

MODELING AJAX APPLICATION PERFORMANCE

Clinton W. Smullen III
Stephanie A. Smullen
The University of Tennessee at Chattanooga
615 McCallie Avenue, Chattanooga, TN 37415
U.S.A.
Clinton-Smullen@utc.edu
Stephanie-Smullen@utc.edu

ABSTRACT

An experimental study of the comparative performance of a real-life HTML application and an AJAX application that implement the same user interface was done. Models of these applications were developed and experimental data was collected on the performance of each when presented with the same tasks. Performance measures were computed for the HTML application and for two versions of the AJAX application. The effect of disabling client caching was also studied. AJAX provided significant performance improvements (40-70%) under certain circumstances, although in a worst case scenario it performed worse (10-15%) than the traditional HTML application.

KEY WORDS

AJAX, performance, modeling, HTML

1. Introduction

AJAX (Asynchronous JavaScript And XML) is a name applied to a set of technologies [1] designed to improve web application responsiveness, including HTML and CSS, Dynamic HTML, client-side scripting (in either JavaScript or Java), dynamic displays using the DOM model, data exchange using XML, and synchronous/asynchronous data retrieval using XMLHttpRequest. A good reference is Crane [2]. Mature commercial examples of the use of AJAX include Google Maps and Gmail. The use of AJAX in web programming is becoming more widespread as newer versions of browsers support XMLHttpRequest objects. However, few studies have been published about the performance of actual AJAX applications.

The commonly cited AJAX performance evaluation case study is the one reported by White [3,4]. White constructed two sample applications for a company sales site, one using HTML and the other using AJAX. He had five users enter specified responses to complete a sale for an existing customer and to add a new customer and enter a sale. He collected the bytes transferred and the time

consumed by the tasks. This AJAX application transferred on average 27% of the bytes that the traditional HTML application transferred, resulting in a 73% performance increase. Performance increases for the AJAX application were also observed in the task times taken by the users to accomplish the work. However, the effects of users' skill levels and training (which could greatly affect the reported outcomes) were not assessed in this report. This is especially true since the two applications do not implement the same user interface. Nevertheless, the AJAX application required less bytes to carry out the tasks than did the HTML application and the users accomplished the assigned work in less time. These time savings can be directly translated into personnel cost savings.

To gain insight into the performance effects of AJAX, this report studies an HTML application and an AJAX application which both implement the same user interface with the same "look and feel".

2. The Application

The application studied in this paper is an existing application that supplies real-time class information extracted from a university student information system (SIS). The user specifies one or more selection criteria (such as department, course/section, meeting days, start time/end time, location, instructor, open/closed) and the application returns a list of courses meeting the specified criteria and additional information about each of the courses (including the title and current enrollment). This is a production application, used daily by students and faculty, not a "test" application. The web server is Apache, and the application uses PHP 5.05 and custom database code to connect with the legacy SIS database. All pages returned are validated XHTML 1.1.

2.1 The HTML Application

The initial page loaded by a user contains the HTML form used to prepare a query. There is a significant amount of "branding" overhead on this page; all of the University's

pages use the same layout, navigation items and stylesheet. These common elements consist of two graphical images, a CSS stylesheet, and JavaScript supporting the common page navigation links, and total 15573 bytes. These elements are linked to the HTML page and are static. For most browsers, they are downloaded once and cached, rather than being loaded with each query and response.

A typical user would first load the HTML page containing the query form (28KB) and the common elements (15.2KB). The user prepares a query and submits the query to a server application. The server application queries the SIS. The data extracted from the SIS is formatted as XML. The server process then reads the XML data and applies an XSLT transform to produce XHTML. The web server returns the XHTML as the response to the client.

The page returned as a response to a query links to the common elements described above, contains the HTML formatted list of courses in answer to the query (or a message if no results are produced), and also contains the HTML form needed to make another query. As a result, even a query that produces no results has a response page of about 28KB (plus the linked common elements).

2.2 The AJAX Application

An AJAX version of the above HTML application was coded. The AJAX version uses JavaScript to run the user interface and the XML handler. Care was taken to create the same “look and feel” for the AJAX version as is used in the HTML version. Both the AJAX and the HTML applications appear the same to the user at the presentation level, so the presentation page structure is the same, as are the graphics and common navigation elements.

The Sarissa [4] libraries (version 0.9.6.1) were used to code the AJAX application. Sarissa is an open-source cross-browser set of libraries based on ECMAScript for developing AJAX applications. It is representative of the types of implementation libraries used for production AJAX. The code and libraries downloaded to the client to implement the AJAX interface for this application total 57KB.

For this investigation, the AJAX application uses only synchronous AJAX XMLHttpRequest calls, for several reasons. Choosing courses is inherently a serial process. The next course selection often depends on the previous selections made. Work in progress will assess asynchronous AJAX performance, and this will appear in a future paper. Since the AJAX application reproduces the same “look and feel” as the HTML application, the use of asynchronous calls was not needed. This application does not exploit AJAX capabilities to enhance the user experience, preload data, or use event-triggered

processing. Hence this application represents in some sense a worst case for AJAX – all of the overhead but few of the advantages other than partial page refresh.

The existing server application was used for the AJAX application. The central IT staff was extremely reluctant to modify a production system for this experiment. Since the query results from the server application are available in XML form, the existing production server application responds to an AJAX XMLHttpRequest with the existing XML output; it skips the XSLT transform and forwards the XML to the client. Specifically, no special XML was created for this experiment. The AJAX application fetches the XML data produced by the production server process, which is then processed in the AJAX client for display. The decision to not require modifications to the existing server process made the data collection possible, but also required additional analysis, described below.

3. Modeling the applications

When a standard web application is contacted by a browser, an HTML page is loaded into the browser, including any graphical elements, stylesheets, or other linked elements. The client interacts with the user interface presented on the HTML page (such as by clicking a link or submitting a form) which sends an HTTP request message to the web server. The web server sends a new HTML page to the browser. This page replaces the previous page in the browser.

When an AJAX application is contacted by a browser, an HTML page is loaded into the browser. This page contains the JavaScript needed to run the user interface and to issue XMLHttpRequest calls, the XML handler to format and present the results, and any other elements needed for the user interface, such as HTML code and stylesheets. The client interacts with the user interface presented on the browser page (such as by clicking a link, submitting a form, or triggering an event). The JavaScript code handles the user interface event, usually generating an XMLHttpRequest message to the web server. The web server sends XML data to the browser. This XML data is handled by the JavaScript code and presented to the user. Only the portion of the browser page needed to display the results is refreshed.

Consider a task consisting of a sequence of operations carried out through an HTML web application. Accomplishing the task requires loading an initial page, and then successively loading pages 1 to n , where n is the number of steps needed to accomplish the task. For an AJAX application, accomplishing the task requires loading an initial page, and then successively handling responses 1 to m , where m is the number of steps needed to accomplish the task. For the application and two implementations discussed above, $n=m$ since both implementations provide the same user interface. Define

the following notation.

H	Size in bytes of the initial elements downloaded by the HTML web application
P_i	Size in bytes of the i th HTML page loaded by the HTML web application
P_a	Size in bytes of the average of the n P_i values
A	Size in bytes of the initial elements downloaded by the AJAX web application
U_i	Size in bytes of the i th AJAX response handled by the AJAX web application
U_a	Size in bytes of the average of the n U_i values

The total bytes required to accomplish the task for each implementation is as follows.

$$\text{HTML total} = H + (P_1 + P_2 + \dots + P_n) = H + n \cdot P_a$$

$$\text{AJAX total} = A + (U_1 + U_2 + \dots + U_n) = A + n \cdot U_a$$

For most AJAX implementations, $H < A$, $P_a > U_a$, and it is likely that $P_i > U_i$ for all i . If $H > A$ and all $P_i > U_i$ then the AJAX implementation has an initial download time better than the HTML application, and the download time for each response is better as well. The AJAX implementation is clearly preferred over the HTML implementation. If $H = A$ and the $P_i = U_i$ then the download times are equal. The greater effort and complexity required to code and maintain an AJAX implementation as compared to an HTML implementation would lead one to select the HTML implementation, unless the AJAX user interface presented advantages to the user. If $H < A$ and the $P_i < U_i$ then the HTML application is clearly preferred over the AJAX implementation based on download times, unless the AJAX user interface presented significant advantages to the user.

Since both of the implementations discussed in section 2 implement the same user interface, the cognitive requirements on the user to view a page or response and to make a new request using the two interfaces are the same. If asynchronous XMLHttpRequest calls were supported, or if the AJAX user interface were significantly improved over the HTML user interface then for a particular application the user could process a response and make a new request using the AJAX interface faster than when using the HTML interface.

The total time needed to accomplish the task is the sum of the time needed to download each of the pages or responses plus the cognitive time needed for the user to view the page or response and make a new request. When using a fast network connection such as a local area network, the download time for a web application is typically not readily visible to the user. For a moderate speed connection, it becomes visible only for large results, and for a dialup connection, the download time is always of concern. In these cases, the value of using

asynchronous requests is diminished, since the slowness of the communications channel may serialize the responses. Downloading multiple responses concurrently over a slow link may actually cause the first of the responses to appear much more slowly than it would when using synchronous requests, thereby increasing the latency of the user interface.

4. Data Collection

A Firefox browser (v1.5.0.1) was used to submit varied queries and to view the response pages. To collect data about the response pages, the “View Page Source” was used. A copy of the page source was supplied to a separate utility to count the bytes in the response since the Firefox “loaded”, “View Document Size”, and “Information” items proved to be unreliable in reporting accurate byte counts.

The initial load of the HTML application requires 43.2KB overhead (15.2KB for the common elements - graphics, CSS, navigation JavaScript, and 28KB for the HTML page with the form); hence $H = 43.2\text{KB}$. Thereafter, each response file is a complete XHTML page, containing the results of the query, the HTML presentation, and the form. As mentioned before, the common elements need to be loaded only once if the browser supports caching.

The initial load of the AJAX application requires 100.2KB overhead (15.2KB common elements - graphics, CSS, navigation JavaScript, 28KB for the HTML presentation and form, and 57KB AJAX JavaScript code). Hence $A = 100.2\text{KB}$. All of these elements need to be loaded only once if the browser supports caching. This initial load could be made smaller if the AJAX application were not reproducing the graphics and look-and-feel of the existing HTML application. After loading the AJAX application, each response is an XML file containing the query result data and other information produced by the server application.

Response bytes	HTML	AJAX	AJAX opt
Min	28,534	8,034	894
Max	1,924,502	1,121,593	1,114,453
Mean	134,342	69,634	62,494
Median	73,587	34,243	25,915
Total	14,777,588	7,659,740	6,874,340

Table 1. Response bytes

One-hundred and ten distinct queries were posed using the production HTML form and also the AJAX form. The number of bytes returned for each query was recorded for the HTML form (the P_i values) and for the AJAX form (the U_i values). The size of each response does not include the common elements loaded from the client cache. The P_i values ranged from 28KB, a query producing no courses, to 1.9MB, a query listing all

courses for the spring semester. The U_i values ranged from 7.8KB to 1.1MB. A summary of these results is shown in Table 1.

As previously noted, the XML data is produced by the standard production process. This process was not modified to optimize the use of AJAX. It includes (encoded as XML data) 7140 bytes of information to allow the XSLT transform to create the query form for the HTML response page. Since the AJAX application already has the form (it is not loading an entire page as HTML does, only refreshing a portion of the screen with the query results) this extra information is unneeded overhead. If a special server process were written to service AJAX requests, the size of each AJAX response would be (at least) 7140 bytes less. The column labeled "AJAX opt" represents this reduction in the size of the AJAX responses.

The effects of using a thin-client were investigated using Microsoft Terminal Services for Windows Server 2003, utilizing RDP 5.2 with a 1280 by 1024 monitor and 24-bit color. A network monitoring package was used to track the traffic to and from the thin client while four separate queries, matching four of those used to prepare Table 1, were entered and viewed. The remote server used the Firefox browser, which used the HTML application on the web server. Hence the load on the web server was identical to that when the HTML application was used by the client to directly contact the web server, and the bytes downloaded to the Windows 2003 server from the web server were the same as when the HTML application was contacted directly by a user. Only the traffic to and from the thin client was monitored. These results are compared in Table 2.

Bytes transferred	HTML	XML	RDP
Query 1	28,543	8,036	11,783
Query 2	237,848	127,023	84,733
Query 3	283,910	156,295	98,352
Query 4	1,924,502	1,121,593	483,230

Table 2. Comparison of Thin Client in Bytes Transferred

The thin client generated fewer bytes transferred to the end client than either the HTML or the AJAX application, presumably due to the on-the-fly compression performed on the data flowing over the link. The end client used to access the HTML application and the AJAX application is a modern web browser, widely available and running on almost all platforms. The end client used for thin client access is a specialized program that must be licensed to and installed on the client computer. In addition to the client software, a remote server is needed to host the remote session used by the client. The bytes transferred between the remote server and the web server are the same as previously documented for the HTML and the AJAX applications. The only savings occur in the

number of bytes transferred to the end client. While the number of bytes transferred over the end user connection is less, the use of thin clients must be viewed as a special purpose solution.

If reducing the number of bytes transferred is the ultimate goal, then having the server compress the data and transmit compressed data to the client, which then decompresses the download to present the result is by far more productive. Query 4 in Table 2 is the "list all courses" query, and the size of the HTML response page is 1,924,502 bytes. When compressed using ZipGenius 1.4, the size of this page is 130,420 bytes, a reduction by a factor of almost 14.8. When the corresponding XML response (1,121,593 bytes) is compressed, the result is 84,508 bytes, a reduction by a factor of 13.3

5. Analysis

The number of bytes transferred to complete a set of queries is a measure of the network impact of the application. This also can be used to predict the latency of the response due to network transfer. While most of the users of the production application studied connect over the university's high-speed network, a significant number do connect from home using dial-up connections.

Table 3 presents the total bytes transferred to complete all 110 queries. The "Caching" column assumes the overhead elements (15.2KB for HTML, 100.2KB for AJAX) are downloaded only once, and thereafter retrieved from the browser cache. To measure the worst-case effects of using a client that does not support caching, that has caching disabled, or that does not retain the common elements from one use to the next, the column "No Caching" assumes the overhead elements must be downloaded with each response. This would also be the result of 110 different users each posing one query using browsers that do not have any of the common elements cached.

Total bytes	Caching	No Caching
HTML	14,793,161	16,490,618
AJAX	7,762,345	18,946,290
AJAX opt	6,976,945	18,160,890

Table 3. Total bytes downloaded bytes for 110 queries

The AJAX application requires significantly fewer bytes than does the HTML application if caching is used. However, if caching is not used, the AJAX application actually requires 10-15% more bytes than does the HTML application. This has implications as many users migrate to mobile devices such as PDAs and cell phones, some of which do not support large caches. Few users of the production application need 110 queries

to accomplish their work. A more typical set of queries would consist of about seven queries used to create a standard five course semester schedule. The average response size (P_a and U_a) values from Table 1 were assumed for each of these seven queries. A user session downloaded the initial elements, H and A, and seven times the average response sizes. Table 4 displays the results, and illustrates again the benefits of using AJAX when caching is available.

Total bytes	Caching	No Caching
HTML	955,967	1,049,405
AJAX	590,043	1,205,673
AJAX opt	540,063	1,155,693

Table 4. Total bytes downloaded for 7 average queries

Following White [4], the performance increase is calculated as follows.

$$(\text{HTML total size} - \text{AJAX total size}) / (\text{HTML total size}) * 100\%$$

This gives a percentage improvement when AJAX is used compared to HTML. A larger performance increase value is better, although any positive value represents a performance increase due to the use of AJAX. A performance increase of 48% means that the savings achieved by using AJAX as a percentage of the HTML size is 48%. Table 5 shows the performance improvement for the two cases discussed above, 110 queries and 7 queries. When caching is used, AJAX provides a significant performance improvement, while when caching is not available, there is a performance penalty (the negative values indicate that AJAX required more bytes than did HTML).

	Caching	No Caching
110 Queries		
AJAX	47.53%	-14.89%
AJAX opt	52.84%	-10.13%
7 Queries		
AJAX	38.28%	-14.89%
AJAX opt	43.51%	-10.13%

Table 5. Performance Increase Comparison

The performance increase due to the use of AJAX is nonlinear in nature, and varies with the size of the HTML response. Figure 1 plots the performance increase in response size for each of the 110 responses in the experiment versus the HTML size; the overhead is not accounted for in this graph. The smaller responses have a much greater performance increase than do the larger responses. The largest responses provide a performance improvement of about 42%.

Additional performance gains result from the reduction in the load placed on the server. The server load reduction

for the AJAX application is due to several factors. No XSLT processing is required for the AJAX application. For the HTML process, the number of bytes handled by the server process is proportional to the sum of the XML data and the HTML response page, since the process reads the XML data, processes the XSLT transform to produce the HTML response page, and the web server then sends the resulting HTML response page to the client. For the AJAX process, no XSLT processing is done; the XML data is simply forwarded to the client by the web server. This means the AJAX application results in significantly fewer bytes being handled by the server. Fewer bytes also means faster connection termination, which could increase the number of connections serviced. No estimates of the exact savings in server execution for this experiment are available.

A visual evaluation of the graph of Performance Increase versus HTML size (Figure 1) shows that it appears to exhibit exponential decrease. To test this observation, the quantity Byte Transfer Ratio is found by subtracting the performance increase percent from 100: Byte Transfer Ratio = 100 – performance increase. The Byte Transfer Ratio is thus (AJAX size/ HTML size)*100, and represents the reduction achieved by using AJAX when compared to the HTML application. A Byte Transfer Ratio of 42% means that the size of the AJAX transfer is 42% of the size of the HTML transfer. A small Byte Transfer Ratio is better; it means the AJAX application is performing more efficiently than the HTML application. However any value less than 100% represents a reduction in bytes transferred when using AJAX.

To determine if the Byte Transfer Ratio is distributed exponentially, the 110 values U_i/P_i for the AJAX application were used. Given P_i , an estimate of U_i/P_i is given by the function $\beta(1 - e^{-\lambda P_i})$. A nonlinear fit to the function $\beta(1 - e^{-\lambda P_i})$ was performed using the SAS 9.1 for Windows (SAS Institute, Cary, NC) NLIN procedure. The SAS NLIN procedure estimates for the parameters β and λ were $\beta=0.5458$ and $\lambda=0.0264$. A Kolmogorov-Smirnov test revealed that the predicted distribution was a good fit for the observed distribution. Both the observed and the predicted Byte Transfer Ratio values for the 110 queries are shown in Figure 2.

The actual Byte Transfer Ratio values level off at about 58%, so the Performance Increase is limited to 42%. The predicted Byte Transfer Ratio values level off at about 55% under this model, giving a predicted Performance Increase of 45%.

6. Conclusions and Future Work

Very few performance analyses of AJAX applications appear in the existing literature or online. Additional work is needed in this area. Do other AJAX applications exhibit performance characteristics similar to the one

described in this report? Do other AJAX applications have a Byte Transfer Ratio which can be fitted by an exponential distribution? Why does this happen? Do other AJAX applications, especially more “light-weight” ones, exhibit a performance improvement curve shaped similarly to the one in this study? What are the performance effects of using asynchronous AJAX that would allow several concurrent queries? It is often stated that using AJAX can reduce latency and improve the user interface response by prefetching data. For an application in which it is difficult to predict the next user query, at what point does prefetching actually decrease performance?

An understanding of the performance characteristics of AJAX for real-life applications is important for web developers. Developers need to learn to develop both of these types of applications, and to discern when each is the appropriate solution to the problem being solved.

References

- [1] L. Paulson, Building rich web applications with AJAX, *IEEE Computer*, 38(10), 2005, 14-17.
- [2] Crane, Pascarello, James, *Ajax in Action* (Greenwich, CT: Manning Publishing Company, 2005).
- [3] AjaxInfo.com, Measuring the Benefits of Ajax, <http://www.ajaxinfo.com/ucst/index.html> retrieved March, 2006.
- [4] A. White, Measuring the Benefits of Ajax, <http://www.developer.com/xml/article.php/3554271> retrieved March, 2006.
- [5] Sarissa, <http://sarissa.sourceforge.net/> retrieved March 2005. The Sarissa project page is <http://sourceforge.net/projects/sarissa>.

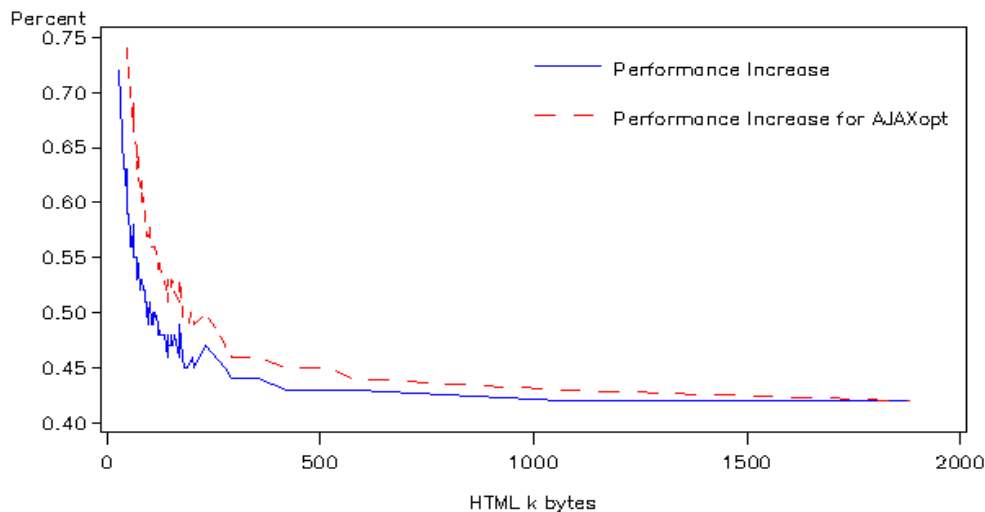


Figure 1. Performance Increase vs HTML size

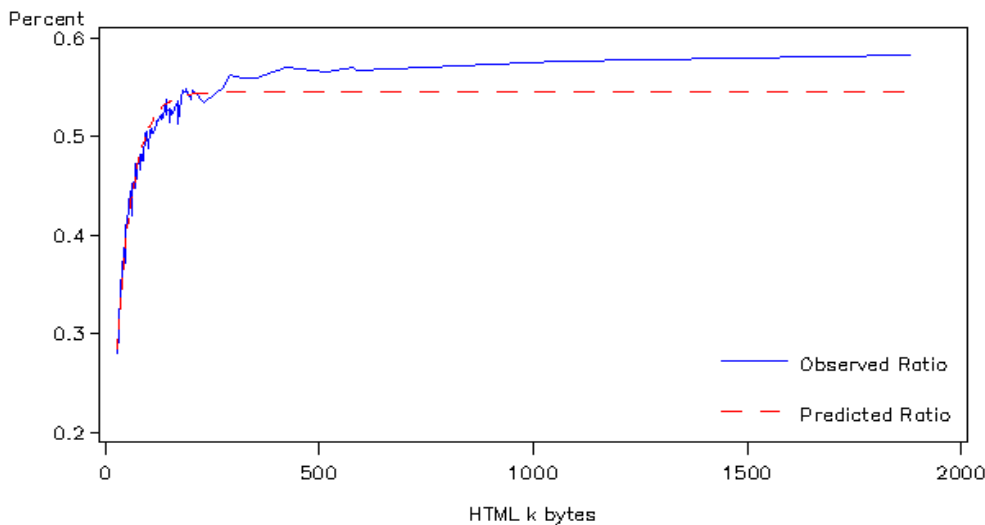


Figure 2. Byte Transfer Ratio predicted and actual