found in our replication package[2].

OpenMRS: We setup our *OpenMRS* system with the *OpenMRS* demo database version 2.2.1 [3] in our load tests. The demo database contains various data for 5,000 patients. *OpenMRS* contains four typical requests: addition, deletion, search, and editing. 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. Because we have both POST and DELETE test events at the ratio near 1:1 dynamically, the influence of the database size is not significant.

We deployed *OpenMRS* on two virtual machines, each with 4-core vCPU, 16GB RAM, and 24GB persistent disk. One machine is deployed as the application server, and the other machine is 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. The execution log in *OpenMRS* is well structured, so we do not need to use log parser tools in our paper.

Apache James: We used JMeter to create load tests to exercise the *Apache James* server. We replicate the similar workloads as a prior study [30]. 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 also periodically delete the emails to reduce the influence on the growth of the database size.

We deployed *Apache James* in a server machine with 8-core vCPU, 32 GB memory on a 2 TB persistent disk. We run JMeter on another machine with 4-core vCPU, 8GB memory and 24GB persistent disk. Finally, we execute one-day-long workloads to load test the *Apache James* server using JMeter. We obtain the web request logs automatically generated by 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. Although we have POST test events like adding products to the cart, it is not a long workload and we have limited the number of users and categories of products to avoid the rapid growth of the database size.

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 JMeter load driver with varying workloads to simulate users accessing the system with the above-mentioned use cases. Then, we obtained the execution logs from the Tomcat server execution logs, which are well-structured logs.

[2]https://zenodo.org/record/5593380#.YXMCq9mZNTY

TABLE VI
PETITT'S TEST RESULTS IN OUR SUBJECTS

Subjects	p-Value	Num of Change Points
OpenMRS	$\ll 0.001$	5039
TeaStore	$\ll 0.001$	4987
Apache James	$\ll 0.001$	5029

Workload diversity. To have more realistic workloads and avoid generating time-homogeneous workloads, we apply a 2-day-long real-world workload trend from WorldCup98 access logs [9], which is widely used for workload designs in prior work [11], [19]. We simulate the workload time series in our workloads based on real-world workload features. We calculate the frequency of each type of request and the number of users every half hour. Then, to closely mirror actual user behaviour, we sort the frequency of the request type and link the WorldCup98 behaviour with the request type in our subjects. For example, we link one GET image request in WorldCup98 to the GET person data request in *OpenMRS*. Finally, we set up the number of threads and frequency of the workloads based on the calculated features. The vastly different workloads between different time periods can provide the maximum support for simulating most customer activities. To have a more comprehensive view of the diversity in the workloads, we apply Pettitt's test [54] to verify if the workload is time-homogeneous. Pettitt's test is a statistical test used to detect any abrupt change point in the time series in Prior works [31], [47]. Table VI presents the statistical results on the workloads in subject systems. We use the classical threshold of 0.05 for statistical tests to determine the statistical significance. When p-value smaller than 0.05, we can confirm that the change points exist in the time series of the workloads. Therefore, there are moments in time when the workload characteristics have significant changes, which proves that the workloads are not time-homogeneous.

V. CASE STUDY RESULTS

In this section, 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 Section II, determining the length of the field workloads is challenging. A set of workloads with a 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 [40]. 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. Section III) to the three datasets obtained from our experiments on the studied subject systems.

TABLE VII
WORKLOAD REDUCTION RESULTS FOR OUR STUDIED SYSTEMS

Project	**Time Length (Minutes)**		**# Clusters**	
	Before Reduction	**After Reduction**	**# Total Clusters**	**# Stable Clusters**
OpenMRS	2,880	470	23	21
TeaStore	2,880	370	21	11
Apache James	2,880	450	22	20

In particular, the datasets contain the logs and the performance metrics (CPU usage) 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 reduces 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 three numbers: the total number of clusters of workloads, 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 VII shows the results of our approach for reducing the performance testing workloads on our studied subject systems. By applying our approach, we find that 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 being reduced by our approach is only 370 minutes, whereas the original set of load testing workloads requires two days (2880 minutes) of execution (i.e., the reduction rate is 87%). 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 our approach can reduce the majority of the clusters of workloads. For example, for *Apache James*, 91% of the clusters of workloads can be further reduced by our approach. For those clusters that cannot be reduced in our approach, we consider the reason being the size of those clusters, i.e., there are only one or two time periods in those clusters, which are difficult to reduce further. For example, for TeaStore, all the clusters that cannot be reduced only have one or two time periods.

When the system is under random and varying workloads, simply reducing the length of load test workloads by time can miss representative workloads. Fig. 2 presents the convergent speed of each cluster when applying our approach for reducing performance testing workloads on our studied subject systems. Each line in the figure shows p-values of each cluster of workloads during workload stability analysis (cf. Section III-C), 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. Moreover, the distribution of lines with low slopes does not concentrate on a part of the time periods. For example, for *OpenMRS*, lines with low slopes exist around the 500th and 1600th minute in the figures. 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 the original set of load testing workloads by time, for example, only keeping a few hours at the beginning, some important field workloads would be missing, and it would be hard to achieve stable performance.

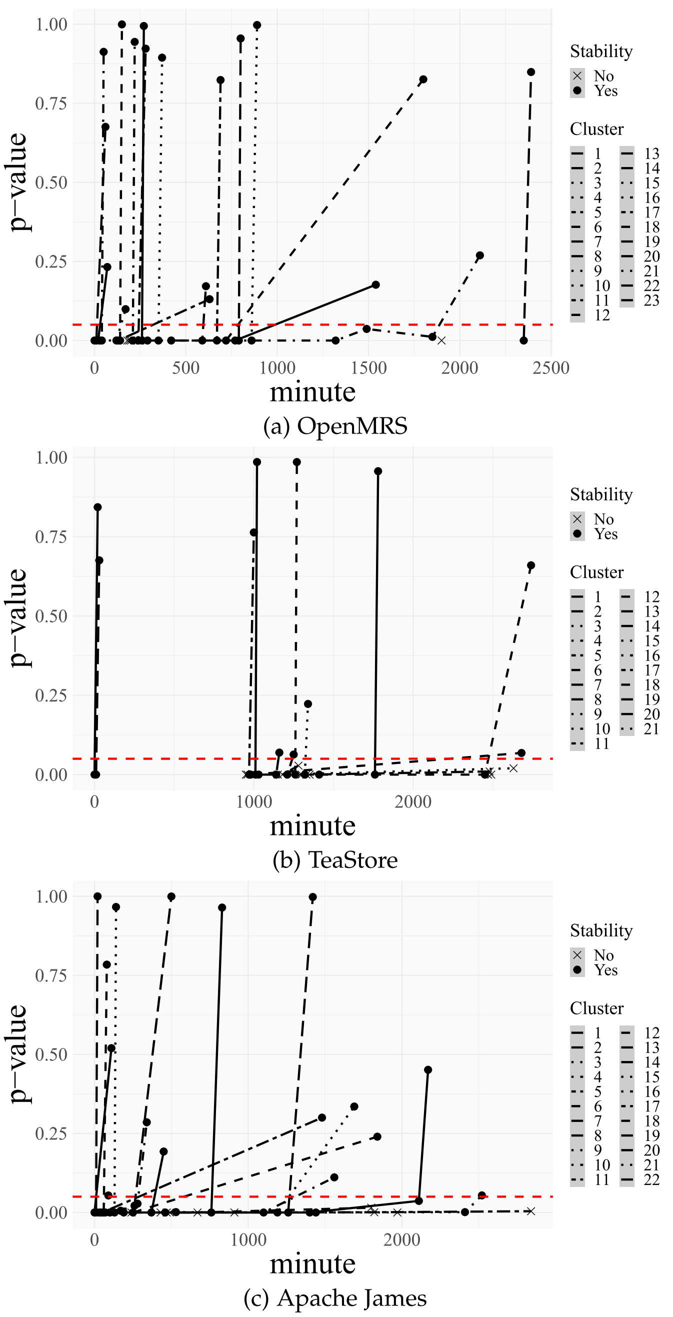


Fig. 2. The convergence speed of the workloads clusters. The red horizontal lines indicate the threshold of p-value (0.05) for determining statistical significance.

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.

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 section.

We apply the random forest learning method to build the model [46]. Random forest is a learning method widely used for classification and regression. The random forest model uses the workload signature every ten seconds as independent variables and performance metrics (i.e., the CPU usage data every ten seconds) as its dependent variables. After training, we obtain a random decision forest consisting of many individual decision trees that operate as an ensemble. Given the workload signatures in a ten-second period as the input, the composing decision trees make their individual predictions of the performance metric and then cast a vote on the final predicted result.

Using a random forest model that takes the workload information in a time period as input and predicts the associated performance, we assume that our systems (more specifically, how the systems react to the requests) are stateless. From one point of view, we observed that the processing time of the requests of the systems is very short, with the maximum processing time below 400 ms and average processing time below 100 ms. From another point of view, we conducted experiments to compare the performance of the system in two different phases of execution under the same workload. We observed that there is no significant difference in processing between the two phases of execution (results are added in our replication package). Therefore, we could not reject our assumption that the performance behaviour of our studied systems is not negligibly impacted by their states.

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 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 in the original set of workloads. We perform statistical analysis to examine the deviation between the predicted and observed performance in terms of CPU usage. Specifically, we calculate the Pearson correlation [29] 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 can 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. To comprehend the difference between the two models (i.e., the model of baseline and M_r), we use the Kolmogorov–Smirnov test [63] 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 [21] to quantify the effect size between the predictions of the 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 used in prior research [46]. 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 to 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 VIII presents the median relative error of the model built on workloads-after-reduction

TABLE VIII
COMPARING THE ORIGINAL PERFORMANCE DATA AND THE PERFORMANCE PREDICTED BY THE MODELS BUILT FROM THE WORKLOADS-AFTER-REDUCTION

Project	**MARE**	**Correlation**
OpenMRS	11.45%	0.88
TeaStore	10.45%	0.92
Apache James	10.24%	0.63

TABLE IX
COMPARING THE PERFORMANCE PREDICTED BY THE MODELS THAT ARE BUILT FROM THE WORKLOADS-AFTER-REDUCTION AND THE ORIGINAL SET OF WORKLOADS (I.E., THE BASELINE)

Project	***p-Value***	**Cohen's D**
OpenMRS	$\ll 0.001$	0.38(small)
TeaStore	$\ll 0.001$	0.12(negligible)
Apache James	0.42	0.02(negligible)

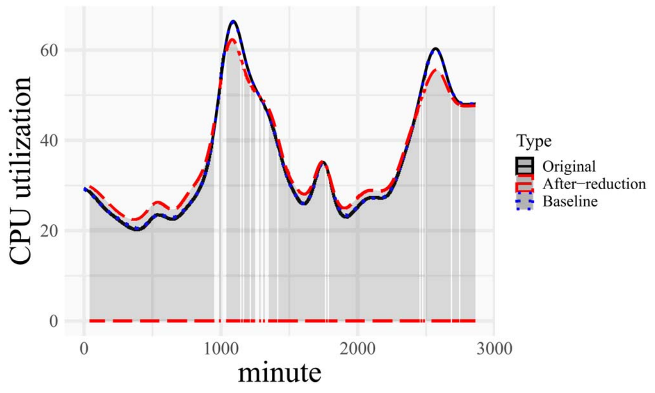


Fig. 3. Prediction performance of TeaStore. The shaded region represents the predicted performance data while the un-shaded area shows the performance data that are used to build the models (the workloads that are kept after the reduction).

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 11.45%, 10.45%, and 10.24%, respectively, for system performance prediction. Moreover, the relatively high Pearson correlations (0.88, 0.92, and 0.63) 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.

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 IX. 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, Fig. 3 presents the trends of both the original and predicted performance data over time, in which we can clearly see how representative the workloads-after-reduction are. In particular, the shaded region represents the predicted performance data. At the same time, the unshaded area shows the performance data used to build models, which is also during the period of the workloads-after-reduction. The graph shows 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, we only provide the example of one system, i.e., *TeaStore*. The run charts of the other two subject systems are included in our replication package.

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 represent the original varying workloads captured in the same field environment. However, in practice, end users' original workloads are typically extracted from the field environments. In contrast, 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 replace the original performance testing workloads in practice.

To simulate the change of runtime environment configurations, it is necessary to test both the change of hardware configurations and software configurations. As the load of servers increases, developers tend to upgrade the hardware to a higher configuration to improve the performance of the services. In contrast, developers may downgrade the hardware to a lower configuration if the default hardware configuration is too high to utilize. Since the various JDKs made by different publishers have been used in reality, the experiments on different JDKs are important. In addition, updating the database version is also a critical change in software. For example, since the security and performance of MySQL 8.0 is better than 5.7, developers would migrate MySQL to version 8.0 if necessary. As a result, testing all these configurations is necessary to evaluate the effectiveness of our approach.

Therefore, this RQ aims to examine whether the replaying results using the workloads-after-reduction generated by our approach in a different environment are still representative of the original workloads from the original field environment.

Approach

To answer this research question, we redeploy our studied subject systems in a new environment to be the replay environment. To minimize potential noise factors, we keep the server configurations consistent (e.g., operating system version, maven version), except for the variables that must be changed (e.g., vCPU numbers and Java source). Additionally, we employ

TABLE X
THE HARDWARE CONFIGURATIONS OF THE ORIGINAL AND THE REPLAY ENVIRONMENTS

Project	**Hardware Configuration (Using Openjdk and MySQL 5.7)**	
	Original	**Replay**
OpenMRS	4vCPU, 16GB memory	8vCPU, 32GB memory 2vCPU, 8GB memory 0.5vCPU, 4GB memory
TeaStore	4vCPU, 16GB memory	8vCPU, 32GB memory 2vCPU, 4GB memory 1vCPU, 4GB memory
Apache James	8vCPU, 32GB memory	16vCPU, 64GB memory 4vCPU, 16GB memory 2vCPU, 4GB memory

TABLE XI
THE SOFTWARE CONFIGURATIONS OF THE ORIGINAL AND THE REPLAY ENVIRONMENTS

Project	**Software Configuration (Using Original hardware configuration)**	
	Original	**Replay**
OpenMRS	OpenJDK, MySQL 5.7	Oracle-JDK, MySQL 5.7 Zulu-JDK, MySQL 5.7 OpenJDK, MySQL 8.0
TeaStore	OpenJDK, MySQL 5.7	Oracle-JDK, MySQL 5.7 Zulu-JDK, MySQL 5.7 OpenJDK, MySQL 8.0
Apache James	OpenJDK, version 2.3	Oracle-JDK, version 2.3 Zulu-JDK, version 2.3 OpenJDK, version 2.2

workloads that are identical to the workload-after-reduction to test the system. We test the higher or lower hardware configuration with OpenJDK and MySQL 5.7. The details of the different hardware configurations between the replay environment and the original environment are shown in Table X. For each system, we consider upgrading and downgrading the hardware configurations. For example, for the *OpenMRS* system, while the original environment has 4vCPU and 16G memory, we consider a replay environment with 8vCPU and 32G memory as well as another replay environment with 2vCPU and 8G memory. To understand the performance of the subject systems when they reach their saturation, we also deploy the systems in extremely basic replay environments, such as using 0.5vCPU for OpenMRS and 1vCPU for TeaStore. Since the minimum configuration of CPU is 2vCPU, we use the cpulimit [2] tool to simulate the hardware environment under 2vCPU by limiting the maximum usage of the CPUs. Moreover, we choose Oracle-JDK and Zulu-JDK to test the different JDK publisher conditions and MySQL 5.7 and 8.0 to simulate the database update across different versions. In particular, because *Apache James 2.X* directly stores data on disk (i.e., without a database), we cannot change the database version of *Apache James*. As a solution, we choose a different version (*Apache James 2.2*) for a comparison. The details of the different software configurations between the replay environment and the original environment are shown in Table XI.

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* [4]. 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 the 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 differences exist 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 values 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 [8] 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 the 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 hypothesize 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 the 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 $\alpha_{Normalize} = \frac{k_\alpha \cdot \alpha + h_\alpha - h_\beta}{k_\beta}$. By 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. [51].
- **Robust** scaling method is an advanced version of the median scaling method. The median absolute deviation is replaced by the 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 [75].

- **Transfer learning** is used to improve a learner for one domain by transferring information from a related domain [71]. In this field, Krishna et al. [43] propose a framework called BEETLE for finding the best source of transfer learning. Several previous studies [49], [53] extend the configuration space for transferring learning. Also, Jamshidi et al. [38] focus on the performance of the transfer learning method under different configuration change scenarios, which are highly related to the scaling problems we will discuss below. For the generalizability of our approach and ease of the experiment, we adopt a convolutional neural network (i.e., CNN) widely used in prior work [5] as the architecture for training our deep learning model. Specifically, we use the log sequences in the workload signatures and the corresponding performance data of the workload-after-reduction as the features and the response variable, respectively, to train the CNN model. To apply transfer learning, we fine-tune the trained model using the replay data to obtain another model. This model can be later used in this RQ to understand the representativeness of the reduced workloads for workload replay.

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 the 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 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 XII 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. Fig. 4 presents the trend of scaled original performance data and predicted data over time for *OpenMRS*. If we exclude the saturated condition of the system, we can observe significant correlations, ranging from 0.34 to 0.92, 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 are still representative of the original workloads from the original field environment. However, we notice that when the system reaches its performance bottleneck, the degradation of the replay's representativeness would be significant. Table XII shows that when using the weakest configurations, the correlation between the original and the replayed system performance is much lower than when using other configurations. As the subjects reach the bottleneck, the systems cannot respond to requests in a common efficiency, which causes the latency of the workloads to be higher than usual. Such results show that with the saturation and slow response of the system, the behaviour of the system under the workload will be different from its behaviour under a normal workload. Our results suggest that our proposed approach for testing length reduction and replay is more appropriate for the scenarios when the system is tested under normal workloads. Further, the results inspire us to be careful of the side effects of changing configuration when having a replay.

Future work on scaling the performance data from different environments is needed. Table XII 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

TABLE XII
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	Configuration	Correlation	MARE After Scaling					
			Max-Min	Median	Model	Robust	Quantile	TL*
OpenMRS	8vCPU	0.91	92.99%	123.99%	33.23%	41.22%	**19.94%**	47.53%
	2vCPU	0.87	467.57%	1315.11%	247.26%	49.24%	**22.26%**	764.36%
	0.5vCPU	0.78	603.73%	128.65%	681.37%	63.57%	**29.19%**	481.71%
	Oracle-JDK	0.91	43.08%	139.74%	16.19%	42.98%	20.49%	**15.68%**
	Zulu-JDK	0.90	75.04%	274.11%	57.94%	42.94%	**20.29%**	434.58%
	MySQL 8.0	0.91	57.21%	148.83%	**12.50%**	43.86%	20.64%	36.70%
Teastore	8vCPU	0.91	45.12%	130.31%	69.58%	48.52%	**22.78%**	87.17%
	2vCPU	0.88	58.01%	139.97%	**14.42%**	100.16%	49.38%	36.40
	1vCPU	0.61	56.22%	159.60%	**23.54%**	107.93%	49.17%	37.13%
	Oracle-JDK	0.92	70.82%	201.17%	**28.53%**	124.86%	55.04%	65.61%
	Zulu-JDK	0.71	54.89%	147.64%	**26.27%**	84.50%	40.91%	64.58%
	MySQL 8.0	0.76	70.55%	164.92%	**30.26%**	64.06%	30.83%	69.62%
Apache James	16vCPU	0.45	63.26%	88.23%	74.80%	106.32%	**44.07%**	70.79%
	4vcpu	0.52	152.95%	81.79%	**14.90%**	100.75%	41.87%	24.68%
	2vcpu	0.29	190.25%	92.78%	54.08%	116.36%	**50.22%**	168.85%
	Oracle-JDK	0.76	66.58%	90.62%	33.67%	71.94%	29.59%	**13.86%**
	Zulu-JDK	0.89	40.58%	98.99%	36.43%	108.48%	46.80%	**22.06%**
	Version2.2	0.97	98.23%	110.94%	**20.86%**	137.63%	60.42%	23.13%

* TL: Transfer Learning

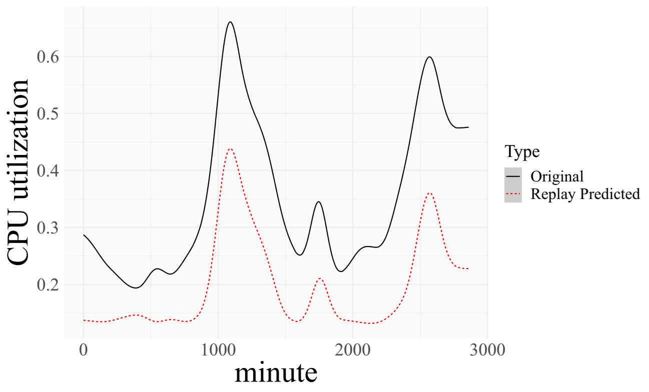


Fig. 4. Comparison of the original performance and the predicted performance based on the replay data (after the Max-Min scaling) for TeaStore.

different environment, using all the selected scaling approaches, the performance model still has a relatively high MARE, while under the same environment, the maximum MARE is only 11.45% (cf. Table VIII).

Such results indicate the limitations of the current scaling approaches for analyzing performance data from different environments. From the empirical results from Jamshidi et al. [38], some non-linear correlation may exist during some configuration changes. However, it is also encouraging to see that the prediction errors become similar to others when scaling methods are applied. Such results signify the importance of advanced scaling methods for future performance engineering tasks that are conducted across different environments. Although Iqbal et al. [42] have developed a debugging tool for fixing misconfiguration on CPU latency, this tool may not address all configuration changes across environments. Among all the scaling methods we used, we also cannot find one that has stable performance. Model scaling, Quantile scaling, and transfer learning methods have a relatively better performance on scaling in different software environments. For example, for Oracle-JDK configuration in *Apache James*, the transfer learning method can have a MARE as low as 13.86%.

As performance data generated in the field contains 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 [46]. However, in these cases, the testing environment is often not completely identical to the production one. 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 use properly and reasonably. Therefore, our findings also advocate the need for future research on better scaling the performance data from different environments to reduce the bias caused by configuration differences.

The challenge of *Apache James* The less-promising results for the *Apache James* system might be due to its nature of being a mail server application. In particular, we designed multiple JMeter scripts to simulate the user operations on the system, e.g., reading and receiving different sizes of mail. When processing an email with a relatively large size, such workloads are likely to influence system I/O more than CPU. As a result, after we increase the cores of the CPU for testing, the usage of the CPU tends to have no change because of the low load on the CPU. Since the CPU variance of *Apache James* under different workloads is low, the results in a trained model optimized in the low variance CPU range perform poorly in unseen CPU usages (e.g., when the number of CPU cores is doubled). This explains the relatively poor performance of the model trained for *Apache James* in a 16vCPU environment in RQ3.

VI. Sensitivity Analysis

In this section, we perform a sensitivity analysis on three parameters in our approach, including the distance measurement, the time period size and the frequency of collecting performance data.

TABLE XIII
The Clustering Results Using Different Distance Metrics

Distance Metrics	Project	Cluster Number	MARE
Euclidean	OpenMRS	2	51.52%
	TeaStore	10	11.02%
	Apache James	2	18.39%
Cosine	OpenMRS	3	42.03%
	TeaStore	2	18.28%
	Apache James	2	16.17%
Pearson	OpenMRS	23	11.45%
	TeaStore	21	10.45%
	Apache James	22	10.24%

A. *Impact of the Distance Measurement in Workload Clustering*

In our study, we use Pearson distance to measure the difference between different workload signatures. To understand the impact of choosing different distance measurements, we also cluster our workload signatures using two other distance metrics that are commonly used in data clustering, including Cosine distance and Euclidean distance.

We find that using Cosine distance on all subjects can only lead to two clusters of workloads. The low number of clusters is caused by the bias from the Euclidean distance, where one dimension (i.e., representing a log event) with large variances in the data may dominate other dimensions. Similarly, using Cosine distance also leads to a very small number of clusters. In particular, the data from *Apache James* can only be clustered into two groups using Euclidean distance. However, we can obtain ten clusters for TeaStore using Euclidean distance since we have a wider range of values (i.e., ten times larger on range due to downstream request). This extensive range in the data causes the Euclidean distances to be more dispersed compared to other subjects, leading to a greater number of identifiable clusters. This is because the relative difference in the performance data may be small, and the Cosine distance cannot distinguish the differences between workloads. The low number of clusters would lead to difficulty in obtaining stable performance data in the same cluster of workloads. In fact, we observe that the prediction results from the Table XIII that using Euclidean distance and Cosine distance obtain a worse MARE while the number of clusters decreases since the cluster results cannot contain all the features of the workloads.

B. *Impact of the Time Period Size*

In order to understand the impact of choosing different sizes of time periods, we chose a relatively larger time period size, i.e., 1200 seconds and a relatively smaller one, i.e., 90 seconds, as comparisons with our original 600-second time period size.

With 1200 seconds as the size of the time periods, we obtain similar stable clusters as our original results with a 600-second time period size. For example, we produce 10 stable clusters for *OpenMRS* using 1200-second time period sizes(compared to the original 23 clusters). Based on the decreased cluster number, the length of the workload-after-reduction decreases from 470 to 300 minutes. However, using the workload-after-reduction, the MARE increases from 11.45% to 16.58%.

When using a 90-second time period, we end up with numerous clusters. Since the target of the stopping rule applied is to search the highest value of the ratio between the dispersion between clusters (i.e., using assumed cluster numbers for calculating the ratio) and the dispersion of the elements in the clusters, the exaggerated differences between clusters would dominate the correlation inside the clusters. The consequence of the domination would tend to split lots of single sample clusters. For example, the workload is split into 37 clusters for *OpenMRS*. There are two reasons that we avoid generating too many clusters. Firstly, the large number of clusters may not be suitable for developers to analyze. Secondly, the shorter time period would contain a small amount of performance data in each time period, which may not be suitable for statistical testing, such as the Kolmogorov–Smirnov statistical tests used in our approach. Without enough sample size, the result of the Kolmogorov–Smirnov statistical test is not reliable [44].

C. Impact of the Frequency of Collecting Performance Data

In our approach, we collect performance data every ten seconds. However, we would like to study the impact of changing the frequency of collecting performance data to every 30 seconds.

With a 30-second sampling frequency, we can obtain a shorter length of workload-after-reduction (i.e., 330 minutes) for *OpenMRS*. Similar to our above discussion on the time period sizes, the Kolmogorov–Smirnov statistical test is biased with falsely reported significant results [44] due to the lower number of samples from the 30 seconds sampling frequency. The MARE of the prediction based on a shorter length of workload-after-reduction increases to 13.21%, which confirms that with a lower frequency at 30 seconds, our approach may not perform as well as using a higher frequency of performance data.

Therefore, based on our findings from both the time period size and the frequency of collecting performance data, practitioners who would like to adopt our approach should ensure that the sample of performance data from each time period is large enough to avoid bias from statistical analyses.

D. Impact of High Utilization of the Systems

In our RQ1 and RQ2, we control the workload of the systems under normal operation of the system (i.e., without saturation) to prevent errors or performance regressions from occurring under high system stress. Thus, the performance of our method in a saturation condition is unknown. To verify the validity of our performance prediction model in this condition, we deployed a 1vCPU, 4GB memory instance for simulation. We ran a 24-hour-long workload for testing in *TeaStore* and grouped the workload into 10 stable clusters. Using our workload reduction approach, the workload is grouped into 8 stable clusters and reduced to a 190-minute version. Using the workload-after-reduction, we could get an 11.1% MARE and a correlation of 0.94 between the prediction results and the original performance. The promising result of the test shows the compatibility of our approach to working under a high-pressure system environment. However, the validation of the prediction model working under a different environment may have a large influence according to the result from Table XII.

VII. Related Work

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

A. Load Test Reduction

Several prior studies [6], [35], [57] share a similar goal to our work. He et al. [35] 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. [6] 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 [37] proposed to find the stop point of performance testing through applying a 5% threshold of the variance in response time. Daly et al. [24] apply the Q statistic to detect changing point in testing for acknowledge the performance of the system. Busany et al. [14] and Jiang et al. [39] 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 [10], [61], [68]. Apte et al. [7] 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 paper, 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. [60] demonstrated how they optimize system resources by running a browser-based load test with Selenium[3]. 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. [33] 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.

B. User Workload Characterization

The prior research from Cohen et al. [20] 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

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

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. [66] 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. [59] 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 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. [34] 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. [74] 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. [15] proposed an automatic workload characterization approach for I/O-intensive software in a virtual environment. Bumjoon et al. [58] defined twenty I/O related metrics to generate I/O workload signatures and clustered the I/O workloads. Eli et al. [23] 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 is easier to be integrated into Dev-Ops [12].

VIII. Threats to Validity

In this section, 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 [46], [17]. However, our subject systems may not represent all the software domains, and our approach and results may not be directly 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 system's execution logs. 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. Furthermore, since we apply a random forest method to learn our prediction model, the requests are assumed to be stateless. We have added an experiment to analyze if the performance distribution of each type has a difference. Through the analysis, we can confirm that the state of request types we used has negligible effect in our experiments. Although we can confirm that the state of request types we used have negligible effect in our experiments based on the conclusion shown in Section V RQ2, it is still a threat for future work that applies our approach in scenarios where requests are highly influenced by their states. In those scenarios, the prediction model using random forest may not be suitable.

Moreover, although we consider different workloads in our study, the impact of the intensity influences our prediction result. We calculate the Pearson correlation between MARE and intensity in each time period for our subject systems. We find that *OpenMRS*, *TeaStore* and *Apache James* have a low correlation, with values of only 0.12, -0.13, and -0.09, respectively. In order to reduce the impact of the intensity of the workloads, we will consider the intensity of the workloads in our future work.

In addition, we do not consider the sequence of the actions during the workload signature generation, which may cause different workloads to be 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 the main performance metric that reflects the system's 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.

In this work, we assume that our approach is applied to CPU-intensive systems and uses CPU usage as our focused performance metric. Under such an assumption, we reduce the interference of other performance-related resources, such as the effect of databases on our experiments. For example, if the system reaches the performance bottleneck of the database, the saturation of the database may cause a long response time and different response codes, which are highly different from the performance behaviours in a normal situation. Moreover, a disk or network problem may also affect the precision of the prediction. The unstable conditions would cause the same workloads to act in different performance behaviours, making it difficult for our approach to obtain stable performance from different workload periods belonging to the same clusters (i.e., impairing the effectiveness of workload reduction). Therefore, applying our approach in a different scenario that violates our assumption may lead to different results and conclusions. Nevertheless, the main methods (e.g., clustering-based workload reduction) of our approach can be conveniently applied to other performance metrics (e.g., network usage) and other types of systems (e.g., memory-intensive systems). We leave such evaluation as the future work.

IX. Conclusion

In this paper, 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 paper 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

[1] "Compute engine: Virtual machines (VMS)." Google Cloud. Accessed: Sep. 3, 2020. [Online]. Available: https://cloud.google.com/compute

[2] "CPUlimit—Limits the CPU usage of a process." Ubuntu Manpage. Accessed: Aug. 1, 2022. [Online]. Available: https://manpages.ubuntu.com/manpages/trusty/man1/cpulimit.1.html

[3] "Demo data—Resources." OpenMRS Wiki. Accessed: Oct. 10, 2020. [Online]. Available: https://wiki.openmrs.org/display/RES/Demo+Data

[4] "pidstat(1): Report statistics for tasks—Linux man page." Die.net. Accessed: Feb. 26, 2020. [Online]. Available: https://linux.die.net/man/1/pidstat

[5] "Text classification from scratch." Keras. Accessed: Oct. 15, 2021. [Online]. Available: https://keras.io/examples/nlp/text_classification_from_scratch/

[6] H. M. Alghmadi, M. D. Syer, W. Shang, and A. E. Hassan, "An automated approach for recommending when to stop performance tests," in *Proc. IEEE Int. Conf. Softw. Maintenance Evolution (ICSME)*, Raleigh, NC, USA. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2016, pp. 279–289.

[7] V. Apte, T. V. S. Viswanath, D. Gawali, A. Kommireddy, and A. Gupta, "AutoPerf: Automated load testing and resource usage profiling of multi-tier internet applications," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, L'Aquila, Italy. New York, NY, USA: ACM, 2017, pp. 115–126.

[8] M. M. Arif, W. Shang, and E. Shihab, "Empirical study on the discrepancy between performance testing results from virtual and physical environments," in *Proc. 40th Int. Conf. Softw. Eng. (ICSE)*, Gothenburg, Sweden. New York, NY, USA: ACM, 2018, p. 822.

[9] M. Arlitt and T. Jin, "1998 world cup web site access logs." acm sigcomm. Accessed: Jul. 1, 2023. [Online]. Available: http://www.acm.org/sigcomm/ITA/

[10] V. Ayala-Rivera, M. Kaczmarski, J. Murphy, A. Darisa, and A. O. Portillo-Dominguez, "One size does not fit all: In-test workload adaptation for performance testing of enterprise applications," in *Proc. ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, Berlin, Germany. New York, NY, USA: ACM, 2018, pp. 211–222.

[11] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 800–813, Apr. 2019.

[12] C. Bezemer et al., "How is performance addressed in devops?" in *Proc. ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, Mumbai, India. New York, NY, USA: ACM, 2019, pp. 45–50.

[13] E. Bureau, "BigBasket app, site crash on high demand." *The Economic Times*. Accessed Oct. 21, 2020. [Online]. Available: 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

[14] N. Busany and S. Maoz, "Behavioral log analysis with statistical guarantees," in *Proc. 38th Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA, L. K. Dillon, W. Visser, and L. A. Williams, Eds., New York, NY, USA: ACM, 2016, pp. 877–887.

[15] A. Busch, Q. Noorshams, S. Kounev, A. Koziolek, R. H. Reussner, and E. Amrehn, "Automated workload characterization for I/O performance analysis in virtualized environments," in *Proc. 6th ACM/SPEC Int. Conf. Perform. Eng.*, Austin, TX, USA. New York, NY, USA: ACM, 2015, pp. 265–276.

[16] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Commun. Statist.-Theory Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[17] J. Chen, W. Shang, A. E. Hassan, Y. Wang, and J. Lin, "An experience report of generating load tests using log-recovered workloads at varying granularities of user behaviour," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA. Piscataway, NJ, USA: IEEE Press, 2019, pp. 669–681.

[18] T. Chen, W. Shang, A. E. Hassan, M. N. Nasser, and P. Flora, "CacheOptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Seattle, WA, USA. New York, NY, USA: ACM, 2016, pp. 666–677.

[19] Z. Chen and W. Jiao, "A proactive self-adaptation approach for software systems based on environment-aware model predictive control," in *Proc. IEEE 22nd Int. Conf. Softw. Quality, Rel. Secur. (QRS)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 992–1003.

[20] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *Proc. 20th ACM Symp. Operating Syst. Princ. (SOSP)*, Brighton, U.K. New York, NY, USA: ACM, 2005, pp. 105–118.

[21] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. New York, NY, USA: Psychology Press, 2009.

[22] R. Colle, L. Galanis, Y. Wang, S. Buranawatanachoke, and S. Papadomanolakis, "Oracle database replay," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1542–1545, 2009.

[23] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. 26th Symp. Operating Syst. Princ.*, Shanghai, China. New York, NY, USA: ACM, 2017, pp. 153–167.

[24] D. Daly, W. Brown, H. Ingo, J. O'Leary, and D. Bradford, "The use of change point detection to identify software performance regressions in a continuous integration system," in *Proc. ACM/SPEC Int. Conf. Perform. Eng. (ICPE '20)*, Edmonton, AB, Canada. New York, NY, USA: ACM, 2020, pp. 67–75.

[25] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[26] B. Droesch, *Five Charts: How Coronavirus has Impacted Digital Grocery*. eMarketer. Accessed Oct. 21, 2020. [Online]. Available: https://www.emarketer.com/content/five-charts-how-coronavirus-has-impacted-digital-grocery

[27] S. Elnaffar and P. Martin, "Characterizing computer systems' workloads," 2002. Available: https://api.semanticscholar.org/CorpusID:13172124

[28] D. Freedman, R. Pisani, and R. Purves, *Statistics (international student edition)*, 4th edn. New York, NY, USA: WW Norton & Company, 2007.

[29] M. H. Fulekar, *Bioinformatics: Applications in Life and Environmental Sciences*. New York, NY, USA: Springer-Verlag, 2010.

[30] R. Gao, Z. M. Jiang, C. Barna, and M. Litoiu, "A framework to evaluate the effectiveness of different load testing analysis techniques," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation (ICST)*, Chicago, IL, USA. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2016, pp. 22–32.

[31] J. Garcia, "Change point evaluation in networking logs with periodicity filtering and bootstrapping," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 1–7.

[32] D. Gesvindr and B. Buhnova, "Performance challenges, current bad practices, and hints in PaaS cloud application design," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 4, pp. 3–12, 2016.

[33] G. Grano, C. Laaber, A. Panichella, and S. Panichella, "Testing with fewer resources: An adaptive approach to performance-aware test case generation," 2019, *arXiv:1907.08578*.

[34] A. Haghdoost, W. He, J. Fredin, and D. H. C. Du, "On the accuracy and scalability of intensive I/O workload replay," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, Santa Clara, CA, USA. USENIX Association, 2017, pp. 315–328.

[35] S. He, G. Manns, J. Saunders, W. Wang, L. L. Pollock, and M. L. Soffa, "A statistics-based performance testing methodology for cloud applications," in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf./Symp. Found. Softw. Eng. (ESEC/SIGSOFT FSE)*, Tallinn, Estonia. New York, NY, USA: ACM, 2019, pp. 188–199.

[36] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 4:1–4:35, 2015.

[37] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Hoboken, NJ, USA: Wiley, 1991.

[38] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," 2017, *arXiv:*1709.02280.

[39] Z. M. Jiang, A. Avritzer, E. Shihab, A. E. Hassan, and P. Flora, "An industrial case study on speeding up user acceptance testing by mining execution logs," in *Proc. 4th Int. Conf. Secure Softw. Integr. Rel. Improvement (SSIRI)*, Singapore. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2010, pp. 131–140.

[40] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Trans. Softw. Eng.*, vol. 41, no. 11, pp. 1091–1118, Nov. 2015.

[41] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automated performance analysis of load tests," in *Proc. 25th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Edmonton, Alberta, Canada. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2009, pp. 125–134.

[42] R. Krishna, M. S. Iqbal, M. Javidian, B. Ray, and P. Jamshidi, "CADET: A systematic method for debugging misconfigurations using counterfactual reasoning," 2020, *arXiv:2010.06061*.

[43] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies, "Whence to learn? Transferring knowledge in configurable systems using BEETLE," *IEEE Trans. Softw. Eng.*, vol. 47, no. 12, pp. 2956–2972, Dec. 2021.

[44] T. Lazariv and C. Lehmann, "Goodness-of-fit tests for large datasets," 2018. Available: https://api.semanticscholar.org/CorpusID:88523551

[45] P. Leitner and J. Cito, "Patterns in the chaos—A study of performance variation and predictability in public IaaS clouds," *ACM Trans. Internet Techn.*, vol. 16, no. 3, pp. 15:1–15:23, 2016.

[46] L. Liao et al., "Using black-box performance models to detect performance regressions under varying workloads: An empirical study," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4130–4160, 2020.

[47] M. Ma et al., "Diagnosing root causes of intermittent slow queries in large-scale cloud databases," *Proc. VLDB Endow.*, vol. 13, no. 8, pp. 1176–1189, 2020.

[48] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora, and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in *Proc. 14th Eur. Conf. Softw. Maintenance Reeng. (CSMR)*, Madrid, Spain. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2010, pp. 222–231.

[49] H. Martin, M. Acher, L. Lesoil, J. M. Jezequel, D. E. Khelladi, and J. A. Pereira, "Transfer learning across variants and versions : The case of linux kernel size," *IEEE Trans. Softw. Eng.*, vol. 48, no. 11, pp. 4274–4290, Nov. 2022.

[50] G. W. Milligan and M. C. Cooper, "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, vol. 50, no. 2, pp. 159–179, 1985.

[51] T. H. D. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. N. Nasser, and P. Flora, "Automated detection of performance regressions using statistical process control techniques," in *Proc. 3rd Joint WOSP/SIPEW Int. Conf. Perform. Eng. (ICPE'12)*, Boston, MA, USA. New York, NY, USA: ACM, 2012, pp. 299–310.

[52] F. Nielsen, *Hierarchical Clustering*, Cham, Switzerland: Springer, 2016, pp. 195–211.

[53] J. A. Pereira, M. Acher, H. Martin, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *J. Syst. Softw.*, vol. 182, 2021, Art. no. 111044.

[54] A. N. Pettitt, "A non-parametric approach to the change point problem," *J. Roy. Statist. Soc.: Ser. C (Appl. Statist.)*, vol. 28, no. 2, pp. 126–135, 1979.

[55] T. Raz, "The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling," *SIAM Rev.*, vol. 34, no. 3, pp. 518–519, 1992.

[56] N. Sandhya and A. Govardhan, "Analysis of similarity measures with wordnet based text document clustering," in *Proc. Int. Conf. Inf. Syst. Des. Intell. Appl. (INDIA)*, Visakhapatnam, India, 2012, pp. 703–714.

[57] H. Schulz, T. Angerstein, and A. van Hoorn, "Towards automating representative load testing in continuous software engineering," in *Proc. Companion ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, Berlin, Germany. New York, NY, USA: ACM, 2018, pp. 123–126.

[58] B. Seo, S. Kang, J. Choi, J. Cha, Y. Won, and S. Yoon, "IO workload characterization revisited: A data-mining approach," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3026–3038, Dec. 2014.

[59] W. Shang, A. E. Hassan, M. N. Nasser, and P. Flora, "*Proc. 6th ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, Austin, TX, USA. New York, NY, USA: ACM, 2015, pp. 15–26.

[60] S. M. Shariff, H. Li, C. Bezemer, A. E. Hassan, T. H. D. Nguyen, and P. Flora, "Improving the testing efficiency of selenium-based load tests," in *Proc. 14th Int. Workshop Automat. Softw. Test (AST@ICSE)*, Montreal, QC, Canada. Piscataway, NJ, USA: IEEE Press, 2019, pp. 14–20.

[61] P. Shivam, V. Marupadi, J. S. Chase, T. Subramaniam, and S. Babu, "Cutting corners: Workbench automation for server benchmarking," in *Proc. USENIX Annu. Tech. Conf.*, Boston, MA, USA. Boston, MA, USA: USENIX Association, 2008, pp. 241–254.

[62] C. Smith, *Software Performance Engineering*, Hoboken, NJ, USA: John Wiley & Sons Ltd., 2006, pp. 509–536.

[63] J. H. Stapleton, *Models for Probability and Statistical Inference: Theory and Applications*, vol. 277. Hoboken, NJ, USA: Wiley, 2008.

[64] B. Stevens, "Ocado is relaunching its app after being forced to scrap it in March due to 1000% jump in traffic." Charged Retail. Accessed Oct. 21, 2020. [Online]. Available: 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/

[65] J. Summers, T. Brecht, D. L. Eager, and A. Gutarin, "Characterizing the workload of a Netflix streaming video server," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Providence, RI, USA. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2016, pp. 43–54.

[66] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. N. Nasser, and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in *Proc. IEEE Int. Conf. Software Maintenance* (*ICSM*), Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2013, pp. 110–119.

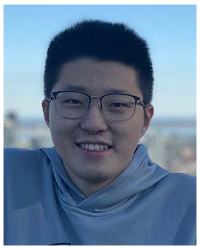
[67] M. D. Syer, W. Shang, Z. M. Jiang, and A. E. Hassan, "Continuous validation of performance test workloads," *Autom. Softw. Eng.*, vol. 24, no. 1, pp. 189–231, 2017.

[68] A. Tchana et al., "Self-scalable benchmarking as a service with automatic saturation detection," in *Middleware*, (Lecture Notes Computer Science), vol. 8275, New York, NY, USA: Springer-Verlag, 2013, pp. 389–404.

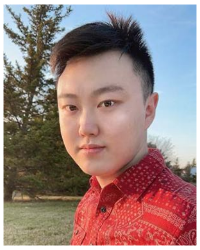
[69] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. Krcmar, "WESSBAS: Extraction of probabilistic workload specifications for load testing and performance prediction—A model-driven approach for session-based application systems," *Softw. Syst. Model.*, vol. 17, no. 2, pp. 443–477, 2018.

[70] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, "TeaStore: A micro-service reference application for benchmarking, modeling and resource management research," in *Proc. 26th IEEE Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, Milwaukee, WI, USA. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2018, pp. 223–236.

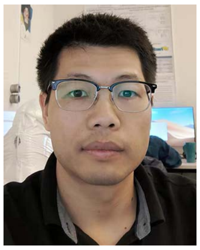
[71] K. R. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, 2016, Art. no. 9.
[72] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: Issues, an approach, and case study," *IEEE Trans. Softw. Eng.*, vol. 26, no. 12, pp. 1147–1156, Dec. 2000.
[73] X. Xiao, S. Han, D. Zhang, and T. Xie, "Context-sensitive delta inference for identifying workload-dependent performance bottlenecks," in *Proc. Int. Symp. Softw. Testing Anal. (ISSTA '13)*, Lugano, Switzerland, M. Pezzè and M. Harman, Eds., New York, NY, USA: ACM, 2013, pp. 90–100.
[74] N. J. Yadwadkar, C. Bhattacharyya, K. Gopinath, T. Niranjan, and S. Susarla, "Discovery of application workloads from network file traces," in *Proc. 8th USENIX Conf. File Storage Technol.*, San Jose, CA, USA: USENIX, 2010, pp. 183–196.
[75] F. Zhang, I. Keivanloo, and Y. Zou, "Data transformation in cross-project defect prediction," *Empircal Softw. Eng.*, vol. 22, no. 6, pp. 3186–3218, 2017.
[76] P. Zhang, S. G. Elbaum, and M. B. Dwyer, "Automatic generation of load tests," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Lawrence, KS, USA, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds., Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 2011, pp. 43–52.
[77] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proc. 41st Int. Conf. Softw. Eng.: Softw. Eng. Pract., ICSE (SEIP)*, Montreal, QC, Canada. Piscataway, NJ, USA: IEEE Press, 2019, pp. 121–130.



Yuanjie Xia received the master's degree from Concordia University. He is a Ph.D. Student with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interest includes software performance testing.



Lizhi Liao (Graduate Student Member, IEEE) received the B.Eng. from Chongqing University of Posts and Telecommunications, and the M.Eng. degree from Concordia University. He is a Ph.D. Student with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, supervised by Weiyi Shang. His research interests include software performance engineering, software log mining and mining software repositories.



Jinfu Chen received the M.Sc. degree from the Chinese Academy of Sciences, and the Ph.D. degree from Concordia University, Canada. He is an Associate Professor with the School of Computer Science, Wuhan University. His research interests include empirical software engineering, software performance engineering, performance testing, and software log mining. For more information, see https://jinfuchen.github.io/jinfu.

Heng Li received the B.Eng. degree from Sun Yat-sen University, China, the M.Sc. degree from Fudan University, China, and the Ph.D. degree in computing from Queen's University, Canada. He is an Assistant Professor with the Department of Computer and Software Engineering, Polytechnique Montreal. Before his academic career, he worked in the industry for years as a Software Engineer with the Synopsys and a Software Performance Engineer with BlackBerry. He and his students' research in the MOOSE lab (moose.polymtl.ca) address practical challenges in software monitoring, performance engineering, log analysis, intelligent operations of software systems (AIOps), software analytics, and quality engineering of traditional and emerging systems (e.g., AI-based and quantum systems).

Weiyi Shang received the B.Eng. from Harbin Institute of Technology, and the M.Sc. and Ph.D. degrees from Queens University, Canada. He is an Associate Professor with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests include big data software engineering, software engineering for ultra-large-scale systems, software log mining, empirical software engineering, and software performance engineering. His work has been published at premier venues such as *ICSE*, *FSE*, *ASE*, *ICSME*, *MSR*, and *WCRE*, as well as in major journals such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *EMSE*, *JSS*, *JSEP*, and *SCP*. His work has won premium awards, such as two SIGSOFT Distinguished Paper Award at ICSE 2020 and 2013. His industrial experience includes helping improve the quality and performance of ultra-large-scale systems in BlackBerry. Early tools and techniques developed by him are already integrated into products used by millions of users worldwide.