

Reducing Web Browsing Delay using Profile-Based Prefetching

Wallapak Tavanapong

Kien A. Hua

Simon Sheu

School of Computer Science, University of Central Florida, Orlando, FL 32816-2362, U.S.A.

E-mail: {tavanapo,kienhua,sheu}@cs.ucf.edu

Abstract: This paper presents a novel technique for prefetching Web pages using the access patterns captured from the past behavior of the clients. For each page requested, the client also prefetches the relevant pages and downloads them along with the requested page in a batch to save communication overhead. The prefetching of these relevant pages can be done while the user is browsing through the requested page. Since future references to these prefetched pages can be satisfied without further communication to the server, this approach noticeably reduces the amount of time needed to browse the pages. The simulation results presented in this paper confirmed our observation.

1 Introduction

World-Wide Web (WWW) is the most popular application of the Internet today. The Hypertext Transfer Protocol (HTTP) is a simple application-level protocol used by WWW to provide convenient access to vast amount of geographically distributed information written in Hypertext Markup Language (HTML). Each HTTP interaction of HTTP 1.0 [Berners-Lee 93] comprises important steps as follows. The client opens a TCP connection with the server. After the client receives an acknowledgment from the server, it transmits an HTTP request to the server which performs some tasks to satisfy the request (i.e., retrieve an HTML document or execute a CGI program), sends the response back to the client, and closes the TCP connection. Upon receiving the response, the client parses the returned HTML document, so called page to extract inlined images (i.e., images embedded in the document). Each inlined image requires one HTTP interaction.

The explosive growth of WWW itself places an extreme demand on the network infrastructure. Moreover, HTTP's simplicity contributes to excessive network bandwidth requirements due to the following:

- **High connection costs:** One round trip time (*RTT*) is required to setup a connection between a server and a client. At least half of the time is due to propagation delay [Padmanabhan et al. 95], which cannot be further improved. Connection establishment also incurs processing overhead on both server and client.
- **Non-persistent connection:** For each inlined image in the requested page, a new connection needs be established instead of using the connection previously established for the requested page. As a result, two round trip times are required per page or one inlined image: one round trip time to setup a connection and another to fulfill the request.
- **Slow start:** To avoid network congestion, when a TCP connection is first started, the sender must wait for an acknowledgment packet before it can transmit more packets. Light network traffic would then allow the sender to transmit more packets before it waits for acknowledgments of those packets previously sent.
- **Non-shared congestion information:** Non-persistent connection causes unnecessary connections for a requested page with at least one inlined image; network congestion information realized in the first connection is not shared by the following connections. For the congested network path, these connections will interfere with each other.

To address high demand on the network bandwidth and improve the amount of time needed to retrieve pages, several approaches have been taken and can be categorized as follows:

- **Persistent Connection:** Persistent connection significantly reduces TCP open/close connection overhead by sharing a connection among multiple HTTP requests. That is, requests for inlined images in the page are

serialized within the connection. Persistent connection is included in the current HTTP protocol (HTTP 1.1) [Fielding et al. 1997].

- **Pipelining:** This approach aims at reducing the number of packets transmitted, it can be further divided into two techniques:
 - *Server Initiative:* Upon receiving an HTTP request from a client, the server retrieves the requested page and parses it to extract inlined image references (e.g.,). It transmits the requested page, retrieves the inlined images, and transmits them in the order of the references [Padmanabhan et al. 95]. This technique saves about half a RTT for each image in addition to the savings due to persistent connections.
 - *Client Initiative:* Upon receiving an HTTP response from the server, the client searches the received page for inlined image references. For each inlined image reference, it issues an HTTP request to the server without waiting for a previously requested image to arrive. The response messages are still serialized [Nielsen et al. 96].
- **Caching:** Caching is the most popular approach because it effectively reduces network bandwidth requirements. This approach does not require changes in HTTP protocol. Caching can be done per client basis and/or through cache servers named *Proxy servers*. In essence, frequently requested pages are maintained in memory and in disk cache. When requested, these pages are returned to the client without contacting the source servers. WWW caching techniques have been investigated quite extensively in both academia (e.g., Harvest Cache [Bowman et al. 94, Neals et al 96] [Williams et al 96]) and industries (e.g., Netscape Proxy Server [Netscape 98], Oracle WebServer [Oracle 98]).
- **Push Technology:** Among the aforementioned approaches, Push technology is the newest effort to improve the time one takes to browse Web pages. In essence, Push technology allows users to specify their interest at different levels (i.e., page level, site level, etc). Push servers and clients collaborate to deliver these pages to the users periodically or according to user-defined schedules. As a result, the users can browse through the pages with essentially no waiting time provided that they are not interested in any other pages.

In this paper, we present an alternative approach, *Profile-based Prefetching (PbP)*, to improve the Web browsing time. The technique can be used in addition to the techniques mentioned above. In our technique, when HTML documents referred in hypertext links in a page are predicted as strongly related to the requested page according to the server profile, these documents will also be fetched and transmitted to the client along with the requested page. Once the requested page arrives, the browser can present it to the user while it is downloading the prefetched pages. When these pages are later requested, since they are already in the client prefetch buffer, no further communication to the server is needed. It is not obvious to determine at what level of relevancy should a page be prefetched. Prefetching too many pages may affect the overall performance of the network if they are not used.

The remainder of this paper is organized as follows. In Section 2, we describe the proposed prefetch technique in detail. The simulation model and the performance results are discussed in Section 3. Finally we give our concluding remarks in Section 4.

2 Profile-based Prefetching (PbP) Technique

2.1 Motivation

Profile-based Prefetching technique is motivated by the fact that in general, once a user goes to a Web site, he/she generally browses around for several pages before leaving for another site. Since the user follows hyperlinks upon his/her interests, it is likely that links are not followed uniformly. If all pages in which the user is interested can be downloaded in a batch, we can address the problems of (i) high connection cost, (ii) non-persistent connections, and (iii) slow start because browsing through these pages will not incur anymore communications to the server.

We can either predict each user's interest using cookies or mine a consensus of interests (i.e., generally what pages will be requested after the current page) with some confidence from the access log files recorded by the

Web Server. This information not only is valuable for the Web administrator to eliminate uninterested pages, or balance loads among servers, but also can help to improve Web browsing time. To verify this approach, we developed a prefetching algorithm presented in the following subsections.

2.2 Terminology

Let us clarify in the following some of the terminology used in this paper.

<i>Client</i>	Browser issuing HTTP requests to some server
<i>Source server</i>	Server publishing HTML documents requested by the clients
<i>Page</i>	HTML document containing anchors and hyperlinks to other pages
<i>Parent page</i>	Page being requested
<i>Descendant page</i>	Page directly and indirectly referred to by its parent page
<i>Child page</i>	Page directly referred to by its parent page
<i>Co-reference (CR)</i>	Probability (percentage) that a child page is needed after its parent page is browsed
<i>Co-reference profile</i>	Data structure containing co-references of related pages at a site
<i>Co-reference threshold (CT)</i>	Minimum co-reference for the client to accept the prefetch of the descendant pages

Figure 1 shows an example of pages residing at a fiction site, *www.science.ucf.edu*. *facility.html*, *departments.html*, and *map.html* are child pages of *welcome.html*. *cs.html*, *biology.html*, and *history.html* are child pages of *departments.html*. The numbers between pages are the co-references. For instance, the co-reference of *departments.html* is 90. We will use this example to illustrate our technique.

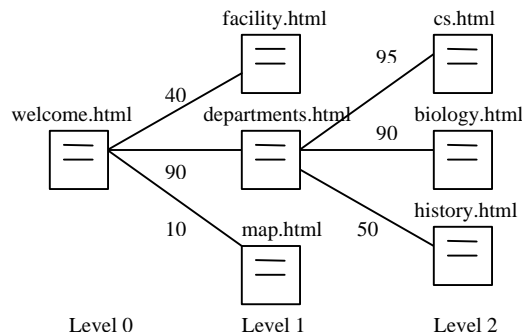


Figure 1: A Web site structure.

2.3 Technique

Initial Step: The server constructs its co-reference profile by applying data mining techniques [Agrawal 94, Agrawal 93] on its access log file. This issue is beyond the scope of this paper. We note that a co-reference is a measure between a child page and its parent. A co-reference of 100 means that every retrieval of the parent page is followed by a retrieval of the child page. A co-reference of 0 indicates that the child page is very unlikely needed after its parent page has been retrieved. The following demonstrates an interaction between the client and its source server when PbP is deployed:

Step 1: If a request is not a GET request, the client submits the request according to an HTTP protocol, and goto **Step 5**; otherwise, the client checks its local *prefetch buffer* for the requested page.

- a) If the page is already in the buffer, goto **Step 5**
- b) Otherwise, the client opens a TCP connection with the source server.

Step 2: After the connection is established, the client sends its HTTP GET request for the page. It also transmits the desired co-reference threshold along with other HTTP request headers. PbP allows clients to specify

their own co-reference thresholds since the clients may have different perspectives of how one page is “strongly related” to another.

- Step 3:** Upon receiving the request and its headers, the server retrieves the requested page and selects its descendant pages to transmit back according to the following criteria. A descendant page is prefetched if its parent page is prefetched, and its co-reference is greater than the co-reference threshold. The relevant pages can be visited in a breadth-first order starting from the page requested. The traversal can be limited to a fixed number of levels determined by the Web administrator. Since cyclic graphs, instead of trees, can occur in practice, the traversal along a certain path should be pruned if pages in the path have been visited previously. We note that the traversal is done using the profile that maintains only the hypertext linking information, not the page contents. This step, therefore, can be done very efficiently.
- Step 4:** The server transmits the requested page back followed by the descendant pages satisfying the prefetch criteria. It then closes the connection.
- Step 5:** The client interprets and displays the page. If this step is reached from **Step 4**, the client continuously receives the descendant pages while rendering the requested page. The prefetched pages are cached in a local buffer for future references.

We note that password protected pages should not be prefetched due to its protective nature. CGI programs should not be prefetched since their output can be large. Similarly, audio and video files are typically large, and should not be prefetched. Wrong prediction on these files would significantly waste network bandwidth.

To illustrate our strategy, let us consider the Web structure shown in Figure 1. Suppose that there is an HTTP GET request to *www.science.cs.ucf.edu/welcome.html*, and the desired co-reference threshold is 50. In this case, *department.html* is the only page in level one, which will be prefetched. This decision automatically excludes page *library.html* from being considered for prefetching. Among the three pages in level two, only *cs.html* and *biology.html* will be prefetched since their co-references are bigger than the co-reference threshold.

3 Performance Study

In this section, we develop a simulation model and present the performance comparison of the proposed technique and that of HTTP 1.1 protocol. Due to space limitation, we briefly discuss our simulation model and some of the results in this paper.

3.1 Simulation Model

We opt for a simulation model that is simple, yet can demonstrate the relative performance difference of the two techniques. Since both HTTP 1.1 protocol and PbP support persistent connection, in this simulation, we will not model the time cost of establishing and closing connections. We also assume that the time cost for issuing an HTTP GET request is negligible compared to the time cost for transmitting a page over the network. This is to be fair for the HTTP protocol since for each requested page, several GET requests need to be issued for its relevant pages while PbP issues only one GET request for the same relevant pages.

Average latency per page is used as our performance metric. It is an average time taken since a client submits a request until it receives the requested page. It is formally defined as

$$\text{Average Latency Per Page} = \sum_{\text{all clients}} \frac{\text{Total wait time for a client}}{\text{Total number of completed requests for the client}}$$

The reduction in the latency leads to the improvement in the users' browsing time. That is the users can browse through more Web documents given the same time period. We measure the latency in two situations: when clients behave according to the server profile and when they do not. We describe these two scenarios in more detail in the following:

- **According to the Profile:** Let us consider a child page with co-reference c with respect to its parent page. To determine whether the client will also request this child page after it has requested the parent page, we create a random number between 1 and 100. If the random number is greater than c , the corresponding child page is not needed; it is requested, otherwise.
- **Random Access:** In this situation, after downloading a parent page, to determine whether the client also wants to request a particular child page, we randomly create a number between 1 and 100. If the random number is less than 51, the child page is not needed; it is requested, otherwise. Thus, each child page has the same probability of 50% of being requested after its parent page has been downloaded.

Although the access pattern should generally follow the client behavior captured in the profile, the random-access workloads were included in our study to investigate the effect on clients who behave significantly different from the norm. In practice, this client should set a higher Co-reference Threshold to avoid downloading unneeded pages. To facilitate our simulation, we first conducted the following experiment.

- We randomly requested pages from 120 different actual Web sites. For each page, we measured the time taken to retrieve the page, the page size in bytes, and the number of hyperlinks in the page, which refers to other pages from the same site.
- We calculated the average of these measurement values. The average number of hyperlinks in a page was 15, and the average page size was 7650 bytes. The data transfer rate was 25000 bit per second (bps).

We use the above values in our simulation. The number of network channels was fixed at 60, which is calculated from the bandwidth of a T1 line (1.5Mbits/sec) divided by the data transfer rate. The network channels were shared among 1000 clients. They concurrently issued requests to the same server maintaining the co-reference profile. After the requested page arrived at the client, the client waited for some period of time (Think time) before issuing another request. Which page to be requested was determined, based on the scenario (according to the profile or random access) being considered. Think time is introduced to simulate the time the user takes to browse through the retrieved page.

3.2 Simulation Results

We varied the co-reference threshold from 20 to 100. Think time was varied from 60 seconds to 140 seconds. The results are plotted in Figure 2.

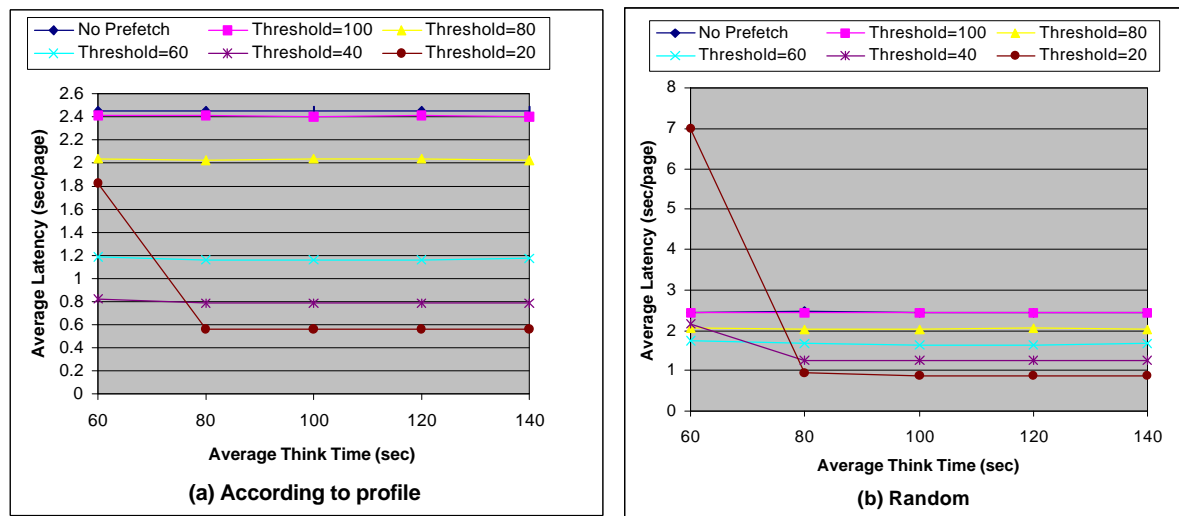


Figure 2: Average latency under various Think times

Figure 2(a) demonstrates the performance comparison of the two techniques when the clients follow the global profile. The two techniques perform comparably only when the clients set their co-reference thresholds to 100.

For most co-reference thresholds, PbP noticeably outperforms the HTTP protocol. This is because PbP takes advantage of the time while the user is thinking to download the relevant pages. We observe that the smaller co-reference threshold leads to the better latency in most cases in our study.

Comparing Figure 2(b) to Figure 2(a), we observe that the improvement in the latency is reduced. When the co-reference threshold is 20 and Think time is less than 80 seconds, PbP performs much worse than the HTTP protocol. This is because PbP prefetches too many pages, and these pages are not referenced. To provide a good average latency for various Think time, co-reference threshold of 20 and 40 are not good choices when the client behaves differently from the profile. Nevertheless, PbP with higher thresholds still outperforms the HTTP protocol.

4 Concluding Remarks

The advance in networking and server technology has revolutionized the way we publish information on the Internet. In recent year, WWW has evolved from retrievals of simple text/HTML pages to include also images, and at the present time, audio, video and Java applets. To accommodate these new features, several techniques have been investigated to extend HTTP protocol to better support the new functionality.

This paper presents another approach to improve HTTP performance using the access pattern captured from the past behavior of the clients. The co-reference probability of the Web pages are used in our scheme to determine whether to preload some of the pages along with the one requested to minimize the communication costs. Comparing this Profile-based Prefetching technique with HTTP, our simulation results indicate noticeable saving for service latency. We also observed from our studies that the proposed technique is able to maintain superior performance even if the clients do not behave in accordance with the server profile. To make our technique useful to a wide range of conditions, clients are allowed to specify a threshold to control the locally desired degree of preloading. Alternatively, the server can set the global threshold shared by all clients. In this case, a larger threshold can be used to limit the prefetching to only pages with very large co-reference probability, if large variances were detected in the server profile. Global thresholds were used in our simulation studies.

References

- [Agrawal et al. 93] Agrawal, R., Imielinski, & Swami, A. Mining Association rules between set of items in large databases. SIGMOD Washington DC, 1993.
- [Agrawal et al. 94] Agrawal R., Fast algorithms for mining association rules. Proceedings of the 20th VLDB Conference. Santiago, Chile, 1994.
- [Berners-Lee 93] Berners-Lee, T. Hypertext Transfer Protocol (HTTP).
- [Bowman et al. 94] Bowman, C. M., P. B., & Hardy, D. R. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado-Boulder.
- [Fielding 97] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., & Berners-Lee, T., Rfc 2068: Hypertext transfer protocol-HTTP/1.1. <ftp://ftp.is.edu/in-notes/rfc2068.txt>, January 1997.
- [Neal 96] Neal, D. The Harvest object cache in New Zealand. Proc. Of the 5th World Wide Web Conference Paris, France, May 1996.
- [Netscape 98] Netscape Communication Corporations. Netscape proxy server 3.5. http://www.netscape.com/comprod/server_central/product/proxy/proxy3_data.html, 1998.
- [Nielsen et al. 96] Nielsen, H. F., Gettys, J., Baird-smith, A., & Prudhommeaux, E. Initial HTTP/1.1 performance. <http://www.w3.org/pub/WWW/Protocols/HTTP/Performance/Pipeline.html>, December 1996.
- [Oracle 98] Oracle Corporation Oracle Proxy server release 1.0. <http://www.oracle.com/products/asd/proxy/proxy.html>. 1998.
- [Padmanabhan 95] Padmanabhan, V., & Mogul, J. Improving HTTP latency. Computer Networks and ISDN System <http://www.nasca.uiuc.edu/SDG/IT94/Proceedings/Dday/mogul/HTTPLatency.html>, December, 1995 25-35.
- [Williams et al 96] Williams, S., Abrams, M., Standridge, C. R., Abdulla, G., & Fox, E. A. Removal policies in network caches for World-Wide Web documents. SIGCOMM 96, 293-305.