Towards an Efficient Performance Testing through Dynamic Workload Adaptation

Osvaldo Huerta-Guevara, Vanessa Ayala-Rivera, Liam Murphy, and A. Omar Portillo-Dominguez

Lero@UCD, School of Computer Science, University College Dublin, Ireland
osvaldo.huertaguevara@ucdconnect.ie, {vanessa.ayalarivera, liam.murphy, andres.portillodominguez}@ucd.ie

Abstract. Performance testing is a critical task to ensure an acceptable user experience with software systems, especially when there are high numbers of concurrent users. Selecting an appropriate test workload is a challenging and time-consuming process that relies heavily on the testers' expertise. Not only are workloads application-dependent, but also it is usually unclear how large a workload must be to expose any performance issues that exist in an application. Previous research has proposed to dynamically adapt the test workloads in real-time based on the application behavior. By reducing the need for the trial-and-error test cycles required when using static workloads, dynamic workload adaptation can reduce the effort and expertise needed to carry out performance testing. However, such approaches usually require testers to properly configure several parameters in order to be effective in identifying workload-dependent performance bugs, which may hinder their usability among practitioners. To address this issue, this paper examines the different criteria needed to conduct performance testing efficiently using dynamic workload adaptation. We present the results of comprehensively evaluating one such approach, providing insights into how to tune it properly in order to obtain better outcomes based on different scenarios. We also study the effects of varying its configuration and how this can affect the results obtained.

Keywords: Software Engineering, Performance Testing, Performance Bug, Workload, Web Systems and Applications

1 INTRODUCTION

Performance testing plays a critical role in the software industry in order to successfully engineer reliable systems and guarantee the best experience for the final users. When a system exhibits failures or errors, it can cause severe damages to companies such as economic and reputational losses. Moreover, the risk of suffering performance degradation in systems is exacerbated with the increase of services residing in the cloud, which must support millions of users interacting with the system at the same time through web or mobile applications [11].

The goal of performance testing is to evaluate how well an application can perform under a particular workload [21,30]. However, it is difficult to generate effective test cases that can expose performance issues promptly as it requires testers to evaluate multiple combinations of workloads, actions, and data [24]. Additionally, a large number of enterprise systems that were developed as data-centric applications are now deployed as software as a service (SaaS) in the cloud. The problem with services in the cloud is that they are difficult to test because their backend components are distributed, thus, making it difficult to emulate the flow of the application, especially when the services are consumed by multiple users at the same time [16,31].

Conducting performance testing typically requires a lot of effort and expertise from testers. The performance of each application is unique and although the application could be the same, different versions and releases are still distinctive, so the set of performance tests needs to be updated accordingly [34]. This problem emphasizes the need for an expert on the application under test in order to get some insights regarding its weakness [38]. However, this situation could lead to dependencies and bottlenecks in the workflow of the testing team [2]. Moreover, due to the strict demands for shorter time-to-market products, testers are required to automate as much as possible their test plans so they can move at the same rate of emerging methodologies such as Agile, Continuous Integration, and Continuous Deployment. In addition, complete coverage of performance testing is not always achievable, as there are no automated oracles that can assure that a system is completely safe in terms of performance [40]. Load testing, an important part of performance testing, is the standard approach for assessing how an application behaves under different loads to find scenarios where it suffers performance issues (e.g., CPU bottlenecks, deadlocks) leading to high response time or low throughput [24] that affect the user experience.

One important problem in load testing is that most of the tools used to assess the performance of applications depend on static (i.e., pre-configured) workloads [32] such as Apache JMeter [1] and IBM RPT [2]. A major disadvantage of these tools is that, in order to define a correct load to stress the applications, they require human expertise and several runs of trial-and-error, hence, taking a lot of time. Additionally, manually designing, executing, and analyzing the values of a test workload can be very difficult due to the scale of the test, increasing the risk of overlooking problems or potential bugs [10].

An innovative approach in the literature is to adapt the workloads, in real time, based on the behavior of the application under test. These types of approaches aim to reduce the effort and expertise needed to carry out performance testing efficiently. A representative example of this type of approach is DYNAMO [3], which can automatically find an adequate workload for a web application without requiring multiple test runs while increasing the number of performance bugs found. This is achieved by dynamically defining the appropriate workloads on the fly based on the analysis of the intermediate testing results.

[1] http://jmeter.apache.org/
[2] https://www.ibm.com/us-en/marketplace/ibm-rational-performance-tester

However, despite its potential benefits, one disadvantage of DYNAMO is that users are required to define several parameters in order to configure it properly (otherwise, there is a risk that results may vary and have negative impacts in the testing process), a characteristic which might hinder its usability and effectiveness. Hence, there are improvements that can be done in terms of identifying the proper combination of the different variables involved in the configuration of DYNAMO, and of this type of approach in general.

To address the above issue, the contributions of this paper are the following:

1. A comprehensive evaluation of the DYNAMO approach, assessing a broad range of configuration alternatives to conduct performance testing efficiently using dynamic workload adaptation.
2. A set of rules of thumb, derived from our experiments using DYNAMO, that advise practitioners in the appropriate usage of these types of approaches.
3. An insightful analysis of how the different configuration variables can affect DYNAMO's behavior and its results.

The rest of this paper is structured as follows: Section 2 presents the background and related work. Section 3 explains the DYNAMO approach, while Section 4 describes the experimental evaluation and results obtained from the use of DYNAMO. Finally, Section 5 presents the conclusions and future work.

2 BACKGROUND AND RELATED WORK

Performance testing is a type of testing whose objective is to ensure that an application-under-test (AUT) is capable of running its business operations under certain varieties of loads [21,29]. As shown in Fig. 1, a performance tester typically configures a Load tool (e.g., Apache JMeter) to run a test for a certain amount of time, usually several hours or even days in the case of industrial applications. In this context, a test workload is comprised of many concurrent virtual customers and a series of operations such as logins, searches, and click events to emulate the real use of the AUT. In order to identify performance issues, monitoring tools collect performance-related information, such as throughput and response time, during the execution of the test. Finally, the tester analyzes the outputs from the monitoring tool to identify trends in the collected data and determine possible bottlenecks or failures in the system.

Software testing automation has become crucial for organizations as oftentimes they require faster deployments and reduced time-to-market. Therefore, testing automation is incorporated as a crucial step in the software development life cycle to minimize the appearance of bugs in production code and reduce the testing efforts. Test suits are collections of test cases designed to evaluate the behavior of software applications. Nevertheless, testing is complex and expensive, then the approach presented in [20] explains how to focus the testing using a property-oriented technique to reduce the complexity of the test suits and yet obtain complete coverage. The authors of [8] propose an architecture to automate the performance testing of microservices by monitoring key indicators

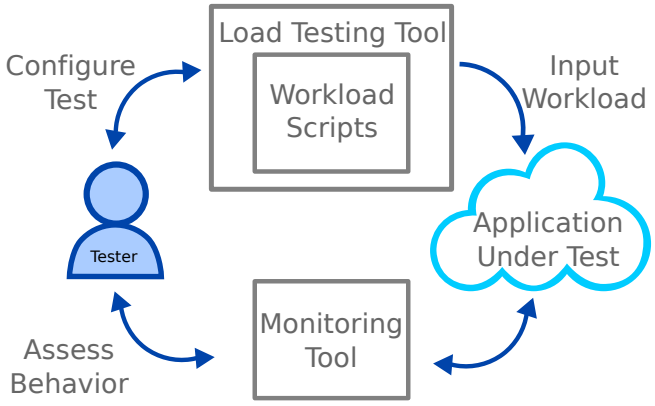

Fig. 1. Performance Testing - Contextual View

like the throughput, response time, and availability, which are used to detect bottlenecks. A framework is proposed by [23] to combine the use of conformance testing and load testing in order to save time and efforts during the validation of web services. Traditionally these types of testing are run by different teams verifying distinctive inputs and outputs. This work is an extension of [25] implementing timed automata and a test generation algorithm to discover bugs and system degradations that appear during high loads of users because of the lack of optimization in the code. Also, in [7], the authors define metrics to test web applications focusing on the stability, quality, and the difficulties involved to automate a test case. Other approaches use model-based testing to reduce the test case derivation efforts [13,14,26]. In [14], the author examines techniques for model-based testing of web applications and websites that apply statistical usage models to generate and evaluate appropriate test suites. Similarly, the works in [13,26] explore the automation of test case derivation to facilitate testing for automotive software.

Conducting performance testing of software systems is particularly challenging as it requires a significant amount of effort to carry out. For instance, identifying the limits of a system is not trivial. Testers require to have previous experience and knowledge about the system to define the correct workloads [27]. Another problem is that there are no methods to assess the effectiveness of performance testing. So the authors of [35] propose the use of mutation testing during performance testing to measure the efficiency of the test suits to discover errors. Mutation testing is a type of test that involves the introduction of faults (mutants) into the code through some predefined rules to unveil errors in the system. Effective performance testing techniques are required to cover a system from high-priority bugs. In [37], the authors propose a model-based approach to test web applications generating synthetic workloads based on the data dependencies and transactions of the application under test. The authors in [4] explain that it is not feasible to test all the possible inputs in multi-core applications. Therefore, automation is applied to generate appropriate test cases using

genetic algorithms. The work presented in [15] defines a declarative approach to incorporate performance testing into continuous software development. With such approach it is possible to model goal-oriented performance tests into the development life cycle.

More recent approaches incorporate new emerging technologies. Machine learning, for instance, uses black box testing, which identifies performance problems based on the feedback and outputs provided by the application. Other methods take into account the use of cloud resources [17] to calculate how many resources are needed to run an application without affecting the performance. In [18], the authors explore the detection of software defective modules using clustering-based under-sampling and artificial neural network. Metamorphic testing [36], on the other hand, aims to reduce the complexity of the testing configuration, the human-based expertise and the use of diagnostic tools to analyze variables like the optimization of Garbage Collection [29].

In general, the process of defining an appropriate test workload to detect performance issues in systems is a difficult problem. Workloads must be representative of the application under test and they must reflect the variability of human behavior [12]. Moreover, workloads are related to specific variables depending on the application under test and it is difficult for a human to identify patterns [5]. Despite there are tools to generate synthetic workloads for testing [9,39] these rely on static techniques for workload generation. This type of technique requires to invest more time and resources to identify an adequate workload for the system. With static workloads, a predetermined number of users are defined per run, whereas in an adaptive approach various diverse workloads will be generated in-test until a high workload (maximum number of users) is found such that the system can handle it without crashing. Other performance testing approaches are based on static code analysis. For instance, the authors of [34] determine the performance tests based on commits and the usage of unit-tests. Meanwhile, the work presented in [33] describes the performance analysis of software systems as a comparison of two versions of software and their performance results to find possible (regression) bugs. In response to these limitations, approaches that dynamically adapt the workload have been proposed [3,22]. This paper conducts an evaluation of DYNAMO because of its advantages of adjusting the workload on the fly using the analysis of key performance metrics to create a customized workload for the AUT with minimal knowledge or experience from the tester. Moreover, in relation to bug accuracy, previous work [3,22] has proved the benefits of using DYNAMO to find bugs in comparison with static approaches.

3 DYNAMO OVERVIEW

The goal of DYNAMO [3] is to help testers to define appropriate workloads without a trial-and-error process. To achieve this, DYNAMO relies on the use of adaptive workloads. Applying adaptive workloads can help to reduce the time invested to find a suitable workload, therefore saving money and resources. This is because an inappropriate low test workload can overlook performance issues

in the application [21]. In contrast, using a "too" high load can saturate the system and prevent the detection of some issues due to critical failures that may arise on the system (as presented in Fig. 2 for illustrative purposes only, as the actual workload curves are application-specific).

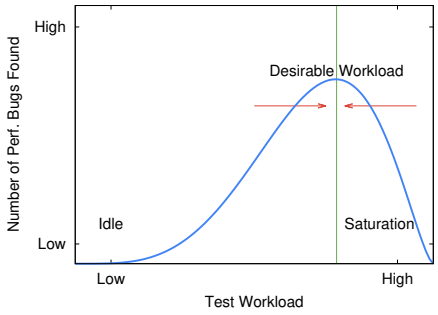

Fig. 2. Bugs vs Test Workload Example

DYNAMO works by constantly monitoring key performance indicators (during the test run execution) to automatically adjust the workload as required. To adjust the workload, a set of the functional transactions are incremented, based on a threshold evaluation that identifies those transactions to be increased, as well as by how much. The transactions that will be stressed are called workload sensitive (WKS). DYNAMO works in two phases: Phase 1 (Ph1), whose objective is to identify the workload sensitive transactions (WKS) involved in the test, and Phase 2 (Ph2), whose goal is to exercise the WKS as much as possible, while avoiding the saturation of the system (as shown in Fig. 3).

During Ph1, DYNAMO conducts two test runs (TR1, TR2) in order to identify the WKS transaction. In TR1, a known low workload is used, while a higher workload (w.r.t. TR1) is used in TR2. After TR1 and TR2 have finished, DYNAMO calculates the performance differences (deltas) per transaction between both runs in order to identify which transactions are the most workload-sensitive. For example, a basic welcome page could be a non-sensitive transaction because its content is static, while executing a login operation is typically more sensitive due to all the internal process triggered to verify a user (e.g., interfacing with a Single Sign-On service). Finally, the transactions are sorted in descending order of their deltas, and the leading ones are considered WKS.

In Ph2, DYNAMO keeps monitoring the performance of the WKS transactions (identified in Ph1) during the rest of the test run execution. Furthermore, their performance is iteratively evaluated, using an adjustment strategy, to identify those transactions that need a workload increment. DYNAMO currently supports 3 strategies to perform the workload adjustments: *Min*, which increases the WKS transactions with the best performance; *Max*, which increases the WKS transactions with the worst performance; and *Random*, which selects a random set of WKS transactions to be adjusted.

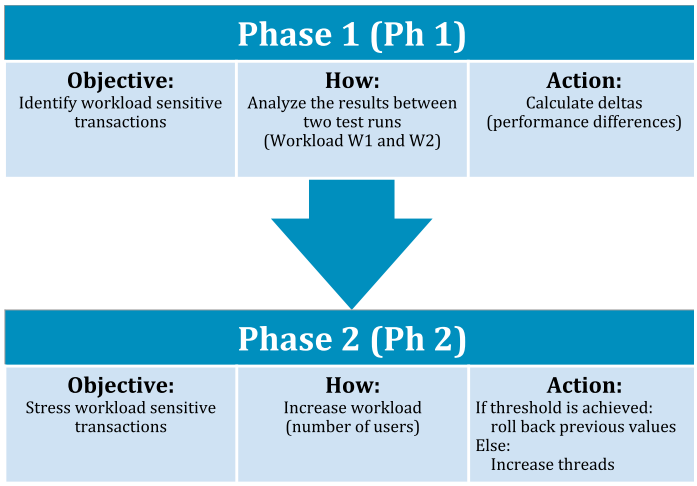

Fig. 3. DYNAMO's Phases Summary

In terms of configuration, DYNAMO requires several user inputs: (1) the test duration (e.g., 1 day). (2) the duration ratio between phases (i.e., 50/50%). (3) Two known seed workloads (WK1,WK2), one low (e.g., 1 user) and the other relatively higher than the first (e.g., 10 users). They are used by Ph1 to conduct the two calibration test runs (i.e., TR1, TR2). (4) A percentage of transactions of interest (%WKS), used by Ph1 to define how many (starting from the top) of the sorted transactions will be tagged as WKS. (5) A sample interval (SI), used by Ph2, to define how often the performance of the workloads will be evaluated (e.g., 5 min). (6) An error rate threshold (ERT), used by Ph2, to define what is considered saturation (e.g., 90%). (7) An adjustment strategy (ADS), used by Ph2, to identify the transactions whose workloads need to be increased. (8) A workload increment (WKINC) to define how big the increment will be (e.g., 20 users). (9) A percentage (%WKINC) to define how many WKS transactions will be increased, based on the chosen adjustment strategy (i.e., 30%).

4 EXPERIMENTAL EVALUATION

Our experiments aimed to evaluate DYNAMO's accuracy, in terms of both WKS labeling and bug finding, with a special emphasis on any trade-offs existing between the accuracies w.r.t. DYNAMO's configuration. Specifically, the conducted experiments addressed the following research questions:

- **RQ**$_1$. *Which is the best ratio (between DYNAMO Ph1 and Ph2) w.r.t. WKS labeling accuracy?*
- **RQ**$_2$. *Which is the best ratio (between DYNAMO Ph1 and Ph2) w.r.t. bug finding accuracy?*

In the following paragraphs, we describe the experimental setup used as well as the obtained results.

4.1 Experimental Setup

The test environment consisted of two virtual machines (VMs) running in isolated conditions to avoid noise during the experiments. One VM worked as the client running JMeter and DYNAMO. The other VM ran a web server hosting the application-under-test (AUT). The client VM had 2 virtual CPUs at 2.20 GHz, 4 GB of RAM and 150 GB of hard disk, running Ubuntu 14.04 with OpenJDK 1.7 with a 1.6 GB heap. This VM also used Apache JMeter 3.2 (a leading open-source tool used for load testing [3]) and the latest version of DYNAMO [3]. The server VM had 20 virtual CPUs at 2.20 GHz, 50 GB of RAM and 150 GB of hard disk, running Ubuntu 14.04 with IBM JVM 1.8 with a 25 GB heap and Apache Tomcat 7.0 as web server container. Additionally, IBM WAIT was used as diagnostic tool due to its robust capabilities to detect performance errors [29,41] such as memory leaks and resource bottlenecks. The AUTs used

Table 1. DaCapo Programs

Name	Description
avrora	It simulates a program running on a grid.
batik	It processes vector-based images.
eclipse	It executes tests in an Eclipse development environment.
fop	It generates PDF files.
h2	It runs banking transactions in a database system.
jython	It runs Python scripts.
luindex	It indexes documents.
lusearch	It runs keyword searches over a data corpus.
pmd	It reviews a set of Java classes.
sunflow	It renders images.
tomcat	It runs queries in a Tomcat server.
tradebeans	It runs stock transactions through Java Beans.
tradesoap	It runs stock transactions through SOAP.
xalan	It transforms XML files into HTML files.

were DaCapo[4] and SPECJVM [5], which are two well-known Java benchmarks that are highly used in the literature. In order to utilize them as web applications, we used a servlet-based tool that enables their access and execution via web, so that each program (within the benchmark) can be used as a different functional transaction [28]. The Dacapo test plan consisted of its 14 benchmark programs (shown in Table 1) configured to run using their *small* size, 1 iteration, and 2 threads. Meanwhile, due to the complexity of SPECJVM, some benchmarks were executed by their specific sub-benchmarks (i.e., scimark: fft, lu, sor, sparse and monte_carlo) to create a test plan of 13 operations, as shown in Table 2. The configuration used to run the benchmarks was 0 sec warm-up time, 10 sec iteration, small data sets, and 2 threads.

Furthermore, to monitor the key performance indicators, we used JMeter for the throughput (tps), error rate (%) and, response time (ms) while nmon[6] col-

[3] http://jmeter.apache.org/
[4] http://dacapobench.org/
[5] https://www.spec.org/jvm2008/
[6] http://nmon.sourceforge.net/

lected CPU utilization (%) and memory consumption (MB). Additionally, WAIT was used to monitor the number (and criticalness) of the identified performance issues.

Table 2. SPECJVM Programs

Name	Sub-benchmark	Description
compiler	.compiler	It compiles Java source files.
compress	all	It runs a data compression alg.
crypto	.rsa	It encrypts/decrypts files.
MPEGaudio	all	It decodes audio files.
scimark	fft, lu, sor, sparse and monte_carlo	It executes floating point ops.
serial	all	It serialises/deserialises objects and primitives.
startup	.helloworld	It runs a hello word program.
sunflow	all	It runs visualization operations.
XML	.transform	It transforms XML documents.

The response time during Ph1 determines which transactions are WKS. In Ph2, the error rate evaluates if the overall workload has reached the threshold and decides whether to increase the load applying one of the three available strategies or to rollback to a previous value and try a different combination to increase the workload. CPU utilization and Memory consumption were monitored to validate the minimal overhead introduced by the use of DYNAMO in the client machine.

DYNAMO has several configuration options to generate adaptive workloads. Therefore, choosing a different combination of parameters can lead to distinct results. The values used in the experiments were defined to cover as many scenarios as possible. The configurations we used for DYNAMO are shown in Table 3.

It is worth mentioning that the test durations and the workload values were defined, per AUT, based on the maximum capacity supported by our test environment.

4.2 Results

To answer RQ_1, our analysis initially focused on assessing how well Ph1 could identify WKS transactions. To accomplish this, we firstly executed DYNAMO using all the in-scope Ph1/Ph2 ratios. Then, the obtained results were compared against a baseline, which was used to calculate the accuracy of Ph1 for labeling WKS transactions. The baseline was created following the traditional performance testing approach, typically utilized in the industry, of using static (i.e., pre-configured) test workloads [19]. Thus, the baseline was calculated with the average results of 10 test runs carried out (per AUT) using a range of static workloads (their consolidated response time information is shown in Table 4).

To maximize the representativeness of the baseline (i.e., covering the full spectrum of potential test workloads, starting with a close-to-idle scenario and ending with a saturated environment), both AUTs were initially tested with a very low workload (i.e., 1 and 2 concurrent users for SPECJVM and DaCapo,

Table 3. Experimental Configuration Parameters for DYNAMO

Phase	Parameter	Value
Initial Settings	(1) Test duration	100 min for SPECJVM and 200 min for DaCapo
	(2) Ph1 and Ph2 ratios	10/90, 20/80, 40/60, 60/40, and 80/20
Phase 1 Settings	(3) Calibration workloads [WK1, WK2]	[1,10] for SPECJVM and [2,20] for DaCapo
	(4) Number of transactions considered as WKS (%WKS)	30% and 50%
Phase 2 Settings	(5) Sample interval (SI)	5 min
	(6) Error rate threshold (ERT)	90%
	(7) Adjustment strategy (ADS)	*Min*
	(8) Workload increment (WKINC)	5 users
	(9) WKS transactions to be increased (%WKINC)	50%

respectively). Then, the workload was gradually increased, per test run, by 100% (w.r.t. the initial workloads used), until reaching a high enough workload that provoked saturation in the system (i.e., 10 and 20 concurrent users for SPECJVM and DaCapo, respectively).

Table 4. Baselines Sorted by Response Time

Trans. ID	**DaCapo** Average response time (ms)	Trans. ID	**SPECJVM** Average response time (ms)
10	34,076.66	01	15,514.34
09	37,903.85	07	26,085.40
08	42,888.26	06	26,319.86
14	43,272.39	08	27,039.00
04	51,381.02	04	29,763.84
07	62,803.12	02	31,332.79
01	63,576.74	03	32,090.00
13	69,162.31	10	33,366.21
02	70,003.78	09	34,099.33
12	73,513.73	05	45,603.02
11	82,565.42	12	48,367.00
05	107,204.49	11	71,762.47
06	118,468.68	13	354,635.71
03	208,623.71	-	-

Furthermore, we calculate of the accuracy of Ph1 to correctly label WKS transactions. It was based on a set B of baseline transactions and a set S of labeled transactions (which is the main output of Ph1) in order to determine how many of the S transactions belong to the B set. Let B be a finite set of (baseline) transactions $B = \{b_1, b_2, \ldots, b_n\}$ and $n(B)$ its cardinality. Let S be the set of

sample transactions $S = \{s_1, s_2, \cdots, s_n\}$ sorted in descending order according to their response time. Now, a function $f(s_i)$ is defined to indicate whether a given sample transaction s_i will be taken into account when calculating the percentage of accuracy (as depicted in Equation 1).

$$f(s_i) = \begin{cases} 0, & \text{if } s_i \notin B \\ 1, & \text{if } s_i \in B \end{cases} \tag{1}$$

Then, the percentage of accuracy can be computed as the sum of the values for all $s_i \in S$ from the previously defined function, all divided by the cardinality of B. This is depicted in Equation 2.

$$\text{percentage of accuracy} = \frac{\sum_{s_i \in S} f(s_i)}{n(B)} \tag{2}$$

Figure 4 depicts the results obtained using DaCapo with a %WKS of 30%. It can be seen that there is a tendency to achieve a higher accuracy when using a higher duration ratio. However, by using a Pareto ratio of 20/80, it is still possible to achieve an accuracy above 75%, which is very close to the accuracies obtained when using higher ratios (e.g., 60/40 or above). This demonstrates that it is feasible to use a lower Ph1/Ph2 duration ratio without compromising the accuracy of the WKS labeling. A slightly lower ratio appears when using 40% of ratio during Ph1, but this value is explained due to the closer delta values of the transactions in the 30% range. An additional advantage of using a ratio of 20% (in Ph1) is that it leaves more time for Ph2, hence increasing the possibility to find more performance bugs.

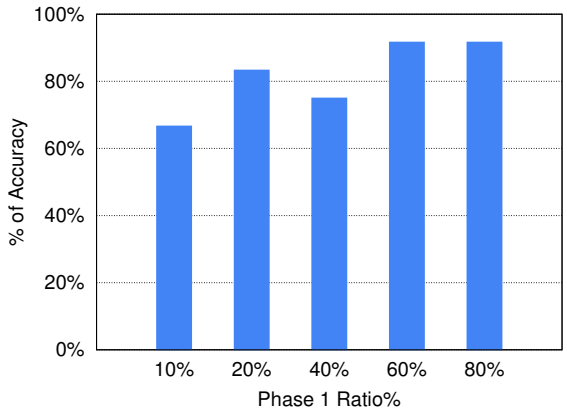

Fig. 4. Accuracy of Ph1 for DaCapo Using a %WKS of 30%

We observed a similar trend with the SPECJVM results, where duration ratios from 20/80 onwards had a good accuracy (above 75%). This supports our finding that the Pareto ratio can achieve a high level of accuracy. Only when

using a ratio of 10% in Ph1, the accuracy drops. This is the result of having only a short amount of time in Ph1, which makes impossible to label with high accuracy the WKS transactions. Finally, it is important to highlight that, in the cases where the accuracy did not reach 100%, the missed transactions were the ones closest to the WKS transactions (according to the baseline table). These results also reinforced the finding that DYNAMO's accuracy tends to be high and that the Pareto ratio can achieve a high accuracy without spending "too much" time in Ph1 (as it would not bring any real benefit).

To answer RQ_2, the focus of our analysis shifted to assess how well Ph2 could identify performance issues. With that aim in mind, we fed WAIT (i.e., our chosen diagnosis tool) with snapshots of the JVM state (i.e., Javacores [6]) sampled during the execution of the test runs. These samples were taken in intervals of 30 seconds, following common industrial standards [1]. WAIT provided a report with all the identified bugs sorted by their frequency of occurrence, information that was used to classify the bugs as low or high relevant. A bug was considered as high relevant if it occurred above 5% (of the test run duration). Alternatively, it was considered as low relevant. Regarding bug finding, DYNAMO's Ph2 en-

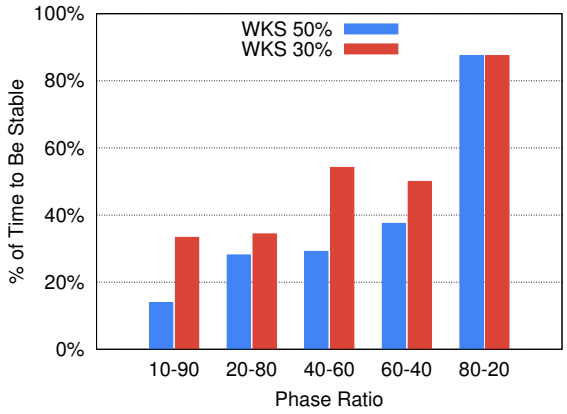

Fig. 5. Percentage of Time in Ph2 to Find a Stable Load

hances this process by finding a load big enough to stress, as much as possible, the AUT without reaching a point of saturation (called stable load). To define the appropriate load, DYNAMO takes samples of the response times, calculates the average of the error rates, and makes load adjustments on the fly. Figure 5 depicts the time taken to achieve a stable load per phase ratio. There, it can be noticed how an earlier stable load was found when using a ratio of 10/90. Yet, the second best result came from the load with a Pareto ratio of 20/80, where the stable load was found when the lapsed time was at 34% (for WKS of 50%) and 28.13% (for WKS of 30%) of the total time of the workload. This is in comparison with spending 87% of the time for the 80/20 ratio.

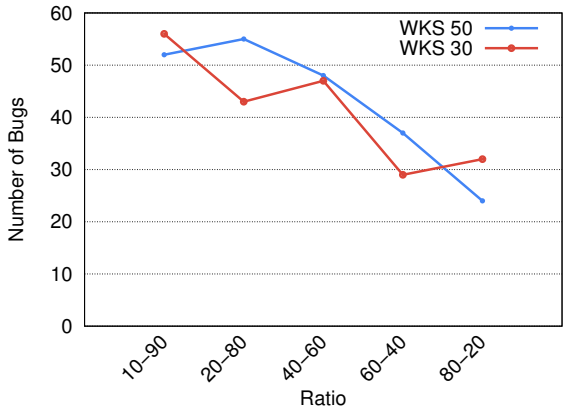

Fig. 6. DaCapo Bugs per %WKS Ratios

When analyzing the number of performance bugs found during Ph2 across the different duration ratios, we observed that the number of bugs decreased when the time spent in Ph2 was lower (as depicted in Fig. 6). Additionally, if the ratio in Ph2 is lower, the time required to find a stable load is longer (as shown in Fig. 5). Consequently, there is less time to find bugs. In terms of load, when a %WKS of 50% was used, a stable load was found earlier and also the number of adjustments was higher. This explains why this value of %WKS found more performance bugs during Ph2 (as shown in Fig. 6) due to the use of higher loads of users as depicted in Fig. 7.

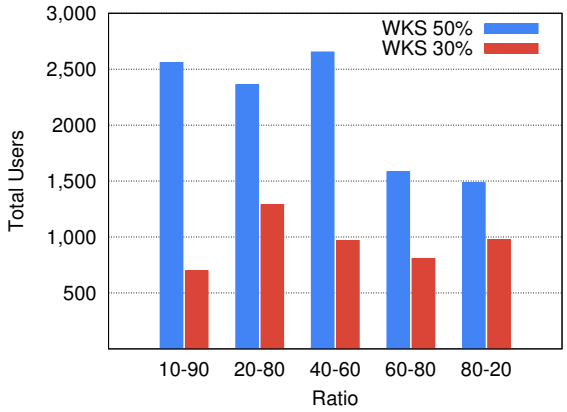

Fig. 7. Number of Users During Ph2

With these results, it is possible to assert that using a higher load is better to find bugs than using a lower load for a long time. This observation denotes the importance of finding a high enough load to stress a system, but without reaching a saturation point (as that would negatively affect the bug-finding process).

Also, it is crucial to consider the distribution of the WKS transactions during Ph2, as a higher %WKS ratio (e.g., 50%) will work better. This is because there will be a broader range of eligible transactions to increase the load, rather than overstressing the server with a load composed of a limited number of transactions. In summary, during Ph2 a higher %WKS is recommended to find a better workload to stress the system and find more bugs.

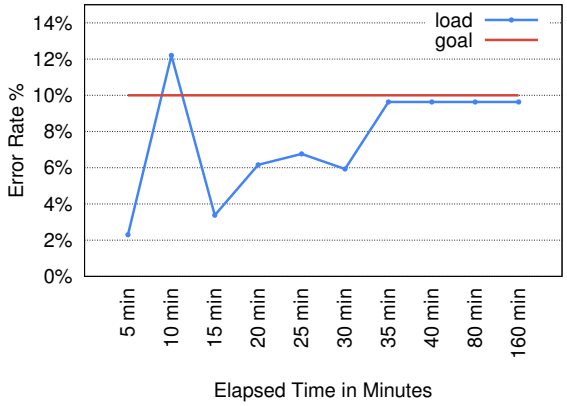

Fig. 8. Workload Adjustment for Ratio 20/80

It is also worth mentioning that the workload adjustments during Ph2 are important because they are part of the core logic of DYNAMO. This illustrated in Fig. 8, where one can see the adjustments made according to the error rate, as well as the process of rollback. During the initial adjustments, the error passed the threshold, then other adjustments were made following the operations shown in Table 5 to define the final workload.

Table 5. Adjustments

Operation	Transactions
Increase	[T11, T3]
Rollback	[T11, T3]
Increase	[T5, T6]
Increase	[T5, T6]
Increase	[T5, T6]
Increase	[T5, T6]
Rollback	[T5, T6]

Furthermore, the bugs considered as highly relevant (due to its frequency of occurrence) were stable in all the phase ratios while using both 50% and 30% as %WKS, as depicted in Fig. 9. However, bugs with low relevance changed depending on the ratio. This is because if less time is spent in Ph2, there is not

enough time to get data samples to analyze the relevance of these bugs. With these results, the tradeoffs of changing the ratios were reflected in the finding of low relevant bugs, while the high relevant bugs kept the same regardless of the ratio. Also, the Pareto ratio generated results that were not far from the ratios that spent more time in Ph2 (e.g., 90/10). What is more important, the Pareto ratio did not compromise the accuracy in WKS labeling during Ph1.

In relation to bug accuracy, previous work [3,22] has proved that DYNAMO is better for finding bugs than traditional testing approaches. Therefore, our analysis concentrated on comparing the results from the different ratios (w.r.t. Ph2). We conclude that Ph2 was affected by two main variables: The Ph2 ratio and %WKS. As shown in Fig. 8, the Pareto ratio obtained good results (compared to the other ratios), also the %WKS of 50% found more bugs. This was also confirmed by Fig. 7, which showed that the %WKS of 50% had a big load, increasing the chances to find bugs. Lastly, lower ratios in Ph2 found a large number of bugs; yet, most of them were categorized with low relevance and were associated with a drop in accuracy caused by having less time during Ph2.

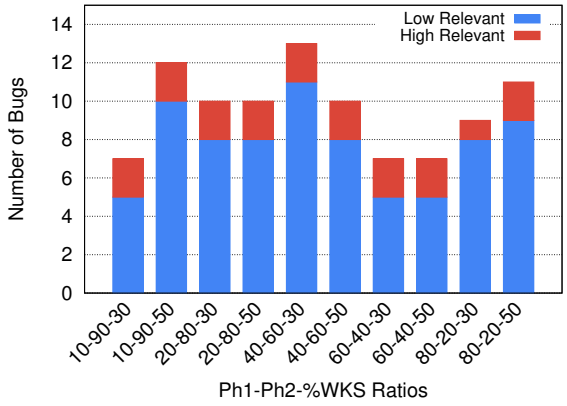

Fig. 9. DaCapo Bug Classification Distribution

5 CONCLUSIONS AND FUTURE WORK

Identifying the appropriate load to stress a system to expose performance issues is challenging. This task requires a lot of effort and experienced testers trying different static workloads. Dynamic workload adaptation approaches make possible to find an appropriate workload automatically. However, they require several configuration parameters. In this paper, we have comprehensively explored such parameters using a representative example of this type of approaches called DYNAMO, which works in 2 phases. After our evaluation, we were able to prove the accuracy of DYNAMO's Phase1 (Ph1) w.r.t. WKS labeling, where all the ratios showed an accuracy above 75% except the lowest one (i.e., 10-90). This was the consequence of the low time spent in Ph1. In DYNAMO's Phase2 (Ph2), we found that using a higher ratio of WKS transactions led to more transactions

to be adjusted and thereby a generated workload closer to the saturation point. Moreover, it was possible to perceive that higher ratios during Ph2 provided more time for workload adjustment, which left more time to run the "ideal" workload. In addition, we showed that the Pareto ratio offered good results between different ratios without compromising bug finding (in Ph2) and WKS accuracy (in Ph1). Finally, we found that the bug finding accuracy during Ph2 was better when higher ratios were used, as this led to more adjustments and testing time used to discover more bugs. These results offer practitioners a valuable reference regarding the use of DYNAMO and the benefits of implementing a dynamic workload adaptation strategy for performance testing.

In future work, we plan to explore other approaches of dynamic workload adaptation to perform a comparative study and derive more guidelines for practitioners. Moreover, we intend to incorporate machine learning techniques to study the behavior of the approaches and then be able to make automatic recommendations of the configuration parameters based on the characteristics learned from the applications under test. Additionally, incorporate information about the structure and behavior of the web application to focus the testing on sensitive code that could trigger performance bugs. Finally, we plan to incorporate the approaches in easy-to-use JMeter plugins with a friendly graphical interface so it can be more easily adopted by performance engineering practitioners as many of the approaches are built as basic research prototypes running in the command line.

Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

References

1. Altman, E., Arnold, M., Fink, S., Mitchell, N.: Performance analysis of idle programs. ACM SIGPLAN Notices **45**(10) (Oct 2010)
2. Angelopoulos, V., Parsons, T., Murphy, J., O'Sullivan, P.: GcLite: An Expert Tool for Analyzing Garbage Collection Behavior. In: 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops. pp. 493–502. IEEE (2012)
3. Ayala-Rivera, V., Kaczmarski, M., Murphy, J., Darisa, A., Portillo-Dominguez, A.O.: One Size Does Not Fit All. In: ICPE '18. pp. 211–222. ACM Press, New York, New York, USA (2018)
4. Aziz, M.W., Shah, S.A.B.: Test-data generation for testing parallel real-time systems. In: IFIP - ICTSS '15. pp. 211–223. Springer (2015)
5. Basak, J., Wadhwani, K., Voruganti, K.: Storage Workload Identification. ACM Transactions on Storage **12**(3), 1–30 (may 2016)
6. Bourne, D.A., Chung, A.A., Price, D.L.: Capturing trace information using annotated trace output (May 31 2016), US Patent 9,355,002

7. Bures, M., Miroslav: Metrics for automated testability of web applications. In: CompSysTech '15. pp. 83–89. ACM Press, New York, New York, USA (2015)
8. de Camargo, A., Salvadori, I., Mello, R.d.S., Siqueira, F.: An architecture to automate performance tests on microservices. In: iiWAS '16. pp. 422–429. ACM Press, New York, New York, USA (2016)
9. Carothers, C.D., Meredith, J.S., Blanco, M.P., Vetter, J.S., Mubarak, M., LaPre, J., Moore, S.: Durango: Scalable Synthetic Workload Generation for Extreme-Scale Application Performance Modeling and Simulation. In: SIGSIM-PADS '17. pp. 97–108. ACM Press, New York, New York, USA (2017)
10. Chen, T.H., Syer, M.D., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M., Flora, P.: Analytics-Driven Load Testing: An Industrial Experience Report on Load Testing of Large-Scale Systems. In: ICSE-SEIP. IEEE (2017)
11. Conley, M., Vahdat, A., Porter, G.: Achieving cost-efficient, data-intensive computing in the cloud. In: SoCC'15. pp. 302–314. ACM (2015)
12. Curiel, M., Pont, A.: Workload generators for web-based systems: Characteristics, current status, and challenges. IEEE Communications Surveys Tutorials **20**(2), 1526–1546 (2018)
13. Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., Orth, P., Salman, K.S., Richenhagen, B., et al.: Smardt modeling for automotive software testing. Software: Practice and Experience **49**(2), 301–328 (2019)
14. Dulz, W.: A Versatile Tool Environment to Perform Model-based Testing of Web Applications and Multilingual Websites. In: ICSOFT. pp. 45–56. SciTePress (2018)
15. Ferme, V., Pautasso, C.: A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments. In: ICPE '18. pp. 261–272. ACM Press, New York, New York, USA (2018)
16. Furda, A., Fidge, C., Barros, A., Zimmermann, O.: Reengineering data-centric information systems for the cloud–a method and architectural patterns promoting multitenancy. In: Software Architecture for Big Data and the Cloud. Elsevier (2017)
17. Grechanik, M., Luo, Q., Poshyvanyk, D., Porter, A.: Enhancing Rules For Cloud Resource Provisioning Via Learned Software Performance Models. In: ICPE '16. ACM Press, New York, New York, USA (2016)
18. Henein, M.M.R., Shawky, D.M., Abd-El-Hafiz, S.K.: Clustering-based Undersampling for Software Defect Prediction. In: ICSOFT. SciTePress (2018)
19. Hooda, I., Chhillar, R.S.: Software test process, testing types and techniques. International Journal of Computer Applications **111**(13) (2015)
20. Huang, W.l., Peleska, J.: Safety-complete test suites. In: IFIP - ICTSS '17. pp. 145–161. Springer (2017)
21. Jiang, Z.M., Ming, Z.: Automated analysis of load testing results. In: ISSTA '10. p. 143. ACM Press, New York, New York, USA (2010)
22. Kaczmarski, M., Perry, P., Murphy, J., Portillo-Dominguez, A.O.: In-test adaptation of workload in enterprise application performance testing. In: ICPE '17 (2017)
23. Krichen, M., Maâlej, A.J., Lahami, M.: A model-based approach to combine conformance and load tests: an ehealth case study. International Journal of Critical Computer-Based Systems **8**(3-4), 282–310 (2018)
24. Luo, Q., Poshyvanyk, D., Nair, A., Grechanik, M.: FOREPOST: a tool for detecting performance problems with feedback-driven learning software testing. In: 38th ICSE-C. pp. 593–596. ACM (2016)
25. Maâlej, A.J., Krichen, M.: A model based approach to combine load and functional tests for service oriented architectures. In: VECoS. pp. 123–140 (2016)

26. Markthaler, M., Kriebel, S., Salman, K.S., Greifenberg, T., Hillemacher, S., Rumpe, B., Schulze, C., Wortmann, A., Orth, P., Richenhagen, J.: Improving model-based testing in automotive software engineering. In: ICSE-SEIP. pp. 172–180. SciTePress (2018)
27. Meier, J.D., Farre, C., Bansode, P., Barber, S., Rea, D.: Performance testing guidance for web applications: Patterns & Practices. Microsoft (2007)
28. Portillo-Dominguez, A.O., Ayala-Rivera, V.: Improving the testing of clustered systems through the effective usage of java benchmarks. In: CONISOFT (2017)
29. Portillo-Dominguez, A.O., Perry, P., Magoni, D., Murphy, J.: PHOEBE: an automation framework for the effective usage of diagnosis tools in the performance testing of clustered systems. Software: Practice and Experience (2017)
30. Portillo-Dominguez, A.O., Wang, M., Murphy, J., Magoni, D.: Automated WAIT for cloud-based application testing. ICSTW (2014)
31. Portillo-Domínguez, A.O., Murphy, J., O'Sullivan, P.: Leverage of extended information to enhance the performance of JEE systems. IT&T 2012 (2012)
32. Ramakrishnan, R., Shrawan, V., Singh, P.: Setting realistic think times in performance testing: A practitioner's approach. In: ISEC'17. pp. 157–164. ACM (2017)
33. Reichelt, D.G., Kühne, S.: Better Early Than Never. In: ICPE '18. ACM Press, New York, New York, USA (2018)
34. Reichelt, D.G., Kühne, S.: How to Detect Performance Changes in Software History. In: ICPE '18. ACM Press, New York, New York, USA (2018)
35. Sánchez, A.B., Delgado-Pérez, P., Segura, S., Medina-Bulo, I.: Performance mutation testing: Hypothesis and open questions. Information and Software Technology **103**, 159–161 (2018)
36. Segura, S., Troya, J., Duran, A., Ruiz-Cortes, A.: Performance Metamorphic Testing: Motivation and Challenges. In: ICSE-NIER (2017)
37. Shams, M., Krishnamurthy, D., Far, B.: A model-based approach for testing the performance of web applications. In: SOQUA '06. p. 54. ACM Press, New York, New York, USA (2006)
38. Spear, W., Shende, S., Malony, A., Portillo, R., Teller, P.J., Cronk, D., Moore, S., Terpstra, D.: Making Performance Analysis and Tuning Part of the Software Development Cycle. DoD High Performance Computing Modernization Program Users Group Conference (2009)
39. Tang, W., Fu, Y., Cherkasova, L., Vahdat, A.: Medisyn: A synthetic streaming media service workload generator. In: NOSSDAV '03. p. 12. ACM Press, New York, New York, USA (2003)
40. Troya, J., Segura, S., Ruiz-Cortés, A.: Automated inference of likely metamorphic relations for model transformations. Journal of Systems and Software **136** (2018)
41. Wu, H., Tantawi, A.N., Yu, T.: A Self-Optimizing Workload Management Solution for Cloud Applications. In: ICWS 2013. pp. 483–490. IEEE (2013)