Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems

2010

Volume 43

Self-Directed Performance Testing

Aleksandr Sukhorukov, *Riga Technical University*

Abstract **- Performance testing of multi-user software systems is typically performed by emulating activities of multiple users working with the system simultaneously. These virtual users collect performance measures like response time and correctness while loading the system during the test. During a typical load scenario number of simultaneous virtual users gradually increases from one to some intended maximum, allowing to find out statistical relation between number of simultaneous users and performance measures.**

However having fixed load scenario has several drawbacks. Usually the test has to be repeated iteratively with adjusted load scenarios, taking into account results acquired from previous test iterations. As load scenarios have to be adjusted manually and repetitions of the test may take considerable time, this approach seems to be ineffective.

This paper presents an alternative approach, where load scenario is adjusted automatically during the test, taking into account performance measures already gathered in the same test. Drawbacks of fixed load scenarios are analysed and methods of eliminating them by automatic adjustment are described. A case study showing implementation of the method and results obtained by applying it to performance testing of a real system is provided.

Keywords: **load generation, performance measurement, software performance testing**

I. Introduction

Software performance testing is an important activity for evaluating quality of multi-user systems. In order to test performance of a system we need to gather performance metrics in conditions of simultaneous work of multiple users. Thus we can get information about impacts of different load levels on individual user's experience.

Design of performance tests is radically different from functional tests [1]. Functional details of test cases like actual input data are not so important in performance testing. More important aspects are user scenarios and produced workload. Target results are different as well. Performance test doesn't focus too much on functional correctness of system operations, it is interested in response times, CPU and memory utilization, request queue lengths and other performance-related issues. Nowadays focus has shifted from performance metrics observable at system level to ones observable on end-user level [2]. End-user doesn't know and doesn't care much about page fault rate on database server or network throughput, what really matters for him or her is observable system response time and principal correctness of responses.

In order to measure these metrics performance test emulates work of multiple simultaneous users performing their usual job. Depending on test goals different workloads can be imitated by means of test load scenarios. Designing load scenario before test is a common approach nowadays. In this paper we present an alternative approach where load scenario is constructed automatically during the test to ensure faster achievement of test goal.

In Section II we describe a conceptual performance test model to introduce basic concepts and terms used throughout the paper. Section III looks more closely at classical approach of fixed pre-designed load scenarios and analyses their drawbacks. Section IV introduces the concept of self-adjusting load scenarios as a solution for problems of fixed load scenarios. A case study showing results of applying this approach in performance testing project is described in the Section V. Section IV concludes the discussion.

II. Performance Test Model

Actual structure of performance test depends on tools being used. In this section we present conceptual tool-independent performance test model to base following discussion upon. The model is illustrated in Fig. 1.

We will use term "load agent" to describe software which actually emulates users during the test as well as hardware machine where this software operates. The discussion will begin at the lowest level of the model.

Level 1 – Requests

At the lowest level test consists of requests. Performance tests usually emulate users at protocol level in order to be able to run virtual users with no use of client software. This approach allows using resources of load agent machine more efficiently and run much more virtual users than it would be possible if client software would be necessary to run for each user. So, requests are data sent to a system under test (SUT) from load agent machine via network and for which SUT should send back some response. In case of Web systems they would be GET or POST requests of HTTP protocol.

At requests level test can also contain additional steps like verifying correctness of the response according to specified rules, handling dynamic communication components like session identifiers etc.

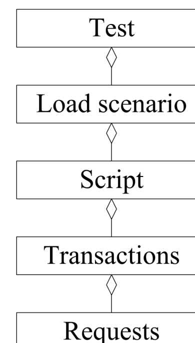

Fig. 1. Performance test model

Level 2 – Transactions

The next level is transactions. We define transaction as a single operation SUT performs which is undividable from end-user point of view.

For example, login transaction is what happens between the moment of clicking "login" button and the moment when the first authorized page of Web portal is displayed on user's browser. Although from server point of view transaction is just a series of requests it must handle, from end-user point of view it seems like a single piece of work.

Therefore durations of transactions are more important from user-centric performance test than durations of separate requests.

Level 3 – Scripts

Transactions are combined in scripts at the higher level. A script is a series of transactions run either in fixed or variable order.

The purpose of script is to emulate a sequence of actions real users can perform to make test realistic. A script usually specifies either fixed or variable delays (also known as think times) between transactions in order to realistically emulate user's activities of looking through displayed data, filling in forms or just moving and clicking a mouse.

Test can include multiple scripts each of them emulating different kinds of users. So, virtual user is a dynamically running instance of some particular script.

Level 4 – Load Scenarios

Load scenarios are prescriptions of how long the test will run and how number of virtual users will change in time. Depending on a test goal, load scenario could define long-run execution of fixed number of users, or gradual increase of number of users from zero to some maximum level. Variations of these two examples as well as more sophisticated load scenarios are also possible.

Test at the highest level can consist of several load scenarios each running on its own load agent machine in order to emulate heavy workloads for which resources of single load agent machine would be insufficient.

III. Fixed Load Scenarios

A classical approach to workload creation in performance tests is to design load scenarios prior to test run. Careful design of both scripts and load scenarios is needed in order to meet test goals [3]. In this approach load scenario specification serves as an input to load agent component, which we will refer to as scheduler, responsible for ensuring workload corresponding to the load scenario. Performance counters are measured then and form the output of the test. This process is illustrated in Fig. 2.

In many cases having fixed pre-designed load scenario is sufficient and is even the best method. For example, if we need to evaluate different performance metrics under different load levels, we would design load scenario where number of virtual users gradually increases from zero to required

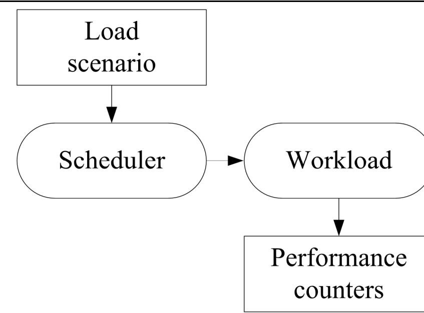

Fig. 2. Test with fixed load scenario

capacity of SUT, execute the test and process the results in straightforward way to acquire captured performance counters as functions of number of virtual users [4]. Having such functions and representing them graphically can be very helpful to determine bottlenecks of SUT as well as to evaluate whether actual SUT capacity corresponds to the required one.

The drawback of this approach is relatively long time needed for test execution. If the goal is narrower then mentioned example, it can be inefficient. Time needed for test execution is one of the most important factors that affect how often the test would be re-run during development. In some software development methodologies, frequent re-running tests is necessary and performance tests could also be included to the test set which should be executed after each little change introduced into code, like in test-driven development [5]. Long execution times may prevent performance tests from inclusion in this test set, so they would be run rarely and detection of performance problems may occur late after their introduction. For this reason fixing lately discovered performance problems may become more expensive.

A wide class of performance test goals for which fixed load scenarios are ineffective is formed by necessity to find out load level at which some state S is reached in SUT. Some examples of such goals may be:

- At what load level at least one transaction's average response time becomes longer than 10 seconds?
- At what load level average utilization of application server's CPU becomes higher than 80%?
- At what load level maximum capacity of SUT is reached in terms of transactions served per second?
- At what load level SUT starts to produce errors?

These goals are obviously different from the goal of discovering performance counters as a function of number of virtual users (Formula 1):

$$p = F(v), \tag{1}$$

where

p – performance counter value;
v – number of virtual users;
F – function to be discovered.

If the goal is discovery of such function, we have to find out p values for all arguments in some interval $[0, v_{max}]$. Oppose to

this goal our example questions require to find out v_0 for a given p_0, such that $F(v_0) \geq p_0$ and $F(v_0 - 1) < p_0$.

To achieve goals of this type with fixed load scenarios we need to discover the function $F(v)$ for load levels in interval $[v_{min}, v_{max}]$ which we expect our target v_0 to belong to. The narrower is the interval the less time test will take, but the risk of selecting wrong interval will also be higher.

Other approach which also is more useful if we do not have prior expectations about approximate interval is to run several tests with load scenarios of different granularities of load increase. The first scenario, for example, could increase load by 100 virtual users at once, so we can get an interval of length $v_{max} - v_{min} = 100$ where v_0 is expected to be found. Then we run the test with smaller load increase granularity to get the narrower interval and repeat these iterations until we find out v_0 value. This approach however requires manual work of reconfiguring load scenarios and still is not time-efficient.

In the next section we present our approach where load scenario is adjusted automatically by scheduler in order to achieve faster v_0 point detection.

IV. Self-Adjusting Load Scenarios

There are known approaches to dynamically generate varying workload content, for example, automatic generation of transaction sequences based on form-oriented models [6]. Here we present an approach for dynamically varying load levels during the test in order to achieve faster test goal achievement.

A. Self-Adjusting Load Scenario Principles

The approach is illustrated in Fig. 3. In this model scheduler is not just an executor of prescribed load scenario, but is an active decision maker of what to do next. Decisions are based on two sources:

1. Configuration of scheduler parameters. These parameters describe goals which need to be achieved during the test and which scheduler's algorithm is capable to process.
2. Performance counters measured up to the moment of decision making. Scheduler may analyze previously captured metrics and base its decisions of load change on them.

The major distinction of this model in comparison with model of fixed load scenario, is feedback of performance counters that scheduler receives.

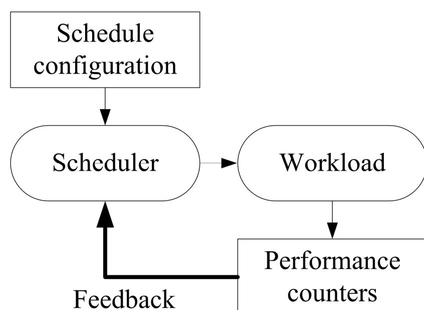

Fig. 3. Self-directed test model

B. Self-Directed Scheduler Algorithm

Now we will describe a simple multi-purpose self-directed scheduler algorithm which can be useful for achieving various performance testing goals. Without loss of generality we can assume function $F(v)$ is monotonically non-decreasing. The algorithm consists of two phases.

The first phase is aimed to find an interval which point v_0 belongs to. To make this happen quickly number of users grows exponentially in time by doubling number of users iteratively.

1. Start with 1 virtual user ($v = 1$) and measure $F(v)$;
2. While $F(v) < p_0$ double the number of virtual users ($v := 2v$) and re-measure $F(v)$;
3. When we find v value at which $F(v) \geq p_0$, we can say that target v_0 belongs to interval $[v/2, v]$.

The second phase is aimed to narrow the interval $[v_{low}, v_{high}]$ discovered:

1. If $v_{high} - v_{low} > 1$ take $v := v_{low} + (v_{high} - v_{low})/2$ and measure $F(v)$;
2. If $F(v) \geq p_0$, then $v_{high} := v$, otherwise $v_{low} := v$;
3. Repeat from step (1) until $v_{high} - v_{low} = 1$. At this point v_{high} value is the target v_0.

In other words in the second phase we use binary search algorithm to find v_0 in interval $[v_{low}, v_{high}]$. Total number n of evaluations of function $F(v)$ in the algorithm can be estimated with Formula 2:

$$n = 2\lceil \log_2 v_0 \rceil, \tag{2}$$

where

v_0 – target number of virtual users to be found.

As the number of $F(v)$ evaluations depends logarithmically on v_0, efficiency of the algorithm comparing with fixed load scenario approach is high. For example, if target load to be determined is about 1000 virtual users we need just 20 evaluations $F(v)$ or 20 steps of load changes.

In order for the algorithm to be effective some more aspects should be taken into account:

- After load increase the scheduler should wait until all new virtual users come into usual work rhythm and then measure average performance counters of interest for at least two times script execution time.
- After load decrease the scheduler should wait until all stopping virtual users correctly finish their tasks and the system "calm down" after load generated by those users, then the scheduler should measure average performance counters of interest for at least two times script execution time.

By script execution time we mean the time needed to execute all transactions of the script once plus sum of delays between transactions. These precautions are necessary in order better measure effect of load and to avoid noise in measured data which may be introduced due to effects caused by load changes.

It should be mentioned that the presented algorithm could fail to work in case there are other factors present that affect

performance counters more than load level being generated by load agent. In this case function $F(v)$ can be stochastically fluctuating and our assumption that it is monotonically non-decreasing would be wrong.

Increasing time the scheduler gathers performance counters for fixed load levels could help to make impact of those factors smaller, but it cannot eliminate them completely. Anyway it could be so, that measuring, for example, response times at load level of 30 virtual users, they could appear longer than for load level of 31 users.

Though probability of such noise is small for sufficiently long measuring periods, this situation is still possible and should be kept in mind.

V. CASE STUDY

In this section we present a case study illustrating usage of the proposed algorithm.

A. *Tests Used in the Experiment*

For a case study a Web-based defect tracking system Mantis deployment was used as a SUT. During the test a single simple script was used containing the following transactions:

1. Open the first page
2. Login
3. Open defect reporting form
4. Open active defects list
5. Logout

The goal of the test was specified as to find load level (number of simultaneously working virtual users) at which average duration of the longest transaction becomes more than three seconds.

Performance testing tool Picus developed by the author [7] was selected to prepare and execute the tests for the case study. Picus provides a convenient mechanism to implement alternative schedulers as plug-ins that was mandatory feature for the purpose of this experiment.

Two tests were prepared with different schedulers:

1. Built-in ramp-up scheduler. This scheduler works based on fixed load scenario, in this case starting increasing number of virtual users by one each minute. Maximum number of virtual users to achieve was specified as 100, so the test should take 100 minutes to complete.
2. Newly developed scheduler based on the algorithm described in the previous section. This scheduler evaluates function $p = F(v)$, which in this case is average duration of the longest transaction under load v and targets finding v_0 for $p_0 = 3$ seconds. Duration of single step of load increase or decrease was configured to be one minute.

B. *The Results*

Results gained in two tests were quite similar. Test 1 detected $v_0 = 53$ virtual users, for test 2, the result was $v_0 = 51$ virtual user. These two results may be considered very close, as because of stochastic nature of performance testing

TABLE 1

TEST RESULTS

	Test 1	Test 2
Target load level discovered	53	51
Time of execution (minutes)	100	12
Time until target discovery	53	12

difference between even two runs of the same tests could be much higher.

The difference between those two tests is time of their execution. Test 1 was run as scheduled for 100 minutes. Test 2 was run 12 minutes, as it could be estimated by formula 2 and taking into account that load changes occur once per minute. These results are summarized in Table 1.

To illustrate dynamic aspects of executed tests, we provide two charts that show number of concurrent users and response times over time for both tests. Time scale at the figures is cropped to 53 minutes, i.e. to the point where both tests have discovered the target v_0 value. Changes of average duration (response time) of the longest transaction over time are shown in Fig. 4.

It can be observed that test 2 quickly achieved load levels where response times are much higher than test 1 ever gained. The peak of more than 8000 milliseconds is where the first phase of algorithm finished. After it fluctuating part of chart follows where the second part of algorithm was executing, performing binary search of target v_0 value.

Changes of load level over time are shown in Fig. 5. At this chart benefits of exponential growth of load during test 2 versus linear load growth during test 1 are well observable.

Looking at these charts it can also be observed that test 2 gives us just so much information as it is necessary for test goal. On the other hand test 1 provides more information relevant for deeper performance analysis. A line corresponding to test 1 at Fig. 4. by its form is equivalent to graph of the function $F(v)$ (and, thanks to one-minute load growth interval, it is also equivalent by scale). From this graph

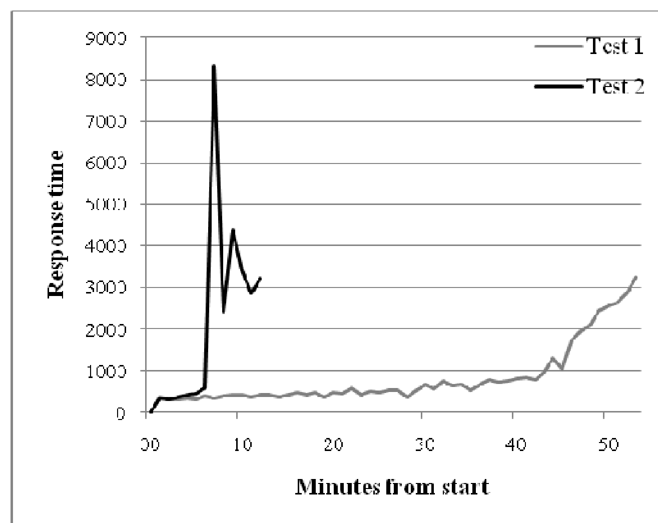

Fig. 4. Response times over time

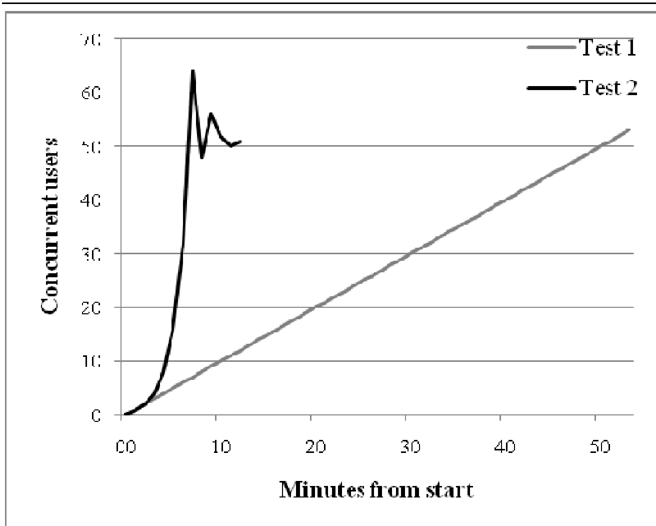

Fig. 5. Load level over time

it can be observed that at load levels of approximately 45 virtual users response times start to increase quicker, what can be an indication that some system resource was exhausted at this level. We could not get such valuable (but irrelevant for the test goal) information from chart of test 2.

VI. Conclusions

We have shown that self-directed performance testing performed by application of smart scheduler can provide faster results for certain types of performance test goals. In the discussed case study time needed to execute self-directed test was 8.3 times shorter than for similar test directed by fixed load scenario.

The proposed algorithm of self-directed scheduler is based on binary search algorithm and works well if test goal is described in terms of performance counter as a function of number of virtual users. For most performance counters the shape of the dependency function is clearly predictable. The algorithm is described for monotonically non-decreasing functions. For -increasing functions or for goals of finding optimum of bell-shaped functions the algorithm would require little adaptations but follows the same principle.

Self-directed performance test scheduling seems to be a topical and promising research direction. Having a high potential of increasing performance testing efficiency, this approach is not discovered enough. In this paper we presented just one possible algorithm of self-directed scheduler, however there are more directions to think about.

Constructing a scheduler algorithm that would be less sensitive against stochastic fluctuations of performance counters without significant loss of efficiency would be one of them.

Other interesting direction of research would be self-directed scheduler of multi-script test, where not just load levels, but also proportion of virtual user types would be automatically adjustable. In this case function F would have multiple arguments or single argument v of function $F(v)$ would be a vector. In general the target solution in multi-script test would be a set of vectors instead of single value, so more sophisticated algorithms would be required to discover it.

References

[1] G. Denaro, A. Polini, and W. Emmerich, “Early performance testing of distributed software applications,” in *4th International Workshop on Software and Performance (WOSP '04)*, January 2004. USA: ACM, 2004, pp. 94-103.

[2] D. P. Olshefski, J. Nieh, and D. Agrawal, “Inferring client response time at the web server,” in *International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS 2002)*, June 2002. USA: ACM, 2002, pp. 160-171.

[3] B. M. Subraya, *Integrated Approach to Web Performance Testing: A Practitioner’s Guide*. USA: IRM Press, 2006.

[4] C. Amza, et al., “Specification and implementation of dynamic web site benchmarks,” in *5th IEEE Workshop on Workload Characterization (WWC-5)*, November 2002. USA: IEEE Press, 2002, pp 3-13.

[5] M. J. Johnson, et al., “Incorporating performance testing in test-driven development,” *IEEE Software*, vol. 24, no. 3. USA: IEEE Computer Society, 2007, pp. 67-73.

[6] D. Draheim, et al, “Realistic load testing of web applications,” in *10th European Conference on Software Maintenance and Reengineering (CSMR 2006)*, March 2006. USA: IEEE Computer Society, 2006, pp. 57-70.

[7] A. Sukhorukov, “Performance testing tool Picus,” in *22nd International Conference on Systems for Automation of Engineering and Research (SAER-2008)*, September 2008. Bulgaria: King, 2008, pp. 165-172.

Aleksandr Sukhorukov acquired Master degree in Computer science at University of Latvia in 2004. Currently he is a doctoral student at Riga Technical University.

Since 2002 he has been working in software testing field, particularly in automation of functional and performance testing. Having software developer’s background he is author of several software automation solutions. Currently he works as an Assistant at Software Engineering Department in Riga Technical University, Meza 1/4 – 528, Riga, Latvia.

Aleksandrs Suhorukovs. Pašvadāma veiktspējas testēšana
Vairāklietotāju sistēmu veiktspējas testēšana parasti tiek veikta, imitējot vairāku lietotāju darbu, kuri vienlaicīgi strādā ar sistēmu. Šie virtuālie lietotāji uzkrāj veiktspējas mērījums, tādus kā atbildes laiki un korektums, noslogojot sistēmu testa laikā. Tipiskajā slodzes scenārijā vienlaicīgi strādājošo virtuālo lietotāju skaits pakāpeniski palielinās no viena līdz noteiktam maksimālajam, ļaujot izsekot statistisku atkarību starp vienlaicīgi strādājošo lietotāju skaitu un veiktspējas rādītājiem. Taču fiksēta slodzes scenārija izmantošanai ir daži trūkumi. Parasti tests ir iteratīvi jāatkārto, mainot slodzes scenārijus, ņemot vērā iepriekšējās testa iterācijās gūtus rezultātus. Tā kā slodzes scenāriji ir manuāli jāmaina, bet testa atkārtojumi var aizņemt ievērojamu laiku, šī pieeja izrādās neefektīva.Šajā rakstā tiek piedāvāta alternatīva pieeja, kurā slodzes scenārijs tiek pieskaņots automātiski testa laikā, ņemot vērā veiktspējas rādītājus, kas jau tika iegūti tajā pašā testā. Tiek analizēti fiksēto slodzes scenāriju trūkumi un tiek piedāvātas to novēršanas metodes, izmantojot automātisko slodzes scenārija pieskaņošanu. Aprakstīts pielietojuma piemērs, kas parāda metodes realizāciju un rezultātus, kas tika iegūti testējot reālās sistēmas veiktspēju.

Александр Сухоруков. Самоуправляемое тестирование производительности
Тестирование производительности многопользовательских систем обычно проводится, имитируя действия множества пользователей, одновременно работающих в системе. Такие виртуальные пользователи накапливают показатели производительности, например, время отклика и корректность, нагружая систему во время теста. Во время типичного сценария нагрузки число одновременных виртуальных пользователей постепенно возрастает от одного до определенного максимума, позволяя найти статистическое соотношение между числом пользователей и показателями производительности. Однако использование фиксированных сценариев нагрузки имеет некоторые недостатки. Обычно приходится итеративно повторять тест, изменяя сценарии нагрузки с учетом результатов, полученных в предыдущих итерациях. Поскольку сценарии нагрузки изменяются вручную, а повторы тестов

могут занимать значительное время, такой подход оказывается неэффективным. В статье предлагается альтернативный подход, при котором сценарий нагрузки автоматически изменяется во время теста, учитывая показатели производительности, уже полученные в этом же тесте. Анализируются недостатки фиксированных сценариев нагрузки, и описываются методы их устранения посредством автоматических изменений сценариев. Показывается пример реализации и применения метода, описываются результаты, полученные в ходе тестирования производительности реальной системы.