2

Designing Web Systems with AJAX:
A Study in the Reduction of Web Page Latency and Bandwidth Costs

Abstract

Designing Web systems is a compromise of serving pages quickly and reliably and conserving server resources and bandwidth. Webmasters attempt to deliver their site's pages as fast as possible while administrators try to reduce the costly consumption of bandwidth. These problems have been addressed by several techniques, including caching and prefetching, but have not yet been completely solved. The proposed research will design, implement, and experimentally evaluate the use of a new web development technique, AJAX, to load pieces of a Web page asynchronously. The project proposes that by using this technique, the average latency and overall transfer size of a Web page can be decreased.

1 Introduction

There are conflicting goals in the design of Web systems. For the Webmaster, the goal is to deliver the content of their pages in the shortest time possible. For the system administrator, the goal is to reduce the consumption

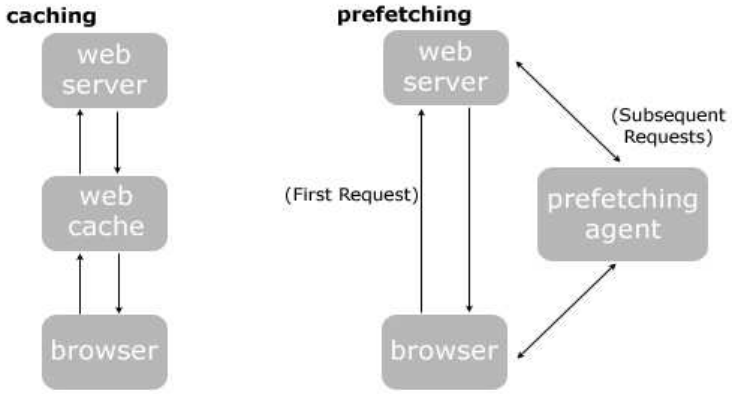

Figure 1: Caching and Prefetching Schemes

of resources, such as processing and bandwidth [14]. In an effort to provide a solution to these issues, several techniques have been developed, namely caching and prefetching.

Caching Web pages is a common way that Web systems can accomplish both of these conflicting goals. For end users, caching provides faster access to pages, while decreasing the amount of network traffic on the server. Prefetching pages is another technique used to accomplish the goals mentioned above. If a Web server can accurately predict which pages a user is likely to need ahead of time, these pages can be "preloaded" for future use. When they are eventually requested by the user, the access time is much faster. Figure 1 illustrates these two different schemes. Different caching and prefetching techniques and the issues related to them are discussed in Section 2.

Despite the advantages of using these techniques, there are drawbacks as

well. With caching, there must be a robust replacement policy that consistently provides users with "fresh" copies of pages, otherwise a user could be presented with outdated information. Pages that are updated on the server must also eventually be updated on the cache [14]. The choice of how often this occurs is crucial to the system; administrators want to conserve network resources while still providing users with the most current data. Another problem with caching occurs when there is a *cache miss*. If a client requests a page from the cache and it is not found, another request must be sent to the server to fetch the page. Due to the additional request, the overall latency could potentially be longer than it would have been had the page been requested from the server originally. Combined with the fact that cache misses occur in almost 50% of requests [12], it is obvious that caching is not always the most effective way to reduce latency.

There are problematic issues related to prefetching as well. Most notably, the Web server must make an assumption about the behavior of any given client in order to preload the presumed next page(s) [13]. If the assumption is incorrect, then unnecessary data is transferred, wasting space and bandwidth. Ideally, prefetching uses idle browser time in order to fetch new pages. However, if other processes are running outside the browser that are utilizing network resources, such as a streaming media application, there is a risk of resource competition between these processes and the prefetching scheme.

Because of the limitations of both caching and prefetching, it is desirable to develop an alternative system for decreasing Web page latency and transfer

size. This project proposes using AJAX, a new Web development technique, to accomplish these goals. Asynchronous JavaScript and XML, or AJAX, is not necessarily a new technology. Rather, it is a Web design paradigm that has recently become popular for use in new applications. AJAX functions by creating a JavaScript *XMLHttpRequest object* that has the ability to request data from an outside page and handle the response without reloading the entire original page [7].

Consider the following example of a traditional form used for signing up for a service on the Web. A user might fill in information such as a desired username, a password, and perhaps contact information. Upon submitting the form, the username must be checked against a list of current users to avoid duplication. If there is a conflict, the user is returned to the page and must enter a new name. Using AJAX, the username can be checked as soon as the field is changed, before submitting the form. Thus, as the user is completing the remainder of the form, he or she is notified that the selected name is invalid, and must be changed. Rather than sending the entire form over and over again, only the necessary data is transmitted.

As the example shows, AJAX has the potential to cut down on latency of transmitting pages, as well the amount of bandwidth consumed by transmitting data. Moreover, this technique can be accomplished on the "developer side" of the Web system. In other words, Webmasters can employ this technique without concerning themselves with constructing a caching or prefetching scheme.

The remainder of the paper is organized as follows. Section 2 examines prior work done in the area of caching and prefetching, and the issues associated with several different schemes. In Section 3 the thesis of the project is formally stated. Section 4 discusses how the project will be implemented, tested, and evaluated. In Section 5, a proposed timetable is given of how the project will be accomplished. Finally, concluding remarks are given in Section 6.

2 Prior Work

Several projects have contributed to the reduction of latency of Web pages and network resource costs. Wong et al. [14] evaluated the tradeoff between resource consumption and page staleness in Web caches. Their hypothesis was that the freshness of pages was proportional to the amount of resources utilized in a system. The goal was to develop an optimal page transmission strategy that conserved network resources while keeping cached pages as fresh as possible. The group mathematically defined equations for staleness and transmission and derived optimal values from them. Their results reflect their hypothesis and suggest that perhaps caches should be organized into similar content groups, so that an optimal strategy can be used for each cache. However, the research was highly theoretical, and should be applied in a real-world setting.

Brian Davison has contributed to several papers concerning the prefetch-

ing of Web pages ([3], [4]). One such paper [4] proposed prefetching data based on the content of pages recently viewed by the user. He proposed that this technique would more accurately reflect user interests in the predictions. The results showed improvements over random link selection, however the content-based approach only accounts for the HTML of the given page. Other resources, such as images and dynamic content are omitted.

Dynamic content can be a problematic issue when designing caching and prefetching schemes. Many dynamically-generated Web pages cannot be cached or prefetched, because they are constantly changing. Candan et al. [1] address the issue of caching dynamic content for database-driven Web sites. Their paper proposed a system that has multiple web servers, one database server, and an invalidator that observes updates to the database and identifies pages that are affected by these updates. The experimental results showed that their configuration required was faster to respond to database updates and suffered less time penalties during a cache miss than other configurations.

Luo et al. [9] explored the use of proxy caches for database-driven Web sites. Their conjecture was that a database under a heavy load can become a bottleneck and a proxy cache can distribute some of the network traffic. They devised two schemes, passive caching, where a database query always results in the same page (unless the data in the database itself is changed), and active caching, where the proxy plays a role in the query processing. Their results show that their passive caching scheme was successful with

respect to the TPC-W benchmark, and their active caching scheme showed performance gains in real-world applications.

Foxwell and Menascé proposed a prefetching scheme for use in the results of Web searches [5]. The project investigated the behavior of users of Web search engines, specifically Yahoo and Lycos, and determined an optimal number of documents to prefetch based on an average hit ratio. They concluded that for certain classes of users and Web sites, prediction of pages is possible, however they noted that there is a cost of network resource use.

3 Thesis Statement

Caching and prefetching techniques are useful in reducing latency and bandwidth costs, but they are not effective in all situations. Caches can contain "stale" pages and prefetching can waste resources by preloading incorrect data. Because of these issues, it is fitting to develop an alternative system that addresses the aformentioned problems. As such, this project will evaluate the following hypothesis:

By implementing a Web system using AJAX, the download time of Web pages, in seconds, and the overall amount of transmitted data, in terms of kilobytes, can be reduced.

4 Implementation and Methodology

Asynchronous JavaScript and XML, or AJAX, is implemented using JavaScript, a Web-based scripting language that is embedded into HTML documents. The process of adding the functionality of AJAX to a Web page is as follows. First a function is defined to create the *XMLHttpRequest object*.

```
function createRequestObject()
{
    if (window.XMLHttpRequest)
        var request = new XMLHttpRequest();
    else if (window.ActiveXObject)
        var request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Because of inconsistencies between Web browsers, there are two possible objects that can be created. The function first attempts to create the standards compiant object that is used by Mozilla (including Firefox), Opera, Safari, and other new browsers. If this object cannot be created, then the Microsoft equivalent ActiveX object is created for the Internet Explorer browser. Next, a function is created to send the request to another page on the server.

```
function sendRequest()
{
    request.open('GET', 'page.php', true);
    request.onreadystatechange = handleResponse();
    request.send(null);
}
```

This function uses the request object that was created in the previous function. The `open` method takes three parameters. The first is the HTTP request method, the second is the URL of the page that the request is being sent to, and the third is a Boolean value that specifies whether the request should be asynchronous. The value of `onreadystatechange` defines another function that will process the response of the request. Finally, the `send` method can transfer additional data to the requested page is the HTTP request method is POST rather that GET. After sending the request, a function must be defined to process the response of the request.

```
function handleResponse()
{
    if (request.readyState == 4 && request.status == 200)
        // The response was received and can be processed.
        // Process response...
    else
        // Error.
}
```

Before any processing can take place, the request must be finished and must receive a response from the server indicating that the request is successful. The value of `readyState` can be one of five possibilities.

- 0: The request is uninitialized.
- 1: The request is loading.
- 2: The request has finished loading.

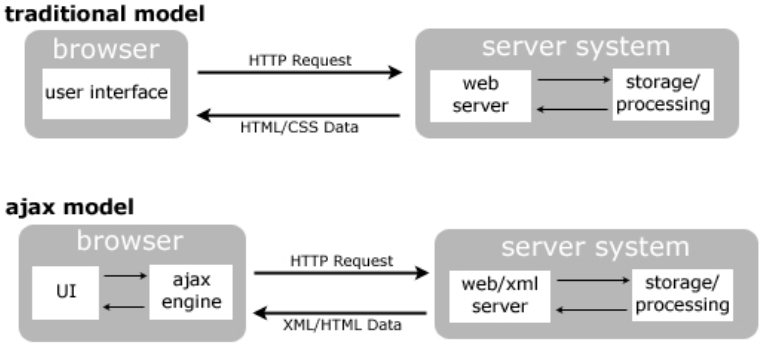

Figure 2: Web Application Models: "Traditional" vs. AJAX [7]

- 3: The request is interacting with the requested page.
- 4: The request is complete.

The `status` value can be any of several server responses. Some common responses are 404 (file not found) and 403 (permission denied). The value 200 indicates that the request has succeeded. After a successful response, the received data can be processed as the developer sees fit. Recalling the example given in Section 1, a request could be sent to a page that determines unique usernames. A response could be in the form of `true` or `false` and the form could be adjusted to indicate whether the user must choose a new name.

AJAX is most well known for its ability to dynamically alter the contents of a page. Using JavaScript's built-in Document Object Model (DOM) manipulation functions, the data from a requested page can be inserted vir-

tually anywhere in the page. It is then possible to dynamically load content such as Cascading Style Sheets (CSS) and additional JavaScript code into their respective `<style>` and `<script>` tags. Because this can be done asynchronously, the remainder of the page can still proceed to load, while the AJAX functions are retrieving the additional data. This contrasts with the traditional top-down loading of Web pages, as illustrated by Figure 2. The time it takes to parse through large amounts of JavaScript and CSS code can significantly slow the loading of the page [11]. The AJAX technique can overcome this bottleneck problem by loading this data separately and simultaneously. Moreover, by sending only necessary data, rather than an entire page, bandwidth can be conserved.

To aid in the building of the experiments, a tool will be created that automatically converts a "traditional" Web page into an AJAX-enabled page. This tool will parse the data of a given HTML document, and extract any CSS and JavaScript information. Next, the extracted information will be stored in external files created by the tool. Finally, the tool will generate AJAX functions similar to those previously described and insert them into a newly generated HTML document.

This project will evaluate it's hypothesis (Section 3) on two metrics, time and space overhead. Time will be measured in terms of the amount of time (in seconds) needed for a page to load. This can be observed by inserting time measurement scripts into the pages themselves, or by using load testing software, such as Siege [6]. Space will be measured in terms of the amount

of data transferred (in kilobytes) from the server to the client. This can be assessed using a packet capturing tool, such as Ethereal [2]. Specifically, the experiments to be conducted can be outlined as follows:

1. There will be eight Web pages created for use in the experiments, which will be of two different configurations.

   (a) The HTML page is linked to outside CSS and JavaScript code via `<link>` and `<script>` tags.

   (b) The CSS and JavaScript code is written inside the HTML page within their respective tags.

   These different configurations will be used to test the effectiveness of the AJAX system. This is not a complete list of configurations, but is most representative of the experiments to be performed.

   The experiments will also have a set of two variables, CSS and JavaScript size, and HTML size. These will serve to measure the scalability of the AJAX system and the effects of increasingly larger data sizes. Each configuration will be tested on combinations of small and large CSS and JavaScript, and HTML sizes.

2. For each of the eight Web pages above, an equivalent AJAX-enabled page will be automatically generated using the conversion tool previously described.

(a) If the conversion tool cannot be created, the AJAX-enabled pages can be hand-coded.

3. Several experimental trials will be run for each configuration and set of variables.

   (a) Loading time will be recorded using time measurement scripts or by using a tool similar to Siege.

   (b) The amount of data transferred will be measured by observing the number of packets transferred from the server to the client. The tool Ethereal can be used to capture network packets.

4. The Web pages will be viewed primarily through Mozilla Firefox 1.5, on a Windows XP platform. For completeness, the pages will be viewed in alternative browsers and operating systems, but the performance of each one will not be measured.

5. Data will be collected and analyzed, and averages and standard deviations will be calculated and plotted.

5 Research and Writing Timetable

The following is a rough outline of the proposed schedule of tasks for researching and writing the document:

- Tasks for CS600 [redacted]

  - Background research on caching and prefetching: 16 October [redacted] - 31 October [redacted] (3 weeks)
  - Finalize and defend project proposal: 31 October [redacted] - 13 November [redacted] (2 weeks)
  - Begin implementation of HMTL/AJAX conversion tool: 13 November [redacted] - 27 November [redacted] (2 weeks)
  - Write/revise initial chapters of final project: 27 November [redacted] - 11 December [redacted] (2 weeks)

- Tasks for CS601 (Spring 2006)
  - Conclude research: 15 January [redacted] - 29 January [redacted] (2 weeks)
  - Finish design/implementation of HTML/AJAX conversion tool: 29 January [redacted] - 19 February [redacted] (3 weeks)
  - Perform experiments and collect data: 19 February [redacted] - 5 March [redacted] (2 weeks)
  - Write remaining chapters analyzing experimental results: 5 March [redacted] - 26 March [redacted] (3 weeks)
  - Finalize and defend thesis: 26 March [redacted] - 23 April [redacted] (4 weeks)

The goal of this research will be to successfully implement a tool that converts a given HTML document into an AJAX-enabled document. Additionally, the experiments will attempt to measure the effects of AJAX in as

many situations as possible. However, if research or implementation time is curtailed, the experiments can instead be tested on hand-coded AJAX Web pages. Further, the number of experiments can be reduced, or the metrics of measuring them can focus on either the time or space overhead. If there is insufficient time for experimentation, the project will present a review of current Web systems and how AJAX could improve their performance.

6 Conclusion

In designing Web systems, the issue of providing users with data as fast as possible while minimizing the use of resources is significant. Several methods of reducing Web page latency and network resource consumption have been discussed, namely caching and prefetching. These methods have been shown in related projects to reduce latency and bandwidth but have also been shown to be uneffective in certain situations, such as when cached pages become "stale" or when a prefetching scheme makes incorrect assumptions.

The proposed research attempts to provide an alternative method to solving these key issues, while avoiding the problems of previous schemes. With AJAX-enabled Web pages, data can be loaded in parallel, relieving the bottlenecks that may occur when parsing through script and style information. Moreover, by creating a tool to automatically convert from the standard Web model to the AJAX Web model, this project will provide a simple way for Web developers to decrease the latency of Web pages while still conserving

network resources. Future work may include comparing the proposed system against various caching and prefetching schemes. Additionally, experiments could evaluate the effectiveness of AJAX with respect to the size of CSS or JavaScript alone, or specifically measure the differences in performance with alternative browsers and/or operating systems.

References

[1] K. Selçuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proceedings of the 20th ACM SIGMOD International Conference on Management of Data*, pages 532–543, New York, NY, USA, 2001. ACM Press.

[2] Gerald Combs. Ethereal: A network protocol analyzer. `http://www.ethereal.com/faq.html`, 2005.

[3] Brian D. Davison. Adaptive Web prefetching. In *Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW*, pages 105–106. Eindhoven University of Technology, 1999.

[4] Brian D. Davison. Predicting web actions from HTML content. In *Proceedings of the 13th ACM Conference on Hypertext and Hypermedia*, pages 159–168, New York, NY, USA, 2002. ACM Press.

[5] Harry Foxwell and Daniel A. Menascé. Prefetching results of web searches. In *Proceedings of the 1998 Computer Measurement Group Conference*, pages 6–11, 1998.

[6] Jeffrey Fulmer. Siege. `http://www.joedog.org/siege/index.php`, 2003.

[7] Jesse James Garrett. Ajax: A new approach to web applications. `http://www.adaptivepath.com/publications/essays/archives/000385.php`, 2005.

[8] Wenwu Lou and Hongjun Lu. Efficient prediction of web accesses on a proxy server. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 169–176, New York, NY, USA, 2002. ACM Press.

[9] Qiong Luo and Jeffrey F. Naughton. Form-based proxy caching for database-backed web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 191–200, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[10] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *SIGCOMM Computer Communication Review*, 26(3):22–36, 1996.

[11] Eddie Traversa. Ajax: What is it good for? `http://dhtmlnirvana.com/ajax/ajax_tutorial/`, 2005.

[12] Jia Wang. A survey of web caching schemes for the Internet. *SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[13] Zheng Wang and John Crowcroft. Prefetching in the World Wide Web. In *Proceedings of the 1996 Global Telecommunications Conference*, pages 28–32. University College London, 1996.

[14] Johnny W. Wong, David Evans, and Michael Kwok. On staleness and the delivery of web pages. In *Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research*, page 17. IBM Press, 2001.