



itest.aztecsoft.com

Performance Testing for AJAX-based Applications

Rajendra Gokhale,
Aztecsoft itest

Table of Contents

ABSTRACT	3
INTRODUCTION	4
AJAX APPLICATION VS NORMAL WEB APPLICATION	4
GOOGLE SUGGEST - A CASE STUDY	7
CHALLENGES IN PERFORMANCE TESTING AJAX APPLICATIONS	8
Definition of Performance Goals and Metrics	8
User Modeling	9
Scripting and Load Simulation	10
CONCLUSION	11
REFERENCES	11

ABSTRACT

The AJAX model of development for Web applications has rapidly gained a lot of popularity because of its promise of bringing the richness and responsiveness of desktop applications to the web. AJAX implementations are fundamentally different from other web implementations in two respects - they make asynchronous requests for parts of the web page. Techniques routinely used for performance testing of traditional web applications need to be modified and enhanced to suit the needs of AJAX-based applications. Using Google's "Google Select" service as a case study we examine the unique challenges of carrying out performance testing of AJAX-based applications and offer suggestions for overcoming them.

INTRODUCTION

AJAX (Asynchronous JavaScript and XML) is an approach to web programming that has been enjoying great popularity ever since it was used by Google for many of its applications, notably “*Google Suggest*” and “*Google Maps*”. There has been a lot of discussion about a number of issues related to AJAX including:

- tools and techniques for implementing AJAX (e.g. Ruby on Rails, DWR, Prototype, Sajax, Ajax.net) [1,3]
- business case for using AJAX [2]
- usability of web applications using AJAX [5]
- optimizing network bandwidth utilization using AJAX for application development[6]

Considering that one of the key drivers for the rapid adoption of AJAX has been its promise of superior performance, it is surprising that there has not been much discussion of AJAX-specific performance testing. When we studied this in some detail, we found that AJAX applications indeed present some unique issues and challenges, which we discuss in this paper.

AJAX APPLICATION VS NORMAL WEB APPLICATION

A typical web application works as follows:

- User supplies input to browser (e.g. types in a URL, clicks on a hyperlink, submits a form)
- Browser sends a request for the URL to the server
- Web server responds with a page
- Browser sends more requests for embedded objects (e.g. images)
- Browser renders the page (including embedded objects)
- Browser waits for user’s next input and then goes back to the first step.

The key points to note here are:

- The browser issues requests for entire pages and the entire page gets refreshed as a result of this action
- These requests occur as a direct consequence of user actions.

In contrast to this, Ajax applications make a number of *asynchronous* web requests for *parts* of the current webpage. These requests are issued by a piece of client-side code that is executed in the browser context. This client-side code is usually implemented in JavaScript and is called *the AJAX engine*.

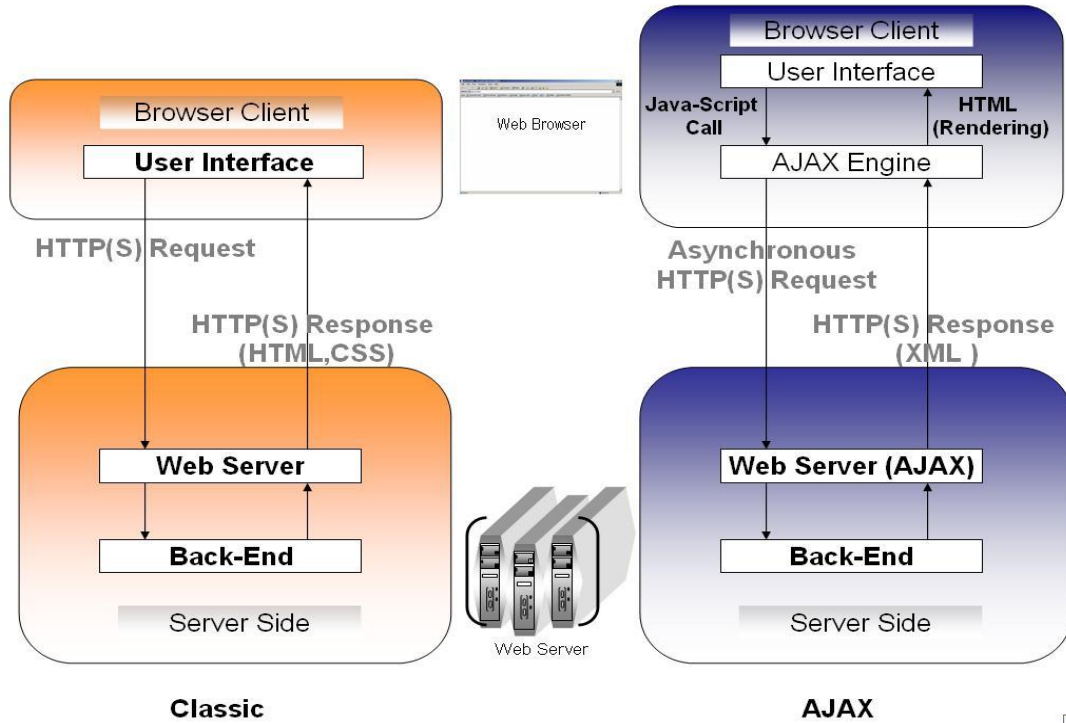


Fig. 1: Comparison of traditional web application with an AJAX-based web application

The following table summarizes the key differences between traditional and AJAX web applications:

Traditional Application	AJAX application
Requests sent as a direct response to user actions	Requests happen <i>asynchronously</i> at intervals determined by the AJAX engine
Requests are typically in the form of forms that use the GET or POST method	Application can use any form of request mechanism supported by the remote server (e.g. XMLHttpRequest)
Browser-server interaction involves requests for <i>complete</i> pages	Requests for complete web-pages are typically interspersed with <i>many</i> requests that serve to update <i>parts</i> of the already loaded page
The response consists of an <i>entire</i> new page	The response is a piece of data that is interpreted by the AJAX engine to update the current DOM ¹

Table 1: Traditional Vs AJAX based web applications

Done right, the AJAX approach can yield a number of important advantages:

- Since the response does not contain the entire page, a smaller amount of data gets transferred across the network thus resulting in better network utilization.
- Instead of reloading an entire page, AJAX applications update only parts of the page, thus improving the responsiveness of the application.

¹ DOM: Document Object Model. See www.w3.org/DOM.

On the other hand, a badly designed/tuned AJAX application can have a counter-productive effect. For example, an AJAX engine can send *too many* AJAX requests and increase, rather than reduce, the load on the network!

It is therefore imperative that an AJAX application be put through a thorough performance testing cycle before it is released for general use. We have selected “*Google Suggest*” as a real-world example so that our discussions retain a practical flavor.

GOOGLE SUGGEST - A CASE STUDY

“Google Suggest” (<http://www.google.com/webhp?complete=1>) is a new interface to the well known Google Search engine (referred to in this paper as “Vanilla Google”). As one starts typing in the “Google Suggest” search textbox, the application guesses what the user is *intending to type* and offers suggestions in real time, *even as the user is typing in the search box*. For example: as one types “bass,” “Google Suggest” might offer a list of completions that include “bass shoes” or “bass guitar”. Similarly, when a user has typed in a part of a word such as “progr” “Google Suggest” might offer completions such as “programs,” “progress,” or “progressive.” A user can choose any of the suggestions by scrolling up or down the list with the arrow keys or mouse. The result is that the user gets a desktop-like feel even when interacting with a web application!

“Google Suggest Dissected” [4] contains a detailed discussion of how this functionality is implemented but we list below some key implementation details that are most relevant for our purpose, as we will see in Section ____.:

- Asynchronous requests for phrase completions are sent to the server at regular time intervals.
- The time interval between these requests is determined dynamically and is a function of the latency observed by the specific client.
- No request is sent if the user has not typed anything since the last request was sent.
- The results obtained dynamically are cached. This comes in useful when the user erases something he has typed, since the cached results can be reused thus saving some unnecessary requests to the server.

CHALLENGES IN PERFORMANCE TESTING AJAX APPLICATIONS

Definition of Performance Goals and Metrics

The goals, and therefore the metrics, for the performance testing of AJAX applications are not the same as for traditional applications. Two of the most widely used traditional measures for web applications, “page views per unit time” and “clicks per minute”, are meaningless in an AJAX context. A user could theoretically be looking at the same page for hours on end without ever clicking any URL, submitting any form or refreshing the page. Although the user may have viewed just one page, he might have generated tens of thousands of requests and may actually have been glued to the screen during this entire period! One example of such an application might be a dashboard for monitoring a chemical plant. The dashboard could get updated at regular intervals without generating even a single page view over an extended period of time.

On the other hand, some performance goals (and metrics) that are relevant for AJAX applications are largely or completely inapplicable for non-Ajax applications.

- **Optimization of the AJAX engine:** While traditional applications need to make sure that all components at the server are properly designed, tuned and configured, in the AJAX application the AJAX engine acts like an intermediate client-side server. When a Google Select user, for example, enters a string in the search-box, the AJAX engine pre-fetches a certain number of words and phrases that are consistent with the text that the user has already entered. This allows the user to select a search item from the options suggested by Google, thus saving typing effort and enhancing the user experience. Great idea, but it introduces a few problems. The AJAX engine could make too many requests, and choke up the network; and/or the web-server; or it could make too few requests and lag behind the user (especially if the user is a fast typist). It would therefore seem reasonable to vary the frequency of AJAX requests in “Google Suggest” as a function of the network speed and the user’s typing abilities, in other words to optimize the AJAX engine. It must be noted that, while there are numerous dimensions that optimization may address (e.g. network utilization, number of computations on the client or server, responsiveness of application, etc.), whatever criteria are chosen, optimization of the AJAX engine is an important goal for any performance testing efforts for AJAX applications. In the context of “Google Suggest”, the performance tester could ask:

“What is the function that currently determines the frequency at which the AJAX engine makes an AJAX request, where the two inputs are (1) observed server-response times and (2) observed user typing speeds?”.

For the sake of optimization, the next question would be how to optimize that function. The answer of course would depend on the application.

- **Measuring the Cross-platform performance of the AJAX application:** In recent years, performance tuning of browser-based applications has been at the server end, where most of the heavy lifting occurs. However, since the AJAX engine resides on the client, performance testing should be done not only to enable optimizing of the AJAX engine, but also to test the compatibility matrix of browsers, operating systems, client hardware, network topologies, and network speeds. For example, memory leaks in some browsers can cause AJAX applications to fail. In the “Google Suggest” context, one would need to carry out tests described in the previous bullet (AJAX engine optimization) on the entire matrix. In particular, one would need to make sure that the application behaves reasonably for different type of network speeds (LAN, Dialup, etc).

User Modeling

User modeling for AJAX applications is very different from that for normal applications. In normal applications one essentially records various scenarios, modifies them for use in a Load Test scenario (by actions such as randomizing the next request), and adds “User Think Times”. While this is undoubtedly an oversimplification, the point we wish to make here is that the only real impact of a user’s action is on the frequency with which the requests are sent. That is not the case with AJAX applications. For example, even the *contents* of AJAX requests sent by “*Google Select*” will *differ* depending on whether the user is a slow or a fast typist! Other factors that impact this include:

- network speed
- speed of the browser’s rendering capabilities
- nature of search strings used

Clearly, one needs to a factor in many more variables when modeling the load on an AJAX application, which makes the process of load-modeling much more complex. We feel that one important element of performance test planning is the method for simulating this behavior. Should one adopt a simplistic approach and generate (say) an AJAX request at statically determined intervals or should one go all the way and try to accurately model a real-life scenario? This is not an easy question, depending as it does upon the nature of the application. This question also crops up in performance testing for normal applications but is much more acute here.

Google	Google Suggest
What is the average number of searches performed by a typical user during the course of a day?	What is the average number of searches performed by a typical user during the course of a day? What is the average length of search strings? <i>(for longer strings the # intermediate Ajax requests would be more)</i> What is the average typing speed of a typical user? <i>E.g.:</i> 10 % Fast (x words/sec) 70 % Medium (y words/sec) 30 % Slow (z words/sec) Response time distribution. ²

Table 2: User Modeling: Google Vs Google Suggest

² The Ajax engine dynamically determines the frequency of intermediate requests based on its observed response times. When response times are higher, it lowers the frequency of responses, thereby reducing the overall number of requests sent in a given duration. It is therefore important to simulate this response time distribution accurately. Note that this is different from the Network Topology modeling carried out during routine performance testing.

Scripting and Load Simulation

For most applications (apart from exceptions such as applications that use applets), load consists of a series of http requests that can often be statically determined at scripting time or easily configured at run-time. Things aren't as simple when it comes to scripting for AJAX applications.

- Script designers need to gain a deep understanding of the AJAX implementation: While carrying out performance testing of traditional web applications, one can largely ignore the client-side code generated since it is primarily used for client-side rendering and validation and has no impact on the server performance. This is not the case in AJAX applications - in an AJAX application; client-side code has to be understood by the test team since that code plays an important role in determining the load that is generated. For "Google Select", the AJAX client consists of over 60 lines of compressed JavaScript, most of which needs to be faithfully reproduced in the test scripts. It may be noted here that the compressed JavaScript format is a format that is designed to be very compact and as a consequence is thoroughly obfuscated for human readers.
- Simulation of the AJAX engine involves hard design choices: An AJAX application determines what requests to make and when to make them, based on the model embedded in the AJAX engine. This model needs to be reproduced when performance testing AJAX applications. There are two broad approaches that can be adopted - one can either embed the AJAX engine code into the test script in the form of a plug-in (the "heavy" model) or simulate it using the Load Testing tool's scripting functionality (the "light" model). The test scripter has to decide which approach to take based on a number of factors:
 - Load Testing is carried out using a farm of machines and every machine in this farm is used to simulate many "virtual users". Load Testing tools achieve this by spawning a number of threads on a single machine. Each thread simulates the actions of one "virtual user". This approach of using multiple threads to simulate multiple users on a single machine ordinarily works since each of these threads is quite lightweight in nature. However that may not be true in case we used the "heavy" model for our AJAX test scripts and this may result in the load testing client itself becoming a bottleneck. Thus the approach of embedding the actual AJAX engine into the Load Testing script should be considered only in situations where this engine is very light-weight and does not make too many demands on the machine it is hosted on.
 - If the nature, mix and frequency of AJAX requests can be easily modeled, it may make more sense to attempt to simulate the AJAX engine "by hand". Since the AJAX engine can actually be viewed in the browser it is conceivable that even a third party could be able to achieve this (for example see "Google Suggest Dissected" [4] that describes how Chris Justus reverse engineered "Google Suggest"). In fact, in real world projects, such reverse-engineering may be unnecessary since the performance testing team would be working closely with the development team and would have access to knowledge about the AJAX engine's implementation.
- Debugging test scripts is harder: One question that a tester has to consider is whether he is doing a simulation correctly. During the scripting phase, testers make use of a feature that most load testing tools allow - that of allowing their users to view the responses received from the server in the form of rendered HTML. However this is not feasible for AJAX applications since the response to an AJAX request cannot be natively rendered by the Load Test tool. More often than not, this response is coded data that has to be interpreted by the AJAX engine before acting upon it. As a result, the Load Testing tool is unable to visually simulate the results of the response as it can do for non-AJAX applications. Debugging test scripts for AJAX applications is therefore much harder.

We attempted to gauge the degree of difficulty involved in creating test scripts for AJAX applications by actually creating performance testing scripts for both versions of the Google search engine using the

load testing tool “Jakarta Jmeter” [9]. Developing the script for “Vanilla Google” was fairly straightforward - the script has to read the search strings from a file and fire random strings at well-defined intervals. In contrast, implementing the corresponding scripts for “Google Select” was much harder, primarily because it involved re-implementing the complex algorithms for determining request frequencies. Developing the test scripts for “Google Select” involved three times as much effort as that required for “Vanilla Google”.

CONCLUSION

Our overall conclusion is that, although performance testing for AJAX applications is significantly more challenging than that for traditional applications, it is certainly practical to attempt. For this to be successful there is a need for identifying the special challenges involved and incorporating solutions in a well-designed methodology. We hope that some of the challenges highlighted in this paper may help in these efforts.

REFERENCES

1. <http://www.adaptivepath.com/publications/essays/archives/000385.php>. The original article by Jesse James Garrett that popularized the term AJAX.
2. <http://www.developer.com/java/other/article.php/3554271>. Good discussion of the benefits of using AJAX, including the business case.
3. http://www.onlamp.com/pub/a/onlamp/2005/06/09/rails_ajax.html. Describes “Ruby on Rails”, a framework popular among AJAX developers.
4. <http://serversideguy.blogspot.com/2004/12/google-suggest-dissected.html>. “Google Suggest Dissected” - a good discussion on how “Google Select” is implemented (at the browser side).
5. <http://www.baekdal.com/articles/Usability/XMLHttpRequest-guidelines> - Thomas Baekdal’s usability guidelines for XMLHttpRequest.
6. <http://www.webperformanceinc.com/library/reports/AjaxBandwidth/index.html> - “Using AJAX to improve the Bandwidth Performance of Web Applications”, by Christopher L Merrill of Web Performance, Inc., published January 15, 2006..
7. <http://www.techweb.com/wire/showArticle.jhtml?articleID=165702733>
8. <http://jakarta.apache.org/jmeter>. The official site for the load testing tool “Jakarta Jmeter”.