



Why load test your Flex application?

Your Flex application is new and exciting, but how well does it perform under load?

Abstract

As the trend to implement Web 2.0 technologies continues to grow and spread throughout the development community and enterprise organizations world-wide, Adobe Flex stands out as the Rich Internet Application (RIA) framework of choice for many (about 30%, according to Forrester Research). Flex enables users to rapidly develop applications with a new and exciting look and feel, that enhances the end-user experience.

Like most new technologies, nothing comes without a price. Designing your Flex based application to achieve maximum performance is still a requirement. If you do not validate the scalability of your Flex application, you will likely wish that you did later on.

In this whitepaper, we will explain the concepts of software performance and performance testing. We will focus on why it is absolutely critical to test Flex based applications for scalability and performance. To demonstrate the need, we have included a case-study, showing how one organization benefited from load testing their Flex based application with our WebLOAD for FLEX solution.

Introduction

Adobe Flex is a framework for developing and deploying cross platform Rich Internet Applications (RIAs). The Flex framework combines the strength of traditional Adobe Flash animation with a robust backend framework capable of supporting enterprise grade internet applications with a superior user experience.

Underneath the sophisticated user interface, the Flex framework provides different communication channels and data exchange methods. It includes multiple components for connecting to backend services for updating and fetching data from remote resources.

The complexity of Flex is further increased by the need for high-performance data connectivity with existing server-side modules and business processes. The LiveCycle ES is a server-based technology introduced by Adobe that includes sophisticated communication services like publish/subscribe messaging, real-time streaming data and data management services.

Most Flex based applications are deployed in a multi-tier architecture which adds additional complexity due to data interaction between remote services and integration between multiple third party components.



As part of any Flex application development lifecycle, most enterprises would most likely want to ensure that the application meets their performance requirements. They would need to find out the answer to some key performance questions, such as: How many concurrent users can my application handle while maintaining the desired level of performance? How many concurrent users can my Flex application support before its performance suffers? At what number of concurrent users will my application fail?

To answer these questions, and others related to a Flex application's performance, enterprises should include performance testing (sometimes known as load testing) in their Flex software development lifecycle.

Flex Performance Testing

What is Application Performance?

When talking about application performance, two key factors are typically discussed: response time and availability. The response time of an application is the measurement of the time required for an application to respond to a user request or interaction. This measurement can be broken down further to calculate the speed of a certain functions within your application, or the speed of a certain transaction the end user performs. Application availability is a measurement of what percentage of time an application is available to the end user. Many times this is described as a percentage of "uptime" during business hours. If your application must be available 24x7, any downtime can hinder the end-user experience and likely cost your company revenue. If an application does not respond to a user request, the user perceives it as essentially unavailable in the end.

When building a new application, the business will probably define performance requirements related to these factors. For example, the time it should take the application to return an answer to a search query will not be more than 1.2 seconds; or the application should be available 99.95% of the time (which is equal to 43.2 seconds of downtime per day, or it means that only one of 2,000 requests to the application is allowed to return an 'unavailable' reply).

Such performance requirements universally apply to all applications. For example, your favorite spreadsheet application must perform at a certain level to meet your business needs. These requirements become more complex and difficult to manage when building a multi-user web based application. With such an application, you need to understand how these two factors (response time and availability) change as application usage levels change (in terms of concurrent users) while supporting the business. You also need to validate the performance of your web based application under expected peak use workloads. To accurately achieve this, you must include performance testing in your application development lifecycle.



What is Performance Testing?

Performance testing (performance verification) ensures that application performance meets the business needs as defined by the performance requirements. As explained above, it is most relevant for multi-user systems, and exercising all tiers of an application's infrastructure.

Performance testing needs to test an application under heavy load (in terms of peak concurrent users using the application), it is done most efficiently using tools that (1) enable simulation of multiple users accessing the application's services concurrently and (2) automate the testing process to verify the application's response time, availability and tolerance for user load . Hence this is why it is commonly referred to as load testing.

It would be difficult to imagine coordinating thousands of people at a work site and having them manually run business transactions and processes , while measuring in real-time how the application performs and behaves as the load fluctuates. Instead, the preferred method is to use an automated load testing tool to generate 'virtual clients' that simulate real-world users and real world application usage.

Another important benefit of load testing using such a tool, is that it generates synthetic load while measuring the application's response time and the performance of the underlying infrastructure. By measuring the response time of every request sent to the application, while collecting server-side performance statistics (such as CPU utilization and memory consumption), a load testing tool can present in a clear and visual manner, the points of performance degradation. It can also correlate the application's response time with the state of various system resources, giving insight into the root cause of performance problems.

The Performance Testing Process

The load testing process can be divided into four stages:

The first stage is the planning phase and considered by most as the most critical stage. In order to determine if the application performs well under load , you need to first define how you expect the application to behave as the workload increases. The following tasks are typically included in this phase:

- Review the application's performance business requirements
- Define the relevant performance measurements (such as throughput and CPU utilization) that should be measured to indicate the system's behavior
- Define the test goals or the desired performance Service Level Agreements (SLAs) that will be used to judge the success or failure of the application's performance

The second stage is defining test scenarios. In this phase you define and document the business processes that you need to test. Creating the test scenarios is usually done by using a load



testing tool to record the traffic sent from the client to the application's servers during real-world usage. The outcome of this recording is commonly referred to as the test script. These test scripts can be parameterized with data from external files, to allow simulation of traffic generated by a large volume of real users. For example, a test scenario that emulates purchasing a specific item from an online store, can be parameterized so that during load test execution, the recorded purchase transaction will be dynamically changed to simulate the purchase of many different items. Using this approach, you could test the performance of the application while thousands of virtual users purchased different items.

Additionally, it is recommended to include verification points in the test script. Application availability is not only measured by the fact that the server sent a response for each request. Sometimes under load, the application may return a response within the desired time, but it would be logically wrong. Everyone has encountered error messages returned by web applications and a load testing tool can verify that the application responds as expected. This can be done by examining the contents of the application's response. This can be achieved very easily, if the response can be made accessible using some kind of object model (like HTML or XML DOM) that allows direct access to the elements of the application.

The third phase is the execution phase, where the test scripts are executed according to the previously defined load-test scenarios. These load-test scenarios include a mixture of test scripts, simulating the real world mix of business transactions expected from the users of the application. Factored into these load-test scenarios is the schedule (when the test should start and its duration) and the load profile (how the amount of traffic and users should vary as a function of time). While the load test is executing, the load testing tool measures and captures the various performance measurements and runtime statistics from the different backend system components. These measurements are displayed within the graphical user interface (GUI) and stored in a database that is available after the test is concluded for further analysis.

The fourth and final phase is the analysis phase, where the results are analyzed and the application is tuned to enhance performance. In this phase the test engineers analyze test results, compare test runs and attempt to isolate any bottlenecks that are the root cause of any performance problems. The test engineers often collaborate with the application developers during this phase to tune the application and the system components to improve performance. This process is typically an iterative process and repeated until all performance issues are resolved.



Why is it Critical to Load-Test Flex Applications?

The performance of your Flex application depends not only on the Flex client (how it was developed, how complex it is, how fast the GUI components render on the end-user desktop), but also on the application's servers and their ability to scale with an increasing user load. Typically the latter is the more relevant factor in determining the performance and scalability of a Flex based application.

Flex based applications require a high level of data integration which increases their complexity. Providing a richer, more engaging user experience requires more data-intensive interactions and introduces new challenges in managing data between the client and the server tiers. Despite best efforts in designing and implementing a high-performance system, it is natural for such a complex system to require tuning to meet business performance requirements. This tuning should be done using a load testing tool to simulate the load of real users.

Flex is new technology and each deployment will likely implement a different combination of features and configurations. The collective body of best practices and experience with Flex in the marketplace is still in its early phases and the only way to ensure that a Flex application will scale to meet the business needs, is to implement a repetitive load testing methodology.

Flex is a rapid development environment, enabling quick integration with existing components like web-services and J2EE objects. The typical Flex application is a highly complex, multi-tier system, with more potential for performance bottlenecks than a typical web application. A typical web application is composed of the traditional three tiers of web-server, application-server and database server. Flex introduces the remoting gateway (for example the open source Adobe Blaze DS or the Flex LiveCycle ES) as a fourth tier. This element provides extremely attractive and productive services that will shorten application development time. These same services may also introduce potential performance problems.

Due to the complexities described above, it is not enough to test Flex application components locally. An end-to-end approach that will test the system from the user's perspective is required. You need to understand how the different components of the system interact under real-life usage scenarios. You need to be able to identify any performance bottlenecks, errors, and/or instabilities as the load on the application increases over time. In many cases, the only way to surface certain performance problems is by placing the application under real world stress loads.

Performing load testing on your Flex application will:

- Reduce the high cost of mission-critical application failure
- Identify your application's "normal" behavior patterns



- identify the scalability and user capacity of your application
- Locate potential performance problems before they occur in production
- Reduce deployment cycles and speed your time to market
- Predict and/or reduce infrastructure costs
- Help you tune your application for maximum performance

Getting Started with Your Flex Load Testing Project

This section reviews some of the considerations in load testing Flex based applications. These include the test environment and choosing the best load testing tool.

Building the Test Environment

The best performance test is one carried out on the actual production system under real-life conditions. However, this is often not possible as we cannot always find an idle window in the production system's schedule. You do not want to hinder the application's responsiveness to the needs of real users by adding synthetic load generated by the load testing tool. Ideally you would like to use a dedicated test environment that is identical to your production environment. Often this is cost prohibitive and technically challenging to achieve. Your production environment is designed to scale with room to spare and includes high-availability mechanisms to prevent various infrastructure failures.

The common approach is to build a smaller scale "copy" of the production environment and begin from there. In order to have 100% confidence in the results of your performance tests, you should schedule at least one "peak-usage" (stress) test that will be done against a larger environment closer in scale to the production environment.

Choosing the Right Tool

Performance testing is a complex task in itself. Some even call it an art and some call it a science. For the reasons listed above, Flex applications are more complex than traditional Web 1.0 and most other Web 2.0 applications. With this level of complexity, a robust tool is required with the right capabilities to support the test engineers trying to perform the task at hand. Among the necessary capabilities are:

Recording: Flex uses a binary data protocol known as Action Message Format (AMF) to communicate between the client and the server. It is critical that your tool be able to read the data exchanged by the client and the server and provide a human-readable representation of the binary messages. WebLOAD deserializes AMF binary messages into JavaScript objects.



Editing and debugging: The test scenarios created by the tool using the above record feature should not only be readable, but also support direct access to the discrete message elements, allowing easy parameterization and verification of messages. Parameterization will allow the user to edit and modify test scenarios and associate them with dynamic data from external sources. For example, you may want to modify the login transaction to use different user names and passwords for different test runs.

Content verification: Failures often take the form of erroneous data returned by a remote service. Therefore, the tool should be able to extract values from the AMF messages and validate their content. You need to be able to define validation rules and detect failures. WebLOAD supports getting AMF responses as JavaScript objects and with efficient direct access to specific elements of these objects, can verify their content. Available Building Blocks within WebLOAD provide easy-to-use functions for AMF content verification.

Load execution (scalability): The tool should be able to generate load and simulate multiple users (up to thousands) while running test scenarios. In addition, the tool should be able to use decoded test scenarios and encode them back into the AMF binary format during execution of the load session. WebLOAD keeps the JavaScript representation of the AMF messages optimal, so only responses that you would like to verify will be deserialized into JavaScript objects. All traffic to the application will be in the native binary AMF protocol.

Runtime Measurements: The tool should provide the ability to collect performance measurements, such as response time (the number of seconds it takes to respond to a single HTTP request) and throughput. WebLOAD enables monitoring of runtime statistics and measurements provided by the LiveCycle Data Services and any other server-side components used by the Flex server application (e.g., web applications, databases, etc.). This allows correlation between the application's response time and performance measurements to the server-side components' performance to better identify bottlenecks and the root cause of performance problems.

Maintaining state: Each AMF session initiated by the Flash player is assigned a session ID by the Flex remoting gateway. To accurately simulate registration of numerous different players (or clients) with the server, this session ID, which carries the session's state should be properly carried by all subsequent requests to the server from the client. This is done automatically by WebLOAD by a process called correlation, during the recording of the test script.

Ease of use: Using the load testing tool, it should be easy to create and manage test scenarios. The tool should allow the test engineer to create complex test cases and validation rules without limitations or restrictions. The tool should be easy to configure and operate, which will translate into faster and more rapid performance testing.



Case Study - A Large Cellular Company

A large cellular company was preparing to go-to-market and launch an extremely attractive and exciting new Flex based application. The application would offer a wide range of services and features to their customers. One such service is the downloading of ringtones and games to their customer's cell phones. This Flex application was very likely going to attract hundreds of thousands of users in the near future.

Load testing was done in a dedicated test environment, identical to the production environment. The application's backend architecture included Websphere 6.1, LiveCycle ES and a very large Oracle database.

Their load testing objective was to ensure the application would scale and meet a performance SLA which required the system to handle 1,000 concurrent users with acceptable response time. The download of the initial SWF (Flash) file was allowed to take up to five seconds for broadband connections, and all other requests were expected to perform in under three seconds.

The test was scheduled to run over a period of four to five days.

The test team, which included employees and consultants, chose WebLOAD to perform the load tests on their Flex based application. The main benefits gained by using WebLOAD were:

- **Rapid deployment:** The test scenarios and the test environment were created in less than two days!
- **Developer friendly:** WebLOAD allowed test developers with a basic knowledge of JavaScript to create and execute the tests.
- **Scalability:** WebLOAD simulated the real-world load by creating 1,000 concurrent users.
- **Transaction verification:**
 - Generating virtual users and virtual traffic is only part of the challenge. WebLOAD not only checks whether the load test passed or failed but also supports review and validation of the response for each individual request. This is essential for troubleshooting and fine-tuning.
 - Functional integrity and accuracy: WebLOAD scripts ran basic functionality scenarios and verified that the application behaved correctly under load.

As described at the beginning of this document, load testing and system tuning are extensive, involved processes. The case here was not different. At the beginning of the load testing



process, serious performance problems were encountered. With only 20 virtual clients hitting the server it was throwing Internal Server Errors (HTTP 500).

After some investigations, and using different tuning approaches, the team was able to change the maximum number of connections per host on the LiveCycle Data Server to 100 and the maximum number of total connections to 400. With this configuration change the server performance improved and eighty virtual clients ran successfully with no errors being thrown. But the load time for the service's home page, which contained the SWF file was taking between sixty and ninety seconds, way above the required five second goal.

When the load was increased to 81 virtual clients, the Internal Server Errors came back again, and increasing the two mentioned parameters did not improve performance anymore. At this point the developers went back to look at their code and found a bug in the error handling routine for a certain scenario. When this bug was fixed the download of the home page dropped to six - eight seconds and the download for other pages was one - three seconds. Additional improvements were done and finally the concurrent user goal was reached.

Today this Flex application is live in production and successfully servicing customers. Thanks to this early phase testing project, this large cellular company avoided the penalty of deploying a high-profile application that was not ready. At the end of the day, they avoided damage to their brand and their business.

Summary

Your Flex application performance should be treated as a high priority for your organization. Inadequate performance can damage your company's brand and business overall. No matter how attractive and rich your Flex application is, if it doesn't provide the expected responsiveness and speed, it will not be successful.

Load testing is an important part of the application software development lifecycle for any mission critical web environment. Load testing of Flex applications is even more crucial, given the additional complexity of its architecture and the fact that Flex is a new technology that has not yet accumulated collective knowledge and best practices.

Choosing the right tool for load testing your Flex based applications is a critical component in the success of your load testing projects. RadView's WebLOAD has successfully load tested multiple Flex applications and can ensure the success of your Flex load testing project too.



RadView Contact Information:

North America

RadView Software, Inc.

991 Highway 22 West, Suite 200

Bridgewater, NJ 08807

Phone: 908-526-7756 Fax: 908 864 8099

Toll Free: 1-888-RADVIEW

International:

RadView Software LTD

14 Hamelacha St, Rosh Haayin 48091, Israel

Phone: +972-3-915-7060 Fax: +972-3-915-7683

www.radview.com