Proceedings of the International Conference on Information Technologies (InfoTech-2008)
19th – 20th September 2008, Bulgaria
vol. 2

PERFORMANCE TESTING TOOL PICUS

Aleksandr Sukhorukov
Riga Technical University
e-mail: Aleksandrs.Suhorukovs@rtu.lv
Latvia

Abstract: Performance testing tool Picus is designed to be an extensible platform for various kinds of performance testing for different environments. The article describes reasons of its creation, briefly overviews its architecture emphasizing its extension capabilities. Experience got in real projects where the tool was used is summarized and prospective development directions are outlined.
Key words: Performance testing, Test automation, Testing tools.

1. INTRODUCTION

Performance testing is an important activity of evaluating and improving software quality. In order to test system's performance from user perspective, we need to gather performance metrics in conditions of simultaneous work of multiple users. Thus we can get information whether and how user's experience is affected by other users working with the system at the same time.

Performance testing is connected closely with reliability and scalability and proper performance testing is able to serve for improvement of these and other quality aspects as well (Ash, 2003). Different kinds of performance testing exist, for example capacity, load and stress testing overviewed by Stottlemyer (Stottlemyer, 2001). Those kinds can vary in purpose and methods, however problem of establishing load conditions in terms of number of simultaneous users is relevant to all those kinds. It is not uncommon for systems to have a requirement of ability to serve thousands of users simultaneously. Executing such test is not imaginable without proper test automation and tool usage that provide means of such automation.

Although there are methods that help to evaluate performance even before the system is actually developed (Xia *et al*., 2006), the most objective final measures of performance can still be acquired only for existing system put into proper environment. There are many factors that impact performance, such as network, server and

client environments (Subraya, 2006), and their influence cannot be fully predicted before the system is deployed and tested.

This article is organized as follows. Section 2 justifies development of Picus performance testing tool; section 3 briefly discusses tool architecture and describes its extension possibilities; section 4 summarizes experience acquired in real projects where Picus was used. Section 5 concludes the paper and outlines prospective development directions that would further improve the tool.

2. NECESSITY FOR A NEW TOOL

There are many performance testing tools available. Extensive lists of tools can be found on web (Hower, 2008). They range from simplistic open-source solutions (Opensourcetesting.org, 2008) like JMeter or OpenSTA to feature-rich commercial products like HP LoadRunner or Compuware QALoad. To select appropriate tool two aspects should be considered:

1) What are features of available performance testing tools, their strength and weaknesses as well as types of systems and types of testing they can handle.
2) What are characteristics of systems that are going to be tested with a tool.

The second aspect is not less important. For example, if we need to test web applications, then we shouldn't select a tool that supports just testing of database of specific type. Even for web applications there are many factors that may affect tool appropriateness: web server type, authentication mechanism, resource types, HTTP methods used, client-side technologies like AJAX, encryption or compression of traffic etc. All these factors may be or may not be supported well by performance testing tool even if it is designed specially for web. And even supported features may be more or may be less easy to use when test is prepared and executed.

Appropriate and handy tool selection is not an easy task, but still it is worth doing it if we know exactly characteristics of systems that are going to be tested. For companies that provide performance testing services the problem is that they do not know exactly what characteristics will the next customer's systems have. Three strategies are possible:

1) Use just one, sufficiently universal tool that will suit most of projects and do not take up projects where this tool will not be appropriate. With this strategy otherwise profitable projects will be skipped and there is a risk that some obstructive technical factors will be discovered in the middle of the project.
2) Select a tool for each project separately. This strategy requires additional effort for tool selection and also may require additional expenses on license acquisition. It is also more difficult for service provider to be an expert in many different tools.

3) Select a tool that may be not so universal at start, but provides means for extending itself. This may require additional effort for tool extension for some projects, however extension once developed can be reused in future projects, therefore need for extension is likely to decrease in time.

The tool named Picus was developed as a result of author's five year experience in providing performance testing services with many different tools both open-source and commercial and recognizing their strengths and weaknesses. Some existing open-source tools provide means for extension but they did not seem to be sufficient enough. So, the third strategy was selected.

3. TOOL ARCHITECTURE

In order to be able to execute heavy load on system two layers were introduced: console and load agent. Through console tests can be configured, started and controlled. Load agents exercise system under test and gather performance statistics such as response times and correctness. Load agents can reside on different machines and are orchestrated by console. This structure is illustrated at Figure 1. Both console and load agents are implemented in Java.

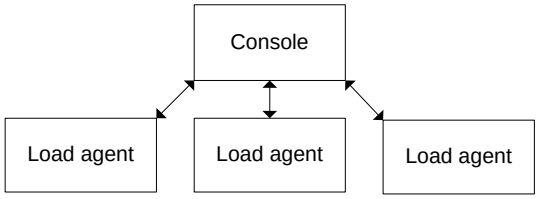

Fig. 1. Console and load agents.

Console communicates with load agents over TCP, issuing commands and receiving statistics. It is also possible to run one load agent in console process, in which case they communicate directly through method calls.

In order to test a specific system, test must be defined. Test in Picus is defined in four layers:

1) Script. In Picus script is Java class that provides a sequence of steps that do some actions with the system under test or verify correctness of received responses. Script is meant to be an emulation of real user performing specific actions with the system.
2) Schedule. Defines how instances of one specific script should be executed: how many script instances to execute (number of users to emulate), how this number changes in time (whether increases or decreases, or increases and then decreases etc), for how long these instances should run and other aspects.
3) Assignment. Defines schedules that one specific load agent should execute.
4) Test. Defines how assignments are distributed between available load agents.

While script is a Java class, other test information (schedules, assignments and test itself) is stored in XML file which can be manipulated through console user interface.

During test execution each layer gathers its own statistics and sends them to the corresponding statistics manager component. There is one separate statistics manager for each test layer. Lower layer component sends statistics to its upper level component, which aggregates them, combines with its own statistics and then sends to its upper level statistics manager. This data flow is illustrated at Figure 2. Each level statistics component can be controlled from console: level of detail that should be sent to upper level, whether to write those statistics in log file, whether to show them on console user interface etc.

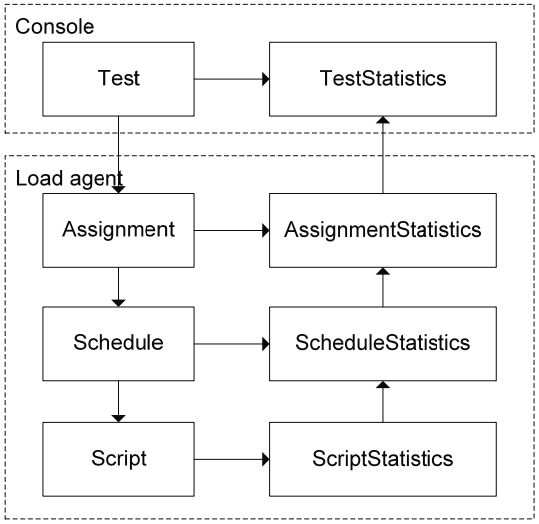

Fig. 2. Data flow through layers.

Picus provides means of extension at several places. Currently clearly defined APIs are provided for:

1) Protocols. Scripts use protocol in order to generate requests to system under test and to analyze its response. New protocol implementations can be plugged in to Picus.
2) Schedules. Different types of schedules are possible. Currently implemented schedules can execute script instance once, execute N script instances for fixed duration, ramp up number of running script instances from zero to N for a certain amount of time. Other types of schedules can be developed and plugged in to Picus.
3) Aggregators. Aggregator implements strategy of how performance statistics are processed and passed to upper layers. Currently there is single aggregator that was sufficient enough until now. However it is possible to plug in additional ones.
4) Communicators. Communicator determines how console communicates with load agents. Currently two kinds of communicators are implemented: for

communication over TCP and for communication via method calls for load agent that lives in console's process.

4. REAL PROJECTS EXPERIENCE

Up to moment of writing Picus was used in four real projects. All four projects required testing of web applications, therefore currently Picus has only one protocol module – for HTTP protocol support. Now we briefly describe characteristics of these projects and how Picus was adapted and applied for performing the tests:

1) A system was developed using Java and Struts. Tomcat was used as an application container and as a web server. Two scripts were developed. One script required unique data of specific format to be sent to application, therefore a data table support was added to Picus during that project. Data tables are CVS files from which scripts can take single line of parameters when needed. Sufficiently large data table guarantees that scripts can get unique set of data each time they need it.
2) A system was developed using Oracle tools and Oracle Web Application Server was used as a web server. Five scripts were developed that executed different data query scenarios. In requests large data amounts were used with unusual encoding, therefore support for such requests was added in Picus web protocol module.
3) A system was developed using Ruby-on-Rails and Microsoft Infopath technologies combination. MS IIS was used as a web server. Emulating work with Infopath required to add to Picus possibility to exchange XML over HTTP. Also existing cookie handling mechanism Picus used was not understood by the application, therefore cookie handling features were improved in web protocol module.
4) A system was developed using Java and JSP. Tomcat was used as an application container and as a web server. For this application there was no necessity to extend Picus somehow. It appeared completely appropriate at once.

As a result of these four projects web protocol module has been improved in many aspects and now provides possibilities that should suit most web application testing.

All those projects completed successfully and took one to three weeks of time. Technical work on test preparation with Picus (including optimizations of the tool itself) took from two to five work days. Experience acquired in these projects showed that Picus conforms well to its initial requirements and fulfills its purpose. Amount of time spent on those projects was close to previous projects where other tools were used and even slightly better. It was possible because instead of struggling with com-

plications that inevitably arise in non-trivial projects, Picus itself was easily adaptable to complicated situations what resulted in saving of time.

Now we select the first of the projects described as a concrete example and discuss it in detail in order to illustrate what results that can be acquired by testing with Picus. The customer was one of Latvian governmental institutions and the system was designed to provide services to users in other governmental institutions. The system was designed to serve up to 200 simultaneous users at peak times.

After analysis of the most typical actions that users perform with the system, two scripts were developed for emulation of these actions. These two scripts together consisted of 59 different transactions. By transaction we mean one action that is atomic from end user's point of view and whose time and correctness should be measured during the test. For example, login transaction is defined as action happening between clicking "Login" button and fully reloading the next page of application. Transaction can technically be implemented as one or more requests being sent to the server.

The test was scheduled to gradually increase number of running script instances from one to 250 during 90 minutes. So, the test load was by 25% bigger than in real conditions of peak times in order to evaluate the system's ability to scale over the designed load.

Test execution results were processed by Picus reporting module named Canis. One of the most important charts is shown at Figure 3 where average response time for each transaction is shown. We can see there that 6 out of 59 transactions lasted more than 5 seconds in average. Four of them were even longer than 10 seconds. Thus actions not conforming to performance requirements were revealed by the test.

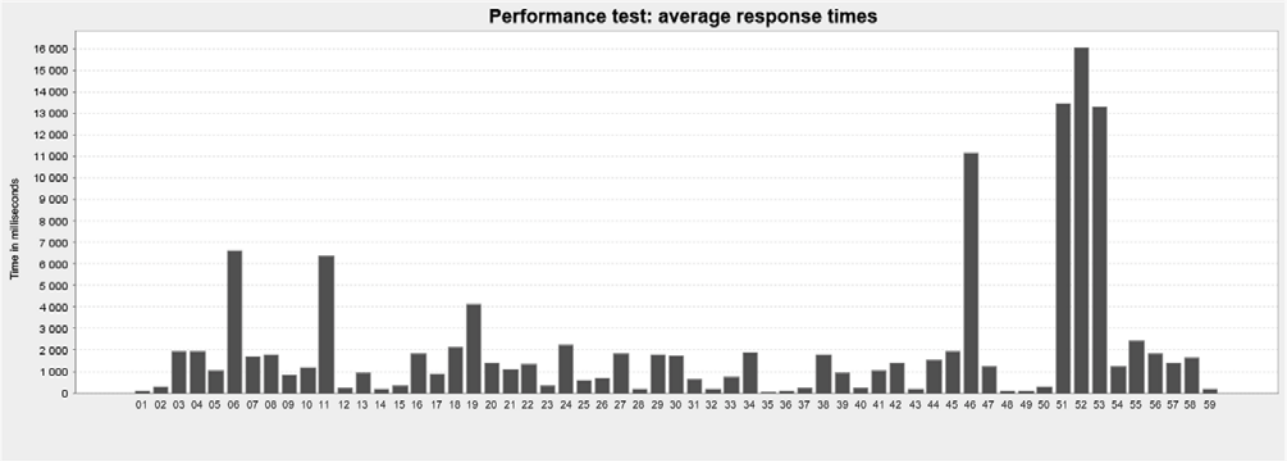

Fig. 3. Average transaction response times.

Besides underperforming transactions the test revealed also some errors specific to functioning under load conditions. Specific validation statements included in scripts allowed to check correctness of specific transactions and even of specific HTTP responses sent by system. Registered mismatches with expected results were

reported as errors and accumulated in test results. Cumulative number of errors of different types is shown at Figure 4.

Analysis showed that error types could be grouped:

1) Errors of type 3, 4, 5 and 6 were in essence different variants of the same problem – system's inability to drill down to filtered set of data that users acquire after searching necessary information.
2) Errors of type 1 and 8 were results of misconfigurations of the system which were fixed by administrators during the test itself, therefore such errors appeared only for several minutes in the test (error type 1 – at the beginning, and error type 8 – at the middle of the test) as the chart shows.
3) Errors of type 2 were caused by users inability to login. As the chart shows such problems were occurring during the whole test and was proportional to number of simultaneous users.
4) Error of type 7 is the most interesting because it first appeared only in the middle of the test when number of simultaneous users reached 130 (about 50 minutes after beginning of the test) and then number of such errors continued to grow. This was an important observation as it helped to discover major design problem in the application.

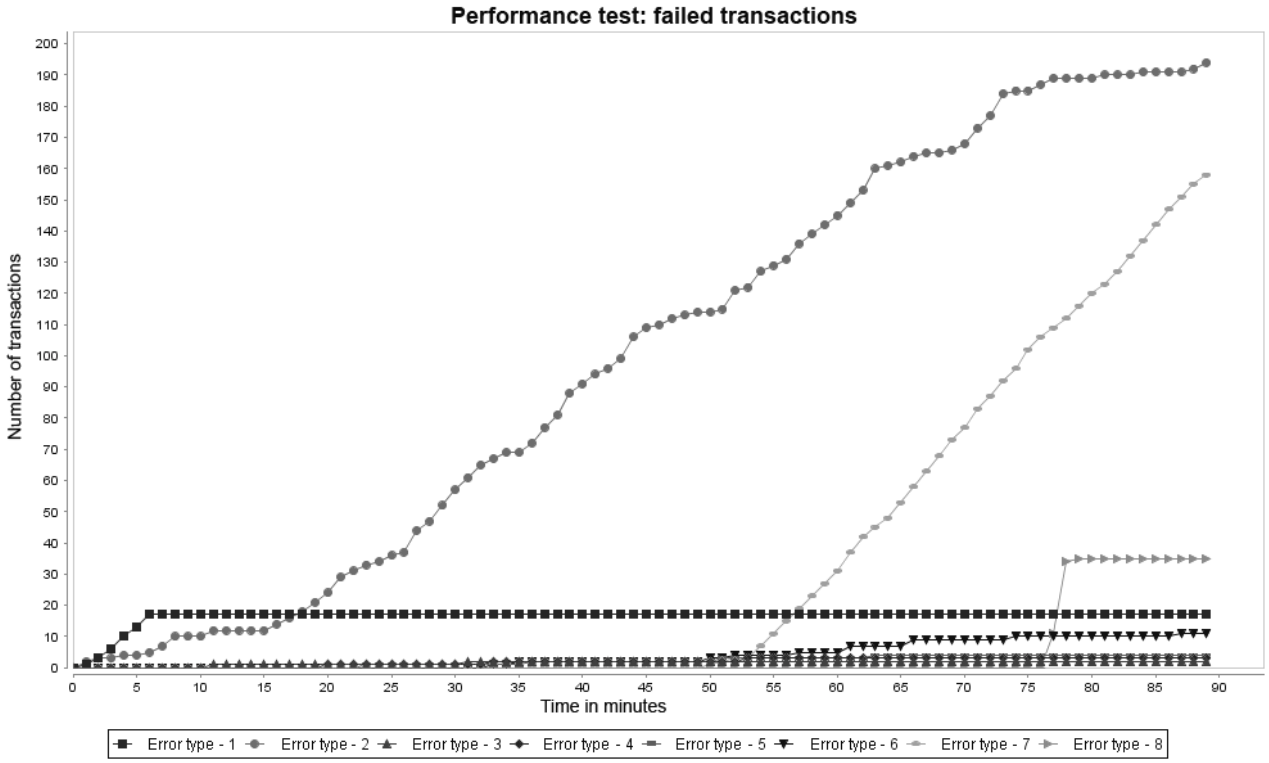

Fig. 4. Failed transactions over time.

The test gave to the customer important information that was shared with developers of the system. The results of the test were very useful for developers so they were able to fix registered problems before the system was released to production.

5. CONCLUSION

Picus development justified itself in real projects. Although Picus already supports several APIs for extension, there are ways to even improve it and provide additional APIs. Some examples of possible extension points are:

1) User interface extensions. Currently data are presented in tabular form at test execution time, but graphical representation could be more obvious. Mechanisms of visual data presentations could be developed as plug-ins.
2) Other statistics source types. Currently script is the basic form of gathering performance data. Currently only scripts implemented in Java are supported. Other script drivers could be added in future. Other types of sources could be added, for example remote performance monitors that could gather system metrics like CPU load, memory consumption, network traffic etc. Such sources should be implemented as plug-ins.
3) Statistics consumer types. Currently there is a strict data flow of statistics, which can be configured but not extended. Statistics consumers should be pluggable to define new types of usage of statistics data. For example, data could be redirected to some database, some third-party monitoring system, or to some custom report builder.

Implementing these and probably other mechanisms of extension could make Picus really universal extensible performance testing tool that could be used in various contexts. Development work on Picus continues and this vision seems to be very realistic.

REFERENCES

Ash L. (2003). *The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests*, Chapter 8. John Wiley & Sons, USA.

Hower R. (2008). Web Site Test Tools and Site Management Tools [On-line]. Available: http://www.softwareqatest.com/qatweb1.html#LOAD

Opensourcetesting.org (2008). *Open source performance testing tools* [On-line]. Available: http://www.opensourcetesting.org/performance.php

Stottlemyer D. (2001). *Automated Web Testing Toolkit: Expert Methods for Testing and Managing Web Applications*, Chapter 9. John Wiley & Sons, NY, USA.

Subraya B.M. (2006). *Integrated approach to web performance testing: A practitioner's guide*, Chapter 1-3. IRM Press, PA, USA.

Xia J., C.K.Chang, J. Wise, Y. Ge (2006). An Empirical Performance Study on PSIM. *The Computer Journal*, Issue 5 (**Volume 44**), p 509-526.