Best Practices for Web Application Load Testing

This paper presents load testing best practices based on 20 years of work with customers and partners. They will help you make a quick start on the road to a productive and effective load testing program.
Contents
Getting Started with Load Testing ................................................................. 3
Planning a Performance Testing Program ......................................................... 3
Recording a Script ............................................................................................... 4
Handling Dynamic and Static Values in Test Scripts ......................................... 4
Testing System Behavior with Validation Logic ............................................... 5
Running the Load Test ......................................................................................... 5
Analyzing the Results ......................................................................................... 6
Getting Started with Load Testing
To successfully test the performance of a web application, tools are not enough. From planning, through test development, execution and analysis, you need to ensure that your testing solution is exercising the right functionality, under realistic conditions, while gathering the information you need in order to identify and resolve bottlenecks.

In this paper, we have assembled a set of load testing best practices based on 20 years of work with customers and partners. They will help you get started on the road to a productive and effective load testing program.

Planning a Performance Testing Program
To ensure that your testing program is comprehensive, consider the following:

- Define how long a test must run before the servers fail. Is an hour good enough? A day? A week? Longer time periods will uncover different types of problems such as OS bugs, app/tool memory leaks, etc.

- Create a testing methodology that addresses the behavior and response when the system is overloaded. For example, when the application fails, what happens and when? Which log files need to be checked when a failure occurs? How do we recover? What are the important trouble indicators to look for and how do we monitor them?

- You need to perform load tests over and over, so make sure that tests are repeatable. Define a methodology for handling backups.

- Define how performance testing fits into the deployment process in your organization.

- Identify the most common workflows in the application you’re testing. For an existing application, check server logs and analytics to reveal the most frequent scenarios. For a new application talk to the product management team to identify the relevant user stories.

- Plan load scenarios that cover a broad range of usage conditions, from light usage, through standard loads and known peaks. When defining each type of load, take into account a variety of users doing different activities in different working environments including mobile.
If your organization has implemented continuous integration, plan and create a set of regression tests that can be executed on every build. WebLOAD’s integration with the Jenkins Continuous Integration server can help you to automate this process.

If your organization uses App Dynamics or another APM tool, you can include it during the analytical process to help identify the root cause of performance issues more rapidly.

**Recording a Script**
Recording is the easiest and fastest way to create the basic test scripts that simulate realistic usage scenarios. While you can improve on your scripts afterwards, it will save time if you record as much functionality as possible first.

- While you are recording, mark the beginning and the end of meaningful business transactions that must meet a performance SLA.
- If your web-site has a mobile application or a mobile specific web-site, don’t forget to record scripts for it as well. You can either simulate the mobile device in your browser or directly record from a mobile device.
- Make sure that you do not record third party code, such as a Facebook share, as part of your load test script. If it is a part of the test flow you can also filter out third party code after recording, if you have noted it.

**Handling Dynamic and Static Values in Test Scripts**
Modern web applications use dynamic values in every session, so recorded test scripts often cannot be replayed without modification. At the same time, it’s important to replace static values with parameters that can be populated at run time.

- First, use WebLOAD’s automatic Correlation Tool to locate and replace all of the dynamic values in the recorded script, such as session-id and security tokens.
- Based on testing thousands of different web applications, RadView has developed a library of correlation rules that cover the majority of dynamic variables and guide you through the replacement process. You can create additional rules to handle any special cases in your own application, and save them in the library.
- Some dynamic values are randomly generated or involve the current date. You can replace the dynamic value with a JavaScript expression that creates the right type of number so that WebLOAD can generate it at run time.
• Make sure you stress the application’s back-end - and not just the cache servers – by replacing static values with parameters that will be automatically defined at run time.

• Make sure that parameters are used as they are in a real application, for example, the username and password must match, but the product can be different, or the same username cannot be used concurrently, but the same product can – create separate parameter sets when needed.

Testing System Behavior with Validation Logic
While WebLOAD will check many performance indicators for you automatically, only you know how your application is meant to behave. By adding validation logic to your tests, you can ensure that the application keeps working as expected as the load increases.

✓ Add validation logic to the script to make sure that the results you get under load are consistent, valid and expected. You can use existing logical building blocks from WebLOAD, or create custom validation logic.

✓ You can validate that a request has completed correctly based on many things including: page (HTML) content, content size, Page title, end Page time, etc.

✓ Reuse custom logic by creating functions and saving them in the validation logic library.

Running the Load Test
For test execution, you build load templates that combine the various scripts you have created under different patterns of load and different scenarios. It is important to consider a broad variety of factors in order to test various objectives such as speed and scalability, application behavior, and reaction to exceptional peaks.

✓ Make sure that you test the application in a production-like environment, including factors like SSL, SSO, load balancing and firewalls. This is essential for simulating accurate end-to-end behavior.

✓ Create realistic runtime scenarios that take into account the test scripts, the number of users, browser types, network limitations, bandwidth, caching options and schedule.

✓ Define the SLA or set of criteria for a successful test. Run different tests and scenarios to check system capacity, system stability and endurance, and scalability.

✓ Avoid the temptation to extrapolate results – one server with 100 users may not run like 2 servers with 200 users. Test the equivalent of your full production stack.
Since a load test scenario can run for a long time, it is a good idea to check on test statistics during execution to make sure that things are running smoothly.

Collect statistics from your back-end environment including web, application and database servers, to diagnose load impact on all applications tiers.

Make sure your load-generators are not overloaded - which might invalidate or skew your results and statistics. Consider using the power of cloud computing to add extra computing power when needed.

Test often. After completing the initial load test, use the results as a baseline, and look for regression in performance – a seemingly insignificant change can cause unexpected performance issues. The easiest way to do this is though automation. The WebLOAD integration with Jenkins Continuous Integration server can be used to automate the regression testing process.

Use a Probing Client, a load generator with a single Virtual User that will help you to understand the user experience (like response times, throughput, hits per second) while the application is under load. You can leverage the cloud to analyze single-user behavior from different geographical regions.

Analyzing the Results
Once a test run is completed, you are left with both performance time and a collection of statistics from your systems. WebLOAD helps you to correlate the different information sources in order to identify the cause of bottlenecks, and it is recommended that you also try the practices below.

It’s important to check all of the application parameters including the number of connections, response time, transaction times and throughput. There are many reasons why an application can break under load and the actual culprit is not always easy to predict.

There are many ways in which an application can “fail” a load test – it can fail to respond, respond with an error code, fail your validation logic, or respond too slowly.

Analyze server side statistics side-by-side with client side statistics and look for any discrepancies that might indicate a problem.

WebLOAD reports contain built-in rules that encompass industry knowledge and best practices – use those rules to find issues quickly.
✓ Take advantage of your company’s production monitoring tools where relevant, to get a deeper understanding of the server’s behavior, for example, WebLOAD integrates with AppDynamics to provide enhanced reports.

✓ To communicate test results effectively, generate reports for each team member in their preferred format.