Sensitivity Analysis and Patterns Implementation on Load Testing Software Systems

Alexandra Nagy*, George Sebastian Chiş
Babeş-Bolyai University, Faculty of Economics and Business Administration, Computer Science Applied in Economics department, Cluj-Napoca, Cluj, Romania.

* Corresponding author. Email:alexandra.nagy@aol.com,george.chis@econ.ubbcluj.ro
Manuscript submitted August 5, 2015; accepted September 30, 2015.
doi: 10.17706/jsw.10.12.1402-1411

Abstract: Starting with the burst of social media, online encyclopedias and sharing resources online, performance testing is more important than ever before. Every venture knows that it should have a fast and a reliable system. Furthermore the business needs to optimize and prepare the system to meet a higher number of users as it grows. A poor performing system can face financial losses and it could also lead to legal repercussions at times. No customer wants to access a slow performing, unreliable website. It is easier to lose clientele than gain them. The system's performance is a key game changer. This article elaborates one of the most used performing testing types, load testing. It describes a set of methods and patterns used to analyze, implement, monitor and deliver load tests results. The methodology described is designed for software systems which offer a large number of service requests and typically have to handle a large number of concurrent users in a short period of time. The purpose of the article is to reveal how load testing is conducted by carrying out load tests on an online system using cloud computing resources. The two main concerns are to find the best load testing approach and to simplify the process by using the right resources.

Key words: Process of load testing, load testing analysis, performance testing, architecture of load tests.

1. Introduction

Many companies or system owners do not include the performance testing phase into their projects. And it is implicitly assumed that load testing a system must be done by the quality assurance team in the testing process. Usually that is a valid point, but there are many things to evaluate when doing load testing. If the system has more than one thousand users per day and it is still growing, load testing should be considered to be conducted by an expert or more thorough, by a group of experts.

This paper brings a new perspective on the concept of load testing. This includes both theoretical and practical levels of understanding. It does explain in details the concept of performance testing and how load testing is included in that subject. At low-level the paper presents the process of creating a proficient load testing procedure, starting from analysis and ending with the interpretation of results.

1.1. Research Hypothesis

The purpose of the research is to establish best practices to be used while conducting load testing on an

online system. At low-level the purposes are to low the costs and the time spent doing load testing. While testing on several systems and creating case studies on load testing, several research hypotheses were established:

- Running load tests on the same instance will increase the confidence in the results collected for tests executed in an ulterior period. This is a good practice for comparing results of the system while changing components or updating the system. The system performance behavior needs to be at least as good as it was before changes were made.
- Changing the number of threads used by the system into one according to the actual usage patterns of a like system will increase the reliability of the load testing process. If a new system is developed and the performance of it needs to be tested, a good practice is to compare it to an existing system that matches the behavior of the one in tests, taking into account the resources that the system has.
- The accuracy of the load testing will increase if all requests that are executed through the application are captured in the scripts. Not only the XHR request should be captured but also the JavaScript request, the CSS requests, other third parties request, HTML or other requests that could influence the load on servers.
- Doing a thorough analysis at the start of the process will increase the load tests accuracy. In this manner there will be no probability of conducting a stress test instead of a load test or not testing at the maximum power that the infrastructure supports.

The entire experiment and research revolves around the research hypothesis that all phases of the load testing process are measured as equally important in order to have an increasing accuracy of providing load testing results.

1.2. Structure of Paper

The paper is organized as follows: Section 2 discusses fundamentals of the load testing concept in comparison with other testing methods that are part of the performance testing process, Section 3 presents the analysis phase of the load testing process, Section 4 summarizes the script implementation phase, case studies developed, Section 5 presents the monitoring and the executed tests analysis results and Section 6 reveals limitations and conclusions.

2. Fundamentals of Load Testing

This section provides a set of foundational building blocks on which to base the understanding of performance testing principles and load testing principles, leading ultimately to being able to differentiate the two concepts mentioned. Furthermore it describes load testing as a process and all steps that need to be followed in each process in order to achieve a realistic load testing.

2.1. Performance Testing Concepts

In software engineering, performance testing is a type of non-functional testing performed in order to determine how a system responds and performs under a particular workload. There is no structure or definition commonly accepted and established for performance testing. There are no "best practices" of defining performance testing, but rather that this kind of testing is a set of testing types that allow the tester to select or adapt testing practices to suit each unique situation. It can be viewed as a context-driven testing practice that revolves around the fact that there is no single "best solution" that applies to all cases of performance testing. In addition, it takes into account the fact that complex software projects often evolve in unpredictable ways. Consequently, the performance tests should be also evolving.

Performance testing is a generic term that encompasses many different testing types, such as: volume testing, stress testing, capacity testing and load testing. Load testing is the most complex form of

performance testing. All load testing definitions mentioned in the technical literature have a common ground. Load testing is a performance type conducted to understand the behavior of the system under a specific expected load, during a specific amount of time.

2.2. Towards Load Testing Insights

Load testing is conducted in different ways depending on the objective and the scope that has been set. Some objectives could be: ensure that the user experience on the system is not changed due to new developed features, ensure that a new system meets the criteria of a good user experience, ensure server configuration does not affect the performance of the system, ensure that the system can sustain the continuous expected load, ensure that the system works under normal conditions when the load generated is suddenly increasing or to find bottlenecks in the system servers. Load testing can be defined as a performance testing technique conducted to determine how a system behaves under normal and peak load conditions in a specific time frame using a set of different tools and technologies. This type of testing is most relevant for multi-user systems and large-scale systems. It can be performed at API level, UI level, at a component level or at system integration level. As performance testing, load testing can be separated in different types such as: Soak/Endurance, Configuration testing, Spike, Scalability Testing and User Experience based load testing. The last one is verifying that a user experience on the system is as good or as better as it was, by running a number of average and maximum threads on the system. Data needs to be collected in order to run user experience load tests. If the system is new, then number of threads to be run should be established following the requirements of the system.

The testing types mentioned above can be combined when conducting load testing. For example when doing experience based load testing and monitoring the servers, scalability or soak testing can also be done. In order to perform load testing, thoroughness in handling the process is needed.

2.3. Load Testing as a Process

Load testing should be viewed as a complex process. A process that includes designing load tests and creating different scenarios, but it also includes doing analysis and planning. Load testing helps engineers to understand the system's behavior and availability under an amount of workload. Load testing as a process can be defined by six phases:

1) *Analysis of Data* – phase when existing records, data from the database or other software that stores data for statistics is analyzed in order to create test scenarios. Also here the environment, all other needed resources, the objective, action points, tools used and other useful information is mentioned.
2) *Identifying Key Scenarios* – this step includes establishing the scenarios based on the analysis done, objective and scope mentioned and frequently accessed areas.
3) *Identify Test Data* – after the scenarios are identified, in this phase the data used for running and designing the load tests should be established, like users, passwords, products or other useful data that needs to be collected.
4) *Designing Load Tests* – this step includes the physical implementation of the scenarios.
5) *Simulate Load and Monitoring results* – in this phase the load tests are ran on the desired environment and tests are monitored.
6) *Analyze Results and Addressing Issues* – in this phase the results are analyzed and the issues are addressed with priority and severity mentioned in the description. Results are analyzed by collecting information from monitoring the servers, but also from the graphs used. The last step is to create a load testing analysis report.

3. Sensitivity Analysis

The most significant part of load testing is conducting the analysis. In this phase the scope, the objective, the physical test environment, the production environment as well as the tools and resources available are identified. The infrastructure, risk analysis and performance acceptance criteria, most used components, the response time, throughput, resource utilization goals and constraints should be identified. Analysis process should be revisited periodically throughout the project's life cycle. Load testing is conducted with more than one objective. Consequently, in order to conduct a realistic load the objective and the scope of load testing should be established.

3.1. Building Use Cases

Use cases need to be created in the analysis phase and are used to define interactions between roles and the system in order to achieve a goal.

In this case study the system is an online grocery website and the actors are: an anonymous user and an authenticated user. A signed in user has access to all the systems components as presented in Fig. 1. An anonymous user cannot proceed to checkout and has access to the following components: products, categories, circulars, sign in, edit account, store locator and cart. The objective of the load is to simulate a simple user flow that can reach all the system components in the states mentioned. The use case is a simulation of the real user flows on the online grocery application. Previous to the creation of the scenario a use case is created.

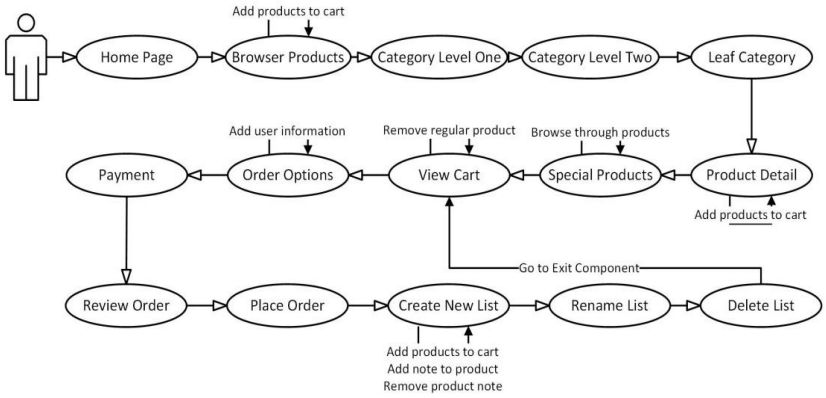

Fig. 1. Authenticated user use-case.

3.2. Data Analysis

The primary goal of the load test is to establish performance benchmarks for the updated system. This will be accomplished by simulating end-user activity against the website and against the APIs. Information that helps in the description of the load test process should be gathered: the date range from where test data was collected, environment from where the data was collected, the testing environment, the application build, the URL/URI and the average page load. Next, the minimum and maximum data needed for execution is collected in the interval of time often accessed by the users. For example the interval selected was from 8:00 am to 10:00 pm as the data revealed in Table 1.

To create the load tests several action points are set: establish the amount of time to run through the scripts, establish the number of threads, establish the number of pageviews per user, establish if session duration needs to be used, create comma-separeted values files with test data, create scenarios based on most viewed pages, take into consideration exit pages and establish what events to add in the scripts based on most used event actions. After objectives and action points were discussed the next step is to establish the number of threads and samples to be run and the amount of time to run through the scripts. For this test to be reliable the calculated number of threads should be calculated as average and maximum. The

maximum value should refer to the number of threads that can be used in order to not overload the system and fail the servers.

Table 1. Overall Analysis

	Users	Pageviews	Bounce Rate	Avg. Session Duration	Pages/Session
Max	4908	29601	26.74%	0:15:46	32.43
Min	397	7550	2.95%	0:03:48	4.85

First a number of average threads should be running for fifteen minutes. To find the number of average threads a peak number and a minimum number of concurrent users is collected during the same time frame. In the study case the number of average threads was collected per day. A formula was established to extract the number of average users from the collected data, presented in Table 2. A median was made from the minimum and maximum number of concurrent users and then divided by the number of hours in a day, divided by the number of minutes in an hour and then multiplied by 15 which is the number of minutes the scripts needs to be executed. Assuming this result equals to "a", "a" will be equal to "a – a x bounce rate". Now "a" contains the number of average users. The same calculation is made for maximum number of threads that can be viewed in Table 3.

Table 2. Average Threads per 15 Minutes

Data collected is based on this Formula	$\left\{\frac{Min+Max}{2}\times\frac{1}{24}\times\frac{1}{60}\times 15\right\}=a-\ a\times BounceRate$	*Users*	663
		Pageviews	4644
		Bounce Rate	14.85%
		Pages/ Session	18.64

Table 3. Maximum Threads per 15 Minutes

Data collected is based on this Formula	$\left\{Max\times\frac{1}{24}\times\frac{1}{60}\times 15\right\}=a-\ a\times BounceRate$	*Users*	1227
		Pageviews	7400
		Bounce Rate	26.74%
		Pages/Session	32.43

Table 4. Load Test Execution Plan

Run One – 15 minutes	Anonymous user – average threads
Run Two – 15 minutes	Anonymous user – average threads
5 minutes break	
Run Three – 15 minutes	Anonymous user – maximum threads
Run Four – 15 minutes	Anonymous user – maximum threads
15 minutes break	
Run Five – 15 minutes	Authenticated user – average threads
Run Six – 15 minutes	Authenticated user – average threads
5 minutes break	
Run Seven – 15 minutes	Authenticated user – maximum threads
Run Eight– 15 minutes	Authenticated user – maximum threads

Load test scenarios are designed based on the created use cases. But also based on the data collected. After establishing the scenarios, a plan for executing the load tests is created. Each scenario will be run two times for fifteen minutes. The signed in scenario will be run with the average and the maximum number of threads, two times each for fifteen minutes. Also, the anonymous scenario will be run with the average and the maximum number of threads, two times each for fifteen minutes. The load test execution process will take two and a half hours without collecting the results as presented in Table 4.

The results should be collected after running each script. After this phase of analysis, guidelines should be made so that every individual involved in the load tests execution process knows how to run the load tests. This is also useful for making a prerequisite list of what tools are needed in order to run the tests.

4. Patterns Implementation

The analysis introduced in the previous chapter helps in designing meaningful load test scenarios. The next step is to design the load test script. In this section a general load test designing technique will be presented.

4.1. Load Testing Patterns Research

The objectives of the load testing could be very different. Sometimes more than one objective is tested with different, but similar scripts. In this case a template was created as a general reusable solution. Patterns are important to be thought off when doing load testing. The script can be used for multiple purposes but with minor changes. Using patters in load testing means less time spent on implementation but more time spent on analysis. If the requirements set by the person in charge or by the client are not proper set then the implementation of the scripts could increase the costs of the project. This usually happens by changing load requests in the implementation phase. Pattern research can be done in the analysis phase based on the scope of load testing.

4.2. Architecture of Load Tests

The architecture of a load test should be well structured in order to have a good performance of the script. Issues may rise because the architecture of the application is not very familiar to the person in charge of implementing the script. Usually experts should have some time to study the application's architecture.

A system can use HTTP or HTTPS protocols or protocols like FTP, TCP or other. Also a system may manage cookies mandatory or not. If these properties of the application are not thought of in design then the scripts may not function properly. The architecture should contain some of the following elements of a test plan: thread groups, listeners, timers, configuration elements, default elements and test data elements. But all the elements mentioned have to be set in a proper order. Starting at the top of the load test architecture, configuration elements are the first. They can be: HTTP request default, cache managers, cookie managers, header managers, authorization managers, login managers or test data configuration elements. Next listeners and the thread group are added. In the thread group, loop controllers, timers, configuration elements and samples can be designed. In the loop controllers other samples can be added and in that samples post-processors, pre-processors and assertions. Problems with the script implementation can appear at the architecture level. For example: if the cookie manager is not used, samples that use authentication cookies may not work; the system uses HTTP and HTTPS for some pages, then the request default port number should be HTTP by default and HTTPS should be added in other samples; if the tool used for load testing is JMeter 2.10 or a newer version of JMeter then for the cookie manager configuration element, HC3CookieHandler and HC4CookieHandler can be used, the last one was introduced for the IPv6 address; performance of the script can be low if configuration elements are not used so we do not have redundant test data.

In addition architecture of load tests should be defined as a collection of components organized to

accomplish a set of testing functions in order for the load test to fulfill its mission for a specified system. Also this includes a set of significant decisions about the organization of the script elements and the selection of the structure in which the elements should be set.

4.3. Designing Load Tests

Designing load tests is the easiest part to do when the analysis provided fits the context. Regarding of the used tools load tests will always be designed in the same way, with almost the same elements. A relevant step when creating a load tests is to analyze the software in which the load tests will be build. It is also important to verify if other plug-ins are necessary to be added in order to perform a realistic load. Keep in mind that the script may not work on older versions or in new versions that do not have the plug-ins installed.

First in the test plan a thread group is added. This element will control the number of threads, virtual users to be used when executing the tests. Also other configurations can be set: the ramp-up period, a scheduler or a number to configure how many times to run the thread elements. A test script recorder should be used. This is a non-test element provided by all load testing tools. The recorder is used so that all requests made by the system are saved. The system requests are recorded and monitored in a listener sample. A simple user flow through the application should be done, to save the requests. The next step is to remove all unnecessary requests. Like JavaScript, CSS, HTML, Google Analytics requests or other third party requests that should not be used. A good practice that can be followed when recording to avoid having files that are unnecessary is to add patterns to exclude files. After setting the basic flow, variables should be added where possible. For example if there is a POST add to cart request then the items that are added to the cart should be different. In order to do that a variable must be set because in real life users add different products to cart. When dealing with variables a good practice is to use a CSV files. All variables used can be added in the document. For example the product name, description and price can be added and then extracted from the CSV. Each time the POST add to cart request will be performed the software will extract in order one of the products from the CSV. Also sometimes variables can be extracted from other requests. For example a GET categories request can return categories with items and with items GUID in the JSON response. The POST request for add item to cart must contain the item's GUID. So the GUID can be extracted from the first call with a post-processor extractor. Post-processors use regular expressions to extract data.

Next step after all request are set in place is to make sure that the script will run without any issues. The script should be run first with one virtual user and the response code for each request should be verified. If the code returned is a positive one than the script should be run with more than one user, to verify that there are no concurrency problems. Usually concurrency issues appear because load tools do not run the requests in order. If you want some of the request to be run in order, a throughput controller should be added. In case there are no issues and the execution was successful the next step is to add other configuration elements. Configuration elements allow the script to use less data and to be more efficient from a performance point of view. The person in charge should make sure that there are no bottlenecks caused by the script implementation. Things to be avoided here are: using redundant data, using unnecessary data, using a lot of listeners that do not help with monitoring or are redundant, using URLs in each sample, not using a HTTP header manager or using samples more than once. For example a HTTP request defaults can be used to store the base URL and port number of the system. In this manner there will be no need to add the base URL in each sample. Also if in other samples another URL is added from a third party service, the URL from the configuration element will not be inherited from the default one. The load testing tool has a default configuration element from which all samples inherit a property. But if samples already have another property they will override the default one.

Last step after all configuration elements have been added is to add listeners and timers. Listeners are

added based on what is the objective of the final results. For example the client may want to know the maximum number of transaction throughput per users or the maximum transactions per second. In order to provide the expected analysis report listeners must be added. A best practice is to add the most used and relevant listeners: simple data writer, aggregate report, aggregate graph, assertions result, composite graph, response times over time, response times percentiles, active threads over time and transaction throughput versus threads. Memory consuming elements may decrease the performance of the script and therefore should not be added. An element can be memory consuming because it saves all the requests made during the execution of the script. A best practice is to disable this element when scripts are run. Similarly timers can be added to the load test script. The scope of the timers is to allow a delay between each request which a thread makes. If timers are not used then the requests will be sent without pausing between them and that could overload the server. Also this is not a real case scenario, because users would not sign in at the same time and do the same flow through the application.

Once all the steps above are finished. Then the script can be configured to run the number of expected virtual users that was set in the analysis phase.

5. Monitoring and Results Analysis

The objective of this section is to present fundamentals of reporting load testing results, to emphasize the importance of monitoring and to describe how monitoring systems under load test should be performed. Furthermore to learn how to apply principles of effective reporting to load test data, learn when to share technical results with possible stakeholders and what questions are expected to be answered from a load testing report perspective.

5.1. Monitoring While Load Testing

The success of a load testing project consists in proper monitoring and proper results analysis. A good testing process has proper monitoring in place during test execution. Monitoring should be mentioned at the start of the project in the analysis plan. How it is performed, by who it is performed, what experts are needed and how will the issues by investigated. The results analysis goes hand in hand with monitoring. The results provided will be as good as the monitoring phase was. The most important fact to take into account is to make use of what resources are available.

The load testing should be planned so that it can produce separate results for each active server in the testing environment. It should always include transaction responses times. Also make maximum use of metrics available during testing. All available results should be reviewed and analyzed carefully even if no issues are found and graphs should be properly monitored. A good practice is to compare graphs with similar trends or patters and look at resources that reach a maximum at a point where another graph shows a problem.

When there are no specific requirements for monitoring several best practices can be followed: monitor website resources - hits per second, throughput, connections and HTTP responses per second; monitor transactions - transactions responses times, transactions responses times under load and transactions responses time distributions; monitor Windows and UNIX resources - memory utilization, average load, CPU utilization, system mode CPU utilization and user mode CPU utilization; monitor IIS and application servers: bytes sent, bytes received, GET requests, POST requests, maximum connections and current connections, used memory, free memory, connection pool size, rolled back, timeouts and current requests and session size; monitor the database server - cache hit ratio, user connections, memory, CPU usage, current open cursors and current logons. If there is a table in which errors regarding the application, services or servers are logged then that tabled should be also monitored; monitor the graphs incorporated in the load testing tool. Also use a monitor tool for servers if there is no internal tool used for monitoring

servers; ask experts to help with the monitoring process, a database administrator and an architect may be very useful when identifying issues caused by the load tests or issues that are blocking the load testing process.

There are many metrics available for monitoring. A best practice is to rely on architects, network experts, database administrators and experts to help with the monitoring phase. Also provide them with specific monitoring requirements. This usually helps the relevant individual involved to know what to look at when load tests are executed. Monitoring can be performed with a multitude of tools depending on the technology used.

5.2. Load Testing Analysis Report

The most challenging part of the load testing process is the analysis of the test results. After monitoring tests during execution and finding bottlenecks, a general analysis of the application and the infrastructure should be performed. Load testing does not mean that the application performance is tested. It means that the application performance within the infrastructure and the architecture available is tested. In order to provide the results found during load testing, analysis of investigated performance issues should be conducted. Analysis allows the expert to study the information on a report and determine cause-effect relationship between results. Reporting and presenting the information should be carefully performed so that all issues are easy to understand. Usually the results should be presented in a commonly interpreted way because the client may not have the necessary technical knowledge.

The key to effective reporting is to present information of interest to the intended audience in a quick, simple and intuitive manner. Equally important is to follow some principles of effective reporting like: report visually, report intuitively, use the right collected data, consolidate data correctly, summarize data effectively, customize reports for the intended audience, use concise verbal summaries and make the data available. In the analysis report graphs can be used along with tables. Reports can be designed in any manner but with relevant data. The final analysis report that will be sent to the client should contain high-level information. A good practice is to design first a low-level analysis report and then extract useful information to add in the high-level analysis report. Why is a low-level report useful? It is useful first because usually clients and management ask for concise information. Secondly it is useful because it helps the person in charge to view the general analysis of the results and at extracting the right data.

A high level report analysis should include: an overview, scope or objective, client name, environment used, assigned expert, code version, code build, load test execution location, virtual machine location from where the tests were executed, virtual machine IP, URL of application, date when the tests were run, system's level of performance, measurements, infrastructure issues, application issues and recommendation. For example the performance level was: good – error percentage was between 3% and 7%. The measurements: production environment supports a maximum number of 2580 concurrent users per hour. The maximum number of transactions per second compared to the maximum number of active threads is 18000 transactions per hour. CPU usage in production may increase to 100% with 2580 active threads that are doing 18000 transactions per second. Also the number of active users can increase if the number of transactions per second is decreasing. Examples of infrastructure issues: cart API application pools may stop after the CPU usage of the IIS servers is above 70%-80%. The recommendation is: if the number of visitors will increase to a percentage of 10% in comparison with the last maximum workload then a new IIS and DB server should be added to the system's infrastructure. For cart API a bigger space should be allocated on the principal disk of the IIS servers. The conclusion was that overall the load testing process went good and no major issues were found.

The most important part of the load testing process is providing useful information to the stakeholders. If the information is not accurate and not close to the real usage of the system then entire process can fail.

6. Conclusion

The paper has identified that all phases of the load testing process are measured equally important. The general theoretical literature on this subject and specifically in the context of load testing is inconclusive on several subjects. The study sought to answer several of these inconclusive questions. Methodological challenges were encountered during the load testing phases. As a direct consequence of these methodological challenges, the study encountered a number of limitations, which need to be considered. Results are measured through five important phases of the process described along the thesis: analysis, script implementation, execution, monitoring tests and results analysis. The benefits of doing a thorough load testing have been shown comprehensive and sustainable in the short-term. However due to technology changing every day and the rapidly increasing number of users, methodologies and patterns need to evolve as well.

To conclude, there is clearly much work to be done in the area of load testing software systems. Perhaps the most direct influence of this work is increasing the knowledge of the load testing domain to a point where the individual can perform a proficient load test on a complex software system while simplifying the process by using the right resources.

References

[1] The apache software foundation community. JMeter Documentation last updated in 2015

[2] Mark, D. A. (1999). The top 13 mistakes in load testing applications.

[3] Daniel, A. M. (2002). Load testing, benchmarking and application performance management for the web. *Proceedings of the* Conference *on published in the 2002 Computer Measurement Group (CMG)*.

[4] Dan, D. (2000). Web application bottlenecks exposed: The most frequently found performance problems and how to Nail them!

[5] Steve, S. (2007). High performance web sites – Essential knowledge for frontend engineers.

[6] Dmitri, N. (2006). Using jmeter to performance test web services.

[7] Budi, K. (2003). Using jmeter. Retrieved 2003, from http://www. Java.com

[8] Menascé, D. (2002). Load testing of web sites. *Internet Computing*, *6(4)*.

[9] Yuval, S., & Zohar, G. (2002). Service for load testing a transactional server over the internet.
Draheim, D., Grundy, J., Hosking, J., Lutteroth, C., & Weber, G. (2006). Realistic load testing of web applications. *Institute of Computer Science*. Freie University, Berlin.

Alexandra Nagy received the title of bachelor of arts in economics following computer science applied in economics study program and the master of science following economic decision support systems at the Department of Cybernetics, Statistics and Business Informatics from Babeş-Bolyai University, Cluj-Napoca, Romania in 2012 and 2015, respectively.

Her previous research interest included website development solutions on e-commerce. In 2011, she has written a paper regarding the investigation of new web technologies hypertext mark-up language 5 and cascading style sheet 3 in comparison with Adobe Flash for Babeş-Bolyai University, the Department of Cybernetics, Statistics and Business Informatics, Cluj-Napoca, Romania. Her current research interests are focused on load testing processes, sensitivity data analysis and patterns implementation on online large scale systems within cloud infrastructure.