

# AJAX Application Server Performance

Clinton W. Smullen III  
*The University of Tennessee at Chattanooga*  
*Clinton-Smullen@utc.edu*

Stephanie A. Smullen  
*The University of Tennessee at Chattanooga*  
*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. Data was collected on the performance of each when presented with the same tasks. Response size and service time performance measures were computed for the applications. AJAX provided significant performance improvements in response size (56%), thereby reducing bandwidth requirements. AJAX provided a mean service time improvement of approximately 16%.

## I. 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; AjaxPatterns [3] categorizes over fifty websites using AJAX applications. 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.

One commonly cited AJAX performance evaluation case study is the one reported by White [4, 5]. 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".

## II. THE APPLICATION

In [6], Smullen and Smullen compared the client-side performance of a real-life HTML application and an AJAX application that implemented the same user interface. Experimental data was collected on the performance of each when presented with the same set of one-hundred ten tasks. Performance measures were computed for the HTML application and for the AJAX application. The current study extends this work by collecting data on a statistically significant sample size and includes server performance results.

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). The application uses a three tier model; the client communicates with the web server, which communicates with the database server. It 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.

### A. 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 style sheet. These common elements consist of two graphical images, a CSS style sheet, 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 (27KB) 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 27KB (plus the linked common elements).

### B. THE AJAX APPLICATION

An AJAX version of the 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 [7] 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. The AJAX application fetches the XML data produced by the production server process, which is then processed in the AJAX client for display.

### III. DATA COLLECTION

When a standard web application is contacted by a browser, an HTML page is loaded into the browser,

including any graphical elements, style sheets, 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 style sheets. 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.

The initial load of the HTML application requires 42.2KB overhead (15.2KB for the common elements - graphics, CSS, navigation JavaScript, and 27KB for the HTML page with the form). Thereafter, each response file is a complete XHTML page, containing the results of the query, the HTML presentation, and the form. The common elements need to be loaded only once if the browser supports caching.

The initial load of the AJAX application requires 99.2KB overhead (15.2KB common elements - graphics, CSS, navigation JavaScript, 27KB for the HTML presentation and form, and 57KB AJAX JavaScript code). 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

TABLE I  
RESPONSE BYTES SUMMARY

Response bytes	HTML	AJAX
<b>N</b>	13,260	13,260
<b>Min</b>	27,650	7,746
<b>Max</b>	2,017,914	1,196,618
<b>Mean</b>	129,828	68,234
<b>Median</b>	63,996	28,996
<b>Total</b>	1,721,523,273	904,776,578

All data was collected by querying the production servers. Thirteen-thousand two-hundred and sixty queries were posed using the production HTML form, and also posed using the AJAX form. The number of bytes returned for each query was recorded for the HTML form and for the AJAX form. The size of each response does not

include the common elements loaded from the client cache; the effects of caching these common elements were studied in [6]. The HTML values ranged from 27KB, a query producing no courses, to 1.9MB, a query listing all courses for the fall semester. The AJAX response byte values ranged from 7.6KB to 1.1MB. A summary of these results is shown in Table I.

The number of bytes returned for a given query is deterministic, and does not vary with the load on the servers. To measure the effects of the use of AJAX on the servers (and hence the apparent responsiveness of the application to the user), the time to service each request was recorded from the Apache web server log. An examination of a small sample of these times collected by two clients showed that they had wide variations due to the exogenous load applied to the servers. To reduce the variance of the service times, repetitions of a set of 121 queries were performed for both the HTML and the AJAX applications. A summary of the response times in seconds is presented in Table II.

TABLE II  
RESPONSE SECONDS SUMMARY

Response seconds	HTML	AJAX
<b>N</b>	13,260	13,260
<b>Min</b>	0.308	0.254
<b>Max</b>	37.584	34.421
<b>Mean</b>	1.527	1.289
<b>Median</b>	0.887	0.731
<b>Total</b>	20,252.32	17,090.15

#### IV. ANALYSIS

The serviced times summarized in Table II are not statistically linearly related to the response bytes summarized in Table I. There is additional processing performed on the servers to prepare the responses.

Some of the queries were presented to the server using HTTP/1.0, with one request and response per web server connection. Some of the queries were presented to the web server using HTTP/1.1 with “keep-alive”, which allowed several request/response pairs per web server connection. There were no statistically significant differences between the sets of data collected using these two methods. For this application, the service time required to generate the data for the response dominates the total request service time, and there was no significant savings to the user when maintaining an open server connection to service sequential requests, as proposed by Windley in [8] for “Comet” AJAX applications to provide low latency data to AJAX client applications.

As previously mentioned, all data was collected by querying the production servers. The production HTML application averaged 180 page views per hour (3 per minute) over the months of October, November, and December; the peak day averaged 676 page views per hour (over 11 per minute). The queries for this study were performed sequentially, but response times suffered due to

contention for server resources from the ongoing production demands on the servers, including the queries from other users. The set of queries reported represents a sample of “real life” responses.

Figure 1 graphs the response time in seconds for 110 separate trials for the AJAX responses to the “List all courses” request. This request returned 1,196,681 bytes. The mean response time for this query was 12.69 seconds, with a standard deviation of 2.59 seconds. The minimum response time for this query was 10.62 seconds, and the maximum response time was 31.02 seconds. The peak value is truncated in Figure 1 for economy of presentation. The random nature of the poor response times is evident in this graph.

TABLE III  
RESPONSE TIMES IN SECONDS

HTML	All	99%	95%	90%
<b>N</b>	13,260	13,129	12,597	11,935
<b>Min</b>	0.31	0.31	0.31	0.31
<b>Max</b>	37.58	15.38	4.36	2.65
<b>Mean</b>	1.53	1.34	1.10	0.98
<b>Median</b>	0.89	0.87	0.81	0.75
<b>Std Dev</b>	2.44	1.53	0.82	0.64
<b>C.V.</b>	159.7	114.1	74.0	65.0
<b>Total</b>	20,252	17,618	13,872	11,685
<b>% CV Improvement</b>		28.6%	53.7%	59.3%
AJAX	All	99%	95%	90%
<b>N</b>	13,260	13,128	12,598	11,935
<b>Min</b>	0.25	0.25	0.25	0.25
<b>Max</b>	34.42	11.26	3.48	2.19
<b>Mean</b>	1.29	1.11	0.92	0.82
<b>Median</b>	0.73	0.71	0.65	0.60
<b>Std Dev</b>	2.26	1.24	0.67	0.59
<b>C.V.</b>	175.2	111.6	73.4	65.9
<b>Total</b>	17,090	14,541	11,545	9,775
<b>% CV Improvement</b>		36.3%	58.1%	62.4%

While the large portion of requests received reasonable response service, a few of the requests received very bad response service, just as in “real life”. These poor response times were seemingly random - not predictable, not repeatable, there was no correlation between poor HTML service and poor AJAX service, and there was no correlation between the size of the response and the reception of poor service.

Table III summarizes the response times for four groups of requests. The All group includes all queries. The 99% group includes 99% of the queries; it excludes the longest 1% of query response times. Correspondingly, the 95% group excludes the longest 5%, and the 90% group

excludes the longest 10%. That is, the service values for 95% of requests are summarized in the 95% column in Table III. Note the improvement of the coefficient of variation (cv) values when moving from the all down to the 90% group. For HTML, the cv decreases from 159 to 65, an improvement of 59%. Similarly, the cv for AJAX improves from 175 to 66, an improvement of 62%.

As an example, 95% of HTML requests received service in 4.36 seconds or less. In the longest 5%, one unfortunate response received a service time nine times as large. For AJAX requests, 95% received service in 3.48 seconds or less, while the longest AJAX request received a service time ten times as large.

Following the methodology used in [4] and in [6], the percent improvement is defined as:

$$(\text{HTML} - \text{AJAX}) / \text{HTML}$$

The mean percentage improvement for the response size in bytes is 56%. This is found by computing the percentage improvement for each query and then calculating the mean of these values. Since the response size is deterministic, the percentage improvement in response size for a given query remains the same for every repeat of the query.

Determining the percentage improvement for response times is more complicated due to the randomness in the data. Computing the percentage response time improvement for each query and then calculating the mean of these values gives an improvement of 2.4%. However, for about 4% of these queries, the AJAX response time was larger than the HTML response time. Indeed, one query resulted in a performance “improvement” of -7135%.

TABLE IV  
TYPICAL USER’S IMPROVEMENT

7 queries	Response bytes	Response secs
HTML	447,973	6.209
AJAX	202,972	5.117
Improvement	54.7%	17.59%
<hr/>		
95% HTML	447,973	5.733
95% AJAX	202,972	4.543
Improvement	54.7%	20.76%

Calculating the performance improvement from the means of the 95% groups in Table III (that is, the improvement of the mean response times of the two 95% level groups) gives an improvement of 16.8%. Calculating the performance improvement from the means in Table II (that is, the improvement of the mean response times of all of the data) gives an improvement of 15.6%.

In a “real life” use of this application, a typical student presents seven queries to build a semester schedule. Using the median values from Tables I, II, and III and totaling over seven queries produces Table IV. A typical user would experience an improvement in bytes transferred of almost 55%, and an improvement in service time of almost 21%.

## V. CONCLUSION

The savings in transmission time between the server and client are significant due to the average 56% reduction in the bytes making up a response. AJAX can significantly reduce the bandwidth required for the client to receive a response, or can improve the response of the application as perceived by the user by presenting a response faster than the traditional HTML application.

However, the reduction in time that the server spends generating a query (16%) indicates that AJAX is less useful as a server productivity tool. In the case of this application, a more efficient database query system is more likely to reduce the server query wait time for all users than the use of the AJAX application. A future extension of this study will measure the contention on the server as well as the service response times, to separate the contention part of the service times.

When the client-server bandwidth needs to be improved, AJAX can be effectively utilized. However, AJAX can have less impact on server utilization than client-server bandwidth.

## Acknowledgment

Dan Chase assisted in the collection of data for this work.

## REFERENCES

- [1] L. Paulson, “Building rich web applications with AJAX”, *IEEE Computer*, 38(10), 2005, 14-17.
- [2] Crane, Pascarello, James, *Ajax in Action*, Manning Publishing Company, Greenwich, CT, 2005.
- [3] AJAX Patterns, “Ajax Examples”, retrieved December, 2006, <http://ajaxpatterns.org/Examples>.
- [4] AjaxInfo.com, “Measuring the Benefits of Ajax”, retrieved March, 2006, <http://www.ajaxinfo.com/ucst/index.html>.
- [5] A. White, “Measuring the Benefits of Ajax”, retrieved March, 2006, <http://www.developer.com/xml/article.php/3554271>.
- [6] C. Smullen, S. Smullen, “Modelling AJAX Application Performance”, 524-074, *Web Technologies, Applications, and Services 2006*, ed. J.T. Yao, ACTA Press, Calgary, 2006.
- [7] Sarissa, <http://sarissa.sourceforge.net/> retrieved March 2005. The Sarissa project page is <http://sourceforge.net/projects/sarissa>.
- [8] P. Windley, “Comet: Beyond AJAX”, retrieved December, 2006, <http://www.irishdev.com/NewsArticle.aspx?id=2173>.

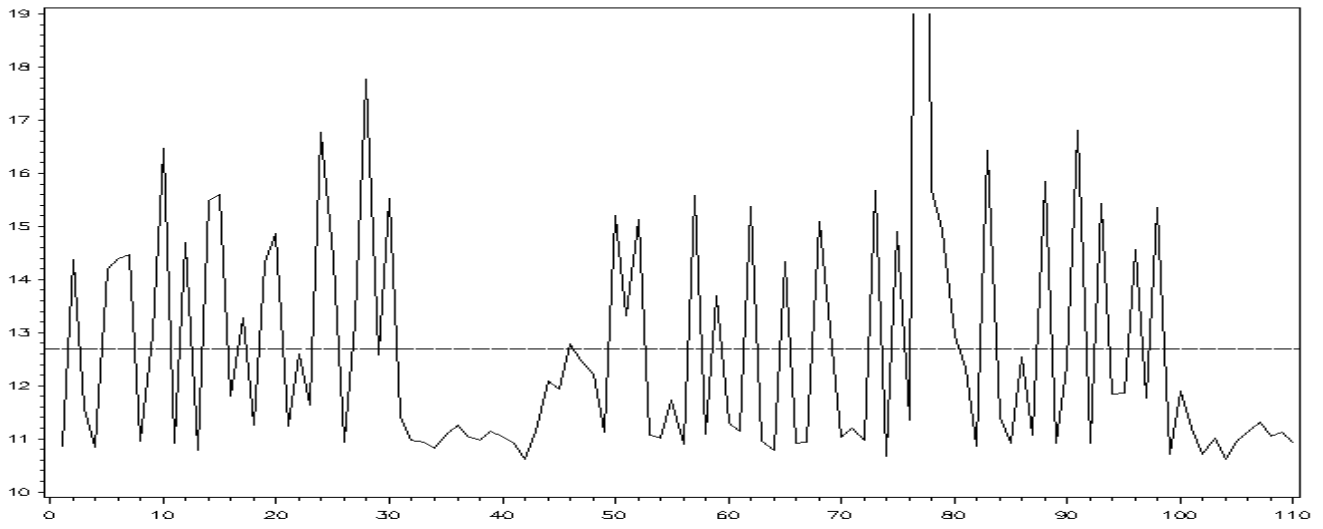


Fig 1. Response time in seconds for each of 110 trials of the same query