Prefetching in World Wide Web

Zheng Wang
Jon Crowcroft

Department of Computer Science, University College London
London WC1E 6BT, United Kingdom

Abstract

This paper considers the use of prefetching in WWW for reducing perceived latency. We first look at a number of basic issues and tradeoff in prefetching. Our analysis shows that, unless the traffic is very light or the prefetching efficiency is very high, statistical prefetching may not necessarily reduce perceived delay. We then present an implementation of deterministic prefetching in a hotlist manager.

1 Introduction

As the World Wide Web (WWW) is now the *de facto* standard interface for interactive information services over the Internet, the perceived latency in WWW interaction is becoming an important issue. Currently, Web users often experience response delay of several seconds or even longer to non-local Web sites. For WWW to be acceptable for general daily use, the response delay has to be substantially improved.

The potential solutions to the problem may lie in the extensive use of caching and prefetching in WWW. Both caching and prefetching exploit the patterns and knowledge in the Web page accesses. Whilst caching exploits the probability of multiple requests to the same Web page, prefetching makes use of the knowledge in a client's requests to multiple Web pages.

In this paper, we focus our attention on the use of prefetching for reducing delay in WWW. We first look at latency and access patterns in WWW interaction, and examine basic issues and tradeoffs in prefetching. We then discuss deterministic client-initiated prefetching in some detail and present an implementation in the HotList Manager.

2 Latency in WWW Interaction

The perceived latency in WWW interaction comes mainly from two sources. At the lower level, the round trip time (RTT) is the most fundamental one. The RTT has two main components: processing latency in end systems, and communication latency over the network. The processing latency depends on the load of the end systems. The communication latency consists of queuing delay and propagation delay. An increase of bandwidth can reduce queuing delay but the propagation delay may not be changed.

The communication protocols decides the number of RTTs required for each interaction thus also affect the perceived latency. In current HTTP implementation, each object in a Web page requires a new TCP connection, hence at least 2 RTTs. The new persistent HTTP can reuse TCP connections so it requires only one TCP setup for each interaction [1].

The latency experienced by users varies considerably depending on the bandwidth available, the propagation delay to the server, the size of the Web page and the server load. To illustrate the delay Web users currently experience, we calculated the delay distribution with 79,430 samples from the client-based traces collected at Boston University during November 1994 to February 1995 [2]. Figure 1 shows the distribution of delay for retrieving an object from non-local servers (i.e., the servers other than the ones on users' LANs). For most objects, the retrieving time is between 0.4-4 seconds. A Web page usually contains a number of embedded objects, thus retrieving time for one web page is several times longer. The distribution of the transmission rates from non-local servers, produced with the same dataset, is shown in Figure 2. It is clear that bandwidth remains the dominant factor in latency.

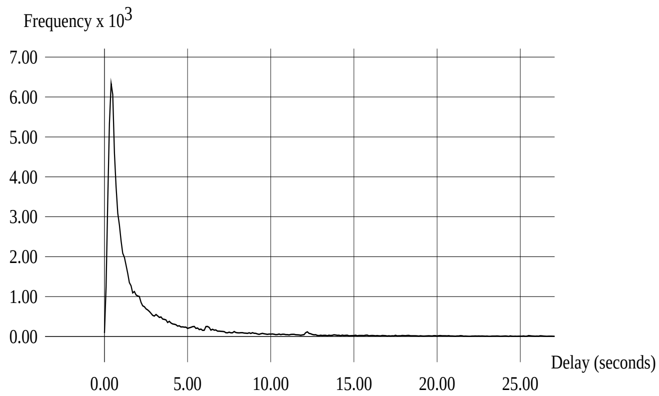

Figure 1: Distribution of Delay for Retrieving an Object from Non-Local Servers

3 Prefetching and Access Patterns

Prefetching tries to reduce the perceived latency by look-ahead. If one can anticipate the Web pages that a user will fetch in the future, it is possible to preload these Web pages

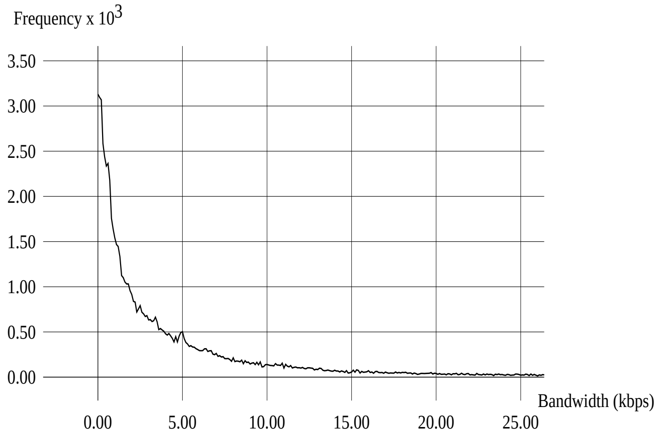

Figure 2: Distribution of Transmission Rates from Non-Local Servers

to the user before they are requested. The perceived latency can then be reduced to that for fetching Web pages from local disks or file servers on local nets. Thus, the effectiveness of prefetching depends on whether there is certain predictability in users' Web page accesses.

The information on access patterns may be derived from servers' access statistics or from clients' configuration. Recent studies on the WWW traffic show that there are considerable interdependencies among consecutive accesses to some Web pages [3]. Such results are not surprising. As WWW is a hypertext based information system, it is natural that many Web pages are closely linked together. The design of a Web page also places some physical constraints on what hyperlinks can be followed from a particular page, and the contents may also provide some strong leads as to the order in which the Web pages should be viewed. Such information may be used as the basis for predicting users' Web page accesses.

Users' personal preferences are also an important factor. The sequence of accesses is ultimately decided by users' individual selection. For regular services such as on-line newspapers, users often know in advance what they want to see. Information on users' future accesses may also be derived from users' planning tools such as diaries or todo lists.

4 Basic Issues in Prefetching

In this section, we examine some important issues in prefetching and discuss the tradeoffs between bandwidth and delay.

Server-Initiated vs Client-Initiated Prefetching

Prefetching may be initiated by clients or by servers. When a Web page is request, a server may anticipate what hyperlinks are likely to be followed, and preload the corresponding Web pages to the client. So, when the user moves to a hyerlink, the corresponding Web pages may already be transferred or being transferred.

For server-initiated prefetching to work, the client has to be prefetching-aware so it can deal with preloaded Web pages correctly. This would require extensions to the current HTTP protocol and modifications to both the client and server software.

A client can also initiate prefetching Web pages based on user's configuration or a user agent can monitor the patterns of past accesses for particular Web pages, and prefetch Web pages on the behalf on the user.

Client-initiated prefetching can be done by individual clients in a way transparent to the servers. The implementation is therefore much simpler.

Server-Initiated vs Client-Initiated Prefetching

The criteria for deciding whether a Web page should be prefetched can be either statistical or deterministic. In a statistical scheme, one can calculate the interdependencies of Web page accesses periodically based on the most recent access logs, and group Web pages with interdependencies higher than a certain threshold for prefetching.

Prefetching can also be configured statically by the users as part of their personalized user interfaces, or even by page designers as part of the content design. For example, a user may configure some sections of an on-line newspaper that are always read with prefetching so that these sections are prefetched when the newspaper's home page is accessed.

4.1 Bandwidth and Delay Tradeoff

Statistical prefetching can potentially have wide applications as the process can be easily automated. However, by its speculative nature, some bandwidth will be wasted, thus it can increase total bandwidth consumption. There is a general tradeoff between bandwidth and latency here. As we reduce the threshold for statistical prefetching, the latency may improve, but at at the price of increased bandwidth consumption. A study on FTP shows that the latency can be reduced by 67% for 7-fold increase in bandwidth [4].

Unfortunately, bandwidth is still scarce resources in most networks, particularly over long distance paths. Thus, statistical prefetching has be used with great care to avoid any waste of bandwidth. Note that, although the perceived delay for prefetched files are very low, the retrieving delay for non-prefetched files may actually increase as a result of the extra traffic load caused by prefetching. When traffic is heavy, aggressive prefetching, such as "get all links", may actually increase the average latency of all accesses.

We now look at the tradeoff in some detail. Suppose that P is the hit rate of prefetching (i.e., the probability of a correct prefetch), and D_o is the average retrieving delay without prefetching. Assuming that the retrieving delay is 0 for prefetched Web pages, and the delay is D_x for non-prefetched ones. So the average delay with prefetching D_n can be written as

$$D_n = P \times 0 + (1 - P) \times D_x = (1 - P) \times D_x$$

To ensure that prefetching reduces the average delay, i.e., $D_n < D_o$, we have

$$D_x/D_o < 1/(1-P)$$

For the sake of simplicity, we assume that the delay D_x and D_o can be calculated based on M/M/1 model, i.e.,

$$D_o = 1/(1-R_o)$$

$$D_x = 1/(1-R_x)$$

where R_o and R_x are respectively the link utilization without and with prefetching. We then get

$$P/((R_x - R_o)/R_o) > R_o/(1-R_o)$$

We define the efficiency of the prefetching E as the ratio of the hit rate of prefetching and the rate of traffic increase to achieve that hit rate, i.e.,

$$E = P/((R_x - R_o)/R_o) > R_o/(1-R_o)$$

The above implies that the efficiency of prefetching must be larger than $R_o/(1-R_o)$, otherwise, the average delay can actually be higher than that without prefetching. The inequality can also be written as

$$R_o < E/(1+E)$$

If E is known, one can calculate the maximum R_o for statistical prefetching to be useful. Figure 3 and 4 show the feasible regions for E and R_o. It is clear that prefetching is only useful when traffic is very light or prefetching efficiency is very high. For example, for $E = 0.5$ (i.e. for each 1% traffic increase, the prefetch hit rate improves 0.5%), R_o must be smaller than 0.3. For $R_o = 0.8$, E must be larger than 4. This is because when traffic is heavy, very little extra traffic may result in substantial increase in queuing delay. Unless the prefetching efficiency is very high, the extra delay experienced by non-prefetched pages may outweigh the decrease in the delay of prefetched pages.

Prefetching Through Caching Proxies

When prefetching is done through caching proxies, the prefetched Web pages are also cached at the proxies. This, in fact, has a number of advantages over prefetching Web pages directly to the client's machines. First, the prefetched Web pages on the proxies can be cached pages to other users who use the same proxies. Second, if the client's local cache is on a file server, to get a Web page from the local cache involves transmission over local nets anyway. In such cases, using caching proxies also on local nets has no extra communication costs. Finally, the caching format on proxies is not browser-specific, so the prefetching may be implemented in a standalone program outside the browsers. In section 5, we will present an implementation of prefetching in a hotlist manager.

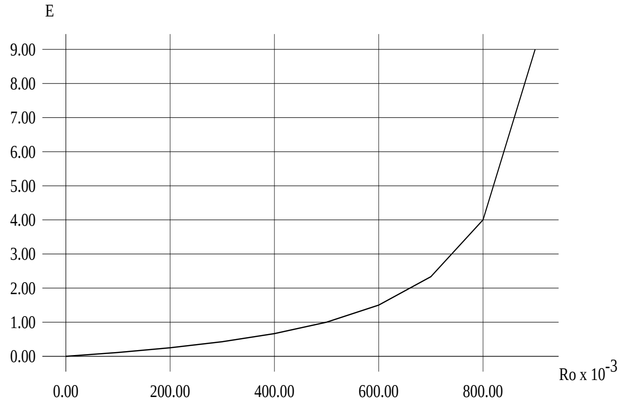

Figure 3: Feasible Region for E (Area above solid line $E > R_o/(1-R_o)$)

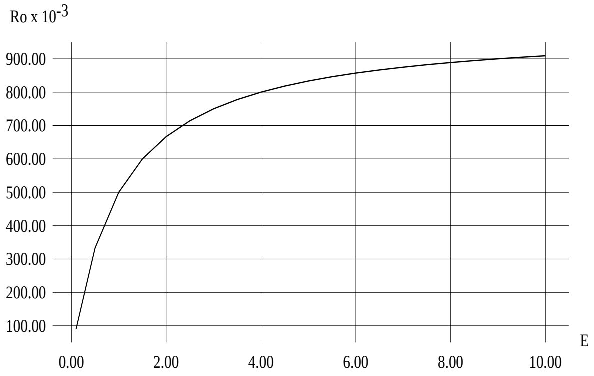

Figure 4: Feasible Region for R_o (Area above solid line $R_o < E/(1+E)$)

5 Deterministic Client-Initiated Prefetching

Deterministic prefetching is the most conservative type as it often has little or no bandwidth overhead, although its scope of use is limited. Nevertheless, when users know what needs to be prefetched, it can reduce perceived latency, and to some extent, even ease congestion at very little cost.

In deterministic client-initiated prefetching, prefetching is configured statically by the users. It can be implemented as part of the browser or simply as an add-on without changes to client and server software.

In this section, we discuss some potential use of the deterministic client-initiated prefetching in detail.

Batch Prefetching

There are many Web pages that are read on a regular basis, such as on-line newspapers, weekly work reports etc. They can be prefetched during the less busy period (e.g. early morning). For large Web pages, such as papers with large graphics, it may also make sense to fetch them in a batch mode and view them later on. Such prefetching, to some extend, is similar to mirroring but batch prefetching is more flexible as it does not require any central administration. It can be done through cache proxies so that all prefetching requests are naturally aggregated at the proxies. Batch prefetching can also potentially allow multicast to be used in distributing some large popular files or software.

Start-Up Prefetching

Prefetching can also be carried out when a browser is started. A set of pages users need to look at that day may be prefetched in the background. It can be integrated with planning tools so that a to-do Web Page is constructed each day for the users and corresponding Web pages are prefetched at the start-up for later viewing.

Pipelining with Prefetching

The current model for navigating is a series of ”click, fetch and view” operation. As a user usually spends some time (seconds or minutes) on a page, we can potentially pipeline the operation by fetching the next page while the user is looking at the current one. This is particularly useful for some information services, such as on-line newspapers, stock market prices and headline tracking services where users can easily specify the sequence of pages to be viewed.

6 Implementation

In this section, we present details of a UNIX implementation of deterministic client-initiated prefetching in a program called Coolist.

Coolist is a standalone hostlist manager implemented in Java. It has a number of useful functions such as proxy selection, personal search engine and prefetching. Coolist can be used in conjunction with Netscape, Mosaic and any Web browsers which support client APIs and caching proxy.

Coolist sits between the browser and caching proxy (Figure 5), and acts as a personal proxy. Under most circumstances, Coolist simply forwards traffic to and from the caching proxy. However, as Coolist can intercept any traffic between the browser and the proxy, it can also work on the requests and responses in various ways.

Although Coolist can prefetch most web pages directly by itself, it does not work with Web sites that require passwords, cookie or transparent redirection. In our implementation, Coolist uses client APIs [5] to control browsers to do prefetching. When Coolist needs to fetch a web page, it issues a client API command to the browser to open the web page specified.

For example, with NCSA Mosaic TCP/IP CCI, Coolist sends the following message a Mosaic browser:

```
GET URL <http://www.ncsa.uiuc.edu/SDG/
Software/XMosaic/CCI/cci-api.html> CURRENT
```

The browser receiving the message will open the specified URL and display in the current window. Invoking Netscape as:

```
netscape -remote 'openURL(http://home.netscape.com
/newsref/std/x-remote.html)'
```

has the same effect on Netscape browsers.

When doing batch prefetching, Coolist forks out a cron job with a line mode browser to fetch the web pages at specified time. In our implementation, we choose Lynx [6] (with the line mode option) since it is available on most Unix systems.

Although prefetched pages can be cached in Coolist, we use the caching proxy as the store for prefetched pages. This simplifies the design of Coolist and that prefetched pages can also be shared by other users connected to the same caching proxy.

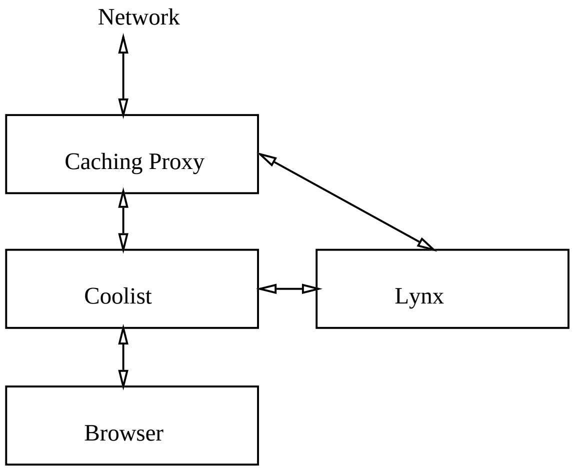

Figure 5: Implementation of Client-Initiated Prefetching in Coolist

Start-up prefetching is configured with a file called .todo in user's home directory. When Coolist is invoked, it starts a low priority thread to prefetching all web pages listed in the .to-do file. Web pages can be grouped together for pipelining prefetching which appears as a single entry in the bookmark.

7 Summary

In this paper, we considered the use of prefetching in WWW for reducing perceived latency. We examined a number of issues and tradeoffs. Deterministic client-initiated prefetching was discussed in some detail and an implementation based Coolist was presented.

References

[1] Jeffrey Mogul, *The Case for Persistent-Connection HTTP*, in Proceedings of ACM SIGCOMM'95, Oct 1995.

[2] Carlos Cunha, Azer Bestavros, and Mark Crovella, *Characteristics of WWW Client-based Traces*, Technical Report TR-95-010, Boston University, Boston, MA 02215, April, 1995.

[3] Azer Bestavros, *Using speculation to reduce server load and service time on the WWW*, in Proceedings of CIKM'95: The Fourth ACM International Conference on Information and Knowledge Management, Baltimore, Maryland, November 1995.

[4] J.D. Touch and D.J. Farber, *An Experiment in Latency Reduction*, in Proceedings of IEEE INFOCOM'94, Toronto, June 1994.

[5] Tom Magliery, Briand Sanderson, Writing Web software through client APIs, In the Tutorial Notes of 4th World Wide Web Conference, Boston USA, Nov 1995

[6] Lynx, *Lynx On-Line Documents*, (http://ftp2.cc.ukans.edu/pub/lynx/), 1995.