DYNAMOJM: A JMeter Tool for Performance Testing Using Dynamic Workload Adaptation

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

DYNAMOJM: A JMeter Tool for Performance Testing using 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 assure optimal experience for users, especially when there are high loads of concurrent users. JMeter is one of the most widely used tools for load and stress testing. With JMeter, it is possible to test the performance of static and dynamic resources on the web. This paper presents DYNAMOJM, a novel tool built on top of JMeter that enables testers to create a dynamic workload for performance testing. This tool implements the DYNAMO approach, which has proven useful to find performance issues more efficiently than static testing techniques.

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

1 Introduction

The importance of providing stable and reliable sites for users has increased with the consumption of services throughout the internet. The performance of a service is still a major concern due to its crucial impact on applications [14]. Performance failures are not acceptable with the rising competition in the market [8]. The goal of performance testing is to assess how well a service can work under certain workloads [10]. Nevertheless, performance testing is challenging due to the multiple variables involved in enterprise-level services [16]. Nowadays, software performance continues to be tested using tools based on static workloads such as JMeter [1], HP LoadRunner [3] and RTP from IBM [4]. An inconvenience with this approach is that the tester requires to have prior knowledge of the applications to define an appropriate test workload. If this does not occur, there is a risk that the application under test (AUT) does not exhibit some performance issues and bugs are not found. DYNAMO [7] is an approach that can automatically find an adequate workload for testing without a try and error process, hence, saving time without compromising bug detection. To generate the workload, DYNAMO relies on the analysis of performance samples to dynamically adjust the workload to the maximum allowed without reaching a saturation point in the system. However, DYNAMO is currently built as a basic research prototype that was developed to demonstrate a new testing approach.

Table 1. Configuration Parameters

Phase	Parameter	Example
Initial Settings	Test duration	200 min
	Phase 1 and Phase 2 ratio	40/60%
Phase 1 Settings	Calibration workloads (WK1, WK2)	[2,20]
	Number of transactions considered as WKS (%WKS)	30%
Phase 2 Settings	Sample interval (SI)	5 min
	Error rate threshold (ERT)	90%
	Adjustment strategy (ADS)	Min
	Workload increment (WKINC)	5 users
	WKS transactions to be increased (%WKINC)	50%

Therefore, it does not provide a graphical interface to facilitate its usage and configuration. Moreover, it has a lot of parameters that need manual configuration using the command line which hinders its usability. To tackle this, in this paper we present DYNAMOJM, a tool built on top of JMeter that incorporates all DYNAMO logic into a more usable plugin for practitioners.

The DYNAMO approach involves two main phases [11]. The objective of Phase 1 (Ph1) is to identify the workload sensitive transactions(WKS) involved in the test, that is, those transactions that are more susceptible to suffer performance degradation. Whereas Phase 2's (Ph2) goal is to exercise the WKS as much as possible while avoiding the saturation of the system. DYNAMO's configuration requirements and some example values are illustrated in Table 1. The calibration workloads in Ph1 are used to calculate performance differences (deltas) between the runs. The adjustment strategies are used to identify the transactions that will be increased. DYNAMO supports 3 strategies: *Min*, which increases the best performance WKS transactions; *Max*, which increases the worst performance WKS transactions; and *Random*, which randomly selects a set of WKS transactions to be adjusted.

2 Background and Related Work

In spite of the increasing access to more and better computational resources (e.g., CPU, RAM), system performance remains a major concern due to its crucial impact on applications. Currently, software performance is typically tested using tools based on static workloads (e.g., Apache JMeter [1] and IBM RPT [4]). This is usually complemented with a diagnosis tool (e.g., WAIT [5] and New Relic [6]) to monitor and analyze in real-time the performance of applications.

Some approaches to enhance performance testing include the use of static code analysis to discover performance errors. The authors of [18] determine a test plan based on code commits and the use of artificial unit tests to compare the performance between different versions. Meanwhile, the work presented

on [17] describes the performance analysis of software systems as a comparison of two versions of software and their performance results to find possible (regression) bugs. Due to the complexity of running performance testing, another used technique is to analyze key performance indicators on the fly. The work of [13] proposes the use of a Timed Automata to monitor and analyze the request invocations and responses between the components of web services. This tool gathers all the information and creates a log with it. After that, the data is examined to create a report that includes the possible performance faults of the services.

The work of [12] incorporates the use of machine learning to analyze the outputs and feedback of the AUT to find bottlenecks in the system. Other methods examine the elasticity of cloud applications [9] to determine how many resources are needed to run the service without affecting the performance. Finally, other authors propose the use of metamorphic testing and mutation testing to improve the results of performance testing. The work of [20] involves the use of metamorphic testing to reduce the complexity of the system configuration by creating test cases based on the inputs and outputs of the program. Another important variable is the optimization of garbage collection [15]. On the other hand, mutation testing is used to improve the effectiveness of performance testing plans. The work presented in [19] describes how using mutation testing in combination with performance testing can lead to a fault detection enhancement.

3 DYNAMOJM's Implementation and Example

The architecture of DYNAMOJM consists of 4 main components: *Controller*, *DecisionMaker*, and the implementation of two interfaces: *WorkloadTool* and *LogAnalizer* (as shown in Fig. 1). The Controller is the main component in charge of managing the required inputs and control the flow of the plugin calling the logic for the different phases. The DecisionMaker contains DYNAMO's logic and functions to analyze the information provided by the implementation of the LogAnalyzer and functions to adjust the workload using the implementation of the WorkloadTool. The interfaces were designed to facilitate the implementation of DYNAMO using other workload tools, as depicted in Fig. 1.

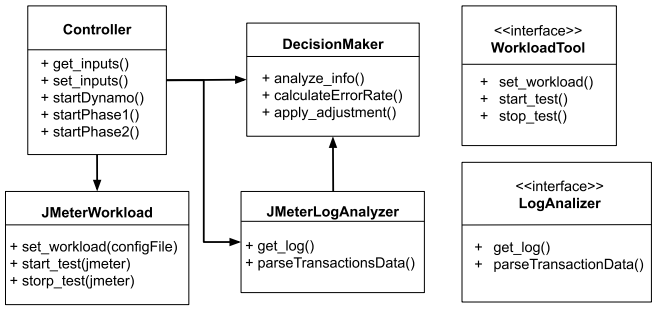

Fig. 1. DYNAMOJM Interfaces

To evaluate the performance of a web application, performance testers (hereinafter users) need to define an estimated workload to be tested in the AUT. The problem with this approach is that users need to have some experience or previous knowledge in the AUT to calculate an adequate workload. The other option is to estimate an ideal workload based on a try and error approach with the risk of expending more time and to overlook problems. With DYNAMOJM, the users prepare their test plan as normal using JMeter and now they have available a configuration panel to help them to identify the best workload with minimum knowledge of the AUT. For our evaluation study, we tested DYNAMOJM using

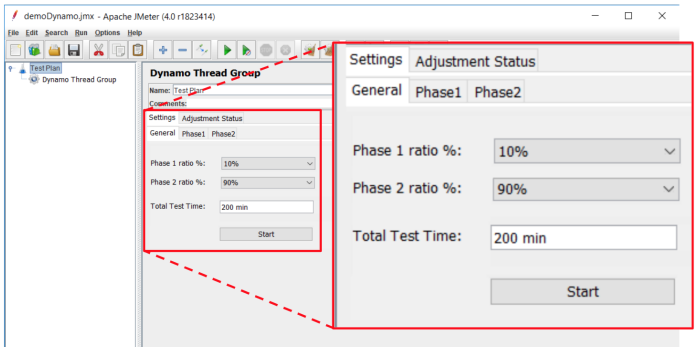

Fig. 2. General Configuration

an experimental web app built around DaCapo [2] which is a well-known Java benchmark used in the literature. DaCapo is wrapped with java servlets to emulate the transactions of a web application. DaCapo consists of 13 benchmarks who are called individually as web transactions.

The main panel of DYNAMOJM is comprised of two tabs: *Settings* and *Adjustment Status* (as shown in Fig. 2). In *Settings*, there are three tabs for the setup of DYNAMOJM (*General*, *Phase1*, and *Phase2*). Users start by accessing the *General* tab and entering the values of the test run (we have used some example values to illustrate our approach): 10% for the Ph1 and 90% for Ph2, considering that Ph1 plus Ph2 should be equal to 100%, we also configure the total test time. As a guideline for practitioners, to configure *General* Settings on DYNAMOJM, we recommend using values between 10% and 40% for Ph1 ratio because reducing the time of Ph2 will lead to the detection of fewer bugs. For the Total Test Time, it depends on the AUT and its requirements. In Fig. 3, the settings of Ph1 are defined: workload 1 of 500, workload 2 of 4000, and using 30% as WKS. Workload 1 indicates the initial load for the AUT; while Workload 2 is considered a high load which the AUT is still able to handle properly. The aim of the calibration workloads is to retrieve the first samples of the AUT behavior. The WKS% is the number of transactions that will be labeled as WKS, so a higher % will mark almost all the sensitive transactions, while a lower value will

mark only a few transactions. The recommendation is to use a value close to the 50% or based on the complexity of the tested transactions. In Fig. 4, we configure the settings for Ph2. Here, the user can configure the type of adjustment choosing from 3 strategies: *min, max* and *random*; in this example it is set to *min* although *max* is another viable option while *random* is not recommended as this strategy was used for research purposes. The number of users that the transactions will be increased by is set to 200. This increment is based on the values of the tests runs of Ph1, because Ph2 will start adjusting the transactions using the second value of the Ph1 as a starting point. The WKS to modify is set to 50% because modifying a high number of transactions at the same time could rapidly saturate the system, this value also depends on the capacity of the system. Finally, the saturation point is set as 90%. The Saturation point is the threshold defined to make the adjustments while the sample interval configured to 5 min, is the time to collect metrics for the analysis and adjustments during Ph2.

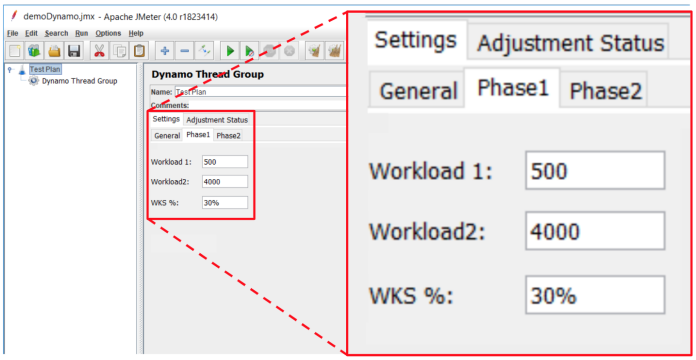

Fig. 3. Phase1 Configuration

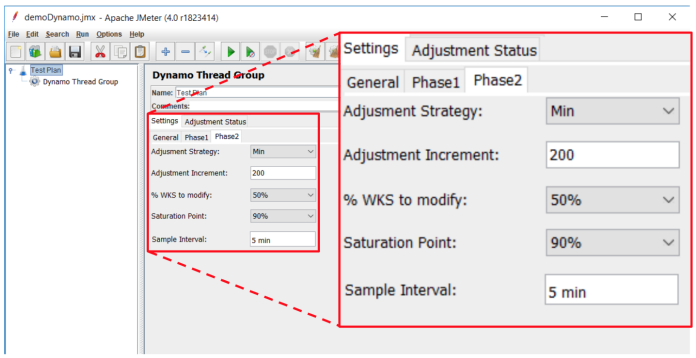

Fig. 4. Phase 2 Configuration

Once the configuration is completed, the user can move to the *General* tab and start the test run. When the run is started, the user can monitor the adjust-

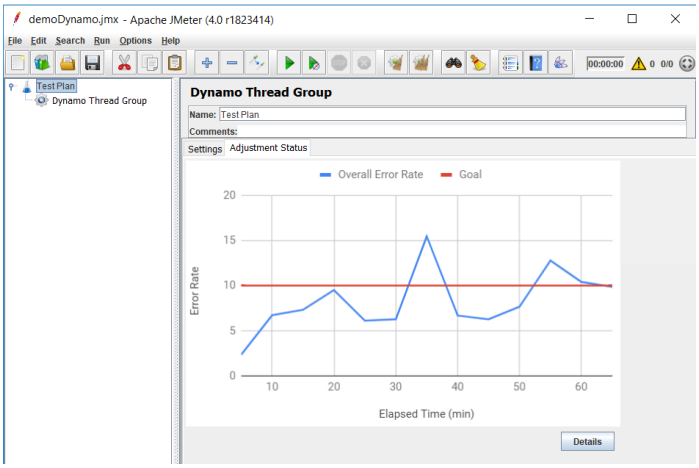

Fig. 5. Workload Adjustment

ments of the workload through the *Adjustment Status* tab. For example, Fig. 5 demonstrates how the workload is adjusted during the test run until reaching the maximum load under the defined parameters. Moreover, when using the *Details* button at the bottom of the window, the user can visualize which transactions are affected during each adjustment.

An example of the type of results that can be achieved with DYNAMOJM are depicted in Figs. 5 and 6: Fig. 5 shows the behavior of the saturation point (i.e., the error rate threshold -goal- which is represented by a red line) exhibited by an AUT during a 60-min test run. It can be noticed how the gradual increases in test workload have an impact in the overall error rate (represented by the blue line). In the early stages of the test, the error rate is very close to 0% because the test workload is low. Nevertheless, during the test run the error rate might become relevant (as a consequence of the increments in workload) and eventually exceed the configured threshold (peaks exhibited in the figure around minutes 30-40 and 50-60 of the test duration). If this occurs, the workload gets automatically decreased in order to keep the error rate under control. Fig. 6 shows the *Details* page that presents the actual workload adjustments made during the test (per functional transaction tested). This is visually depicted by the segments of some transactions (within the stacked bar chart that represent 5-minute sections of the test run) which gradually become considerable wider that others which remain very narrow. This is the result of DYNAMO gradually increasing the test workload of the most sensitive transactions (i.e., the wider segments) by following the selected adjustment strategy (e.g., *min* in our example) until the appropriate combination of workloads (i.e., the one closest to the saturation point) is found. On the contrary, the least workload-sensitive transactions never get stressed. As a result, their test workload remains low during the whole test run. This is visually illustrated by the most narrow segments in the figure.

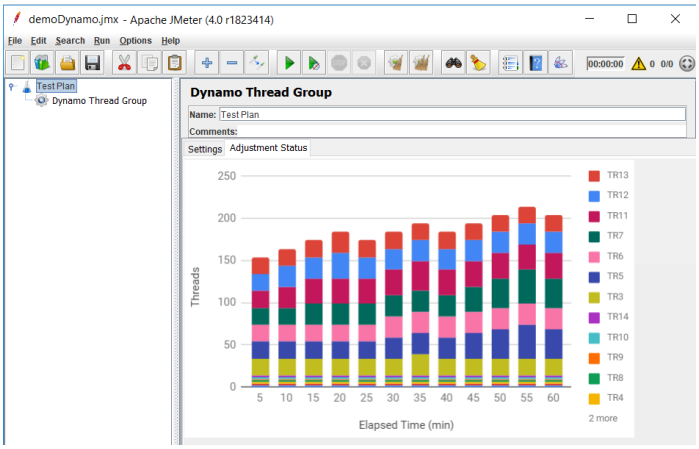

Fig. 6. Transaction Details

4 Conclusions

Designing the appropriate workload for performance is challenging because it requires experienced testers to define a workload, as well as a series of trial-and-error test runs, to find the appropriate load to stress a particular system. By using DYNAMO, it is possible to find such workload automatically. However, DYNAMO itself requires several configuration parameters which have been implemented in this tool. The accuracy of DYNAMO to find bugs or potential issues has been proved using a proof of concept demo [7]. However, with the DYNAMOJM plugin presented in this paper, it is possible to use the same logic and automation with a friendly graphical interface easy to adopt by performance engineering practitioners.

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. Apache JMeter. http://jmeter.apache.org/
2. Dacapo benchmark suite. http://dacapobench.org/
3. HP Loadrunner. https://ssl.www8.hp.com/sg/en/ad/load-runner/load-runner.html
4. Ibm rational performance tester. https://www.ibm.com/us-en/marketplace/ibm-rational-performance-tester

5. IBM WAIT Tool. https://publib.boulder.ibm.com/httpserv/cookbook/Major_Tools-IBM_WholeSystem_Analysis_Tool_WAIT.html
6. New relic. https://newrelic.com
7. Ayala-Rivera, V., Kaczmarski, M., Murphy, J., Darisa, A., Portillo-Dominguez, A.O.: One Size Does Not Fit All. In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18. pp. 211–222. ACM Press, New York, New York, USA (2018)
8. Conley, M., Vahdat, A., Porter, G.: Achieving cost-efficient, data-intensive computing in the cloud. In: SoCC'15. pp. 302–314. ACM (2015)
9. Grechanik, M., Luo, Q., Poshyvanyk, D., Porter, A.: Enhancing Rules For Cloud Resource Provisioning Via Learned Software Performance Models. In: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering - ICPE '16. pp. 209–214. ACM Press, New York, New York, USA (2016)
10. Jiang, Z.M., Ming, Z.: Automated analysis of load testing results. In: Proceedings of the 19th international symposium on Software testing and analysis - ISSTA '10. p. 143. ACM Press, New York, New York, USA (2010)
11. Kaczmarski, M., Perry, P., Murphy, J., Portillo-Dominguez, A.O.: In-Test Adaptation of Workload in Enterprise Application Performance Testing. In: 8th ACM/SPEC on ICPE '17 Companion. pp. 69–72. ACM Press (2017)
12. Luo, Q., Poshyvanyk, D., Nair, A., Grechanik, M.: Forepost: a tool for detecting performance problems with feedback-driven learning software testing. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 593–596. ACM (2016)
13. Maâlej, A.J., Hamza, M., Krichen, M.: Wsclt: a tool for ws-bpel compositions load testing. In: 2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. pp. 272–277. IEEE (2013)
14. 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)
15. Portillo-Dominguez, A.O., Wang, M., Murphy, J., Magoni, D.: Automated WAIT for cloud-based application testing. ICSTW (2014)
16. Portillo-Domínguez, A.O., Murphy, J., O'Sullivan, P.: Leverage of extended information to enhance the performance of JEE systems. The IT&T 11th International Conference on Information Technology and Telecommunication 2012 , Cork, Ireland, 29-30 October 2012 (2012)
17. Reichelt, D.G., Kühne, S.: Better Early Than Never. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18. pp. 127–130. ACM Press, New York, New York, USA (2018)
18. Reichelt, D.G., Kühne, S.: How to Detect Performance Changes in Software History. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18. pp. 183–188. ACM Press, New York, New York, USA (2018)
19. 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)
20. Segura, S., Troya, J., Duran, A., Ruiz-Cortes, A.: Performance Metamorphic Testing: Motivation and Challenges. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER) (2017)