INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

User Directed Prefetching

A thesis

submitted to the Faculty of Graduate Studies and Research

of

Carleton University

in partial fulfillment of the requirements for the degree of

Master of Computer Science

in

Computer Science

By

VIVIAN TIN-WAI LEE

Sept. 2000

copyright 2000, Vivian T.W. Lee

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-57760-0

Canada

Abstract

Since an advantage of the Internet is that it provides quick access to the vast information distributed in the World Wide Web (WWW), there is a need to keep the user's waiting time to a minimum. Researchers have been focusing on reducing the user access latency through software enhancements. The two main areas of research are: 1) Caching, and 2) Prefetching. This thesis presents a prefetching cache proxy model, which allows an user to select the items to be prefetched. This thesis shows that the user directed prefetch algorithm can greatly reduce the number of prefetching documents. This thesis sets out to prove that under a low bandwidth situation, where the penalty of prefetching the wrong page is very high, the User Directed Prefetching Algorithm is more suitable.

Acknowledgments

I would like to give thanks to Professor Danny Krizanc, who gave me ideas and guidance throughout the research. My thanks also go to Radim Kolar, who shared the Smart Cache source code with me: the implementation of the Smart Cache is used in the proxy server described in this thesis.

Table of Contents

1. INTRODUCTION	1
1.1 PROXY SERVER OVERVIEW	2
1.2 THE HTTP PROTOCOL	3
1.3 CACHING.....	5
1.4 PREFETCHING.....	6
2. RELATED WORK.....	7
2.1 CACHING.....	7
2.1.1 LRU Algorithm	8
2.1.2 LRU-MIN Algorithm	8
2.1.3 Shared Caching Proxy.....	9
2.2 PREFETCHING.....	10
2.2.1 Web Crawler.....	10
2.2.2 Top 10 Prefetching.....	10
2.2.3 Intelligent Prefetching.....	12
2.2.4 Breath-first Prefetching.....	13
3. MOTIVATION	14
4. ASSUMPTION.....	16
4.1 PREFETCH TEXT AND IMAGE	16
4.2 EMBEDDED LINKS TRAVERSAL	16
4.3 INFINITE CACHE	17
4.4 UNDEFINED BROWSING STRATEGY	17
5. DESIGN OVERVIEW	18
5.1 DESIGN WALK-THROUGH.....	18
5.2 DESIGN ISSUES.....	22
6. EVALUATION	23
6.1 LIMITATION.....	24
6.2 EVALUATION GOAL.....	24
6.3 SIMULATION.....	25
6.4 WEB TRACES	25
6.5 URL LINKS DEPENDENCY.....	26
6.6 SESSION Delineation.....	27
6.7 WEB TREE RECONSTRUCTION.....	28
6.7.1 Log Traces Preprocessing.....	28
6.8 NO PREFETCHING SIMULATION	32
6.9 BREATH-FIRST PREFETCHING SIMULATION	32

6.10	USER DIRECTED PREFETCH	34
7.	SIMULATION RESULTS.....	36
7.1	NO PREFETCHING	36
7.2	BREATH -FIRST PREFETCH	37
7.3	USER DIRECTED PREFETCH	37
7.3.1	Hit Rate	38
7.3.2	Server Load	40
8.	CONCLUSION	42
	REFERENCES	44
	APPENDIX A: PERSONAL PROXY SERVER.....	47
	APPENDIX A-1: PROXY.JAVA	48
	APPENDIX A-2: REQUESTHANDLER.JAVA	52
	APPENDIX A-3: SELECTEDLINKREQUEST.JAVA	62
	APPENDIX A-4: SELECTEDLINKSCANVAS.JAVA	66
	APPENDIX A-5: PERSISTENTREQUESTLIST.JAVA.....	68
	APPENDIX B: TRACES STATISTICS	75
	APPENDIX B-1: SESSIONLENGTH.JAVA	76
	APPENDIX B-2: STANDARDDEVIATION.JAVA	80
	APPENDIX C: SIMULATION.....	82
	APPENDIX C-1: NODE.JAVA	83
	APPENDIX C-2: NODECONTENT.JAVA.....	84
	APPENDIX C-3: BUILDTREE.JAVA	85
	APPENDIX C-4: READTREE.JAVA.....	90
	APPENDIX C-5: NOPREFETCH.JAVA	91
	APPENDIX C-6: BFS.JAVA	95
	APPENDIX C-7: USERPREFETCHMULTIPLEDOC.JAVA	102
	APPENDIX C-8: USERPREFETCHONEDOC.JAVA	110

List of Table, Figures and Algorithms

FIGURE 1-1. A PROXY SERVER AS A FIREWALL	2
FIGURE 1-2. SHARED CACHING PROXY	5
FIGURE 2-1. TOP TEN PREFETCHING.....	12
FIGURE 5-1. DESIGN WALK-THROUGH OF USER DIRECTED PREFETCH PROXY	19
FIGURE 5-2. USER DIRECTED PREFETCH STATUS WINDOW	21
FIGURE 6-1. URL WEB TREE	29
ALGORITHM. 6-1 WEB TREE REBUILD ALGORITHM.....	30
FIGURE 6-2. LOGICAL DIAGRAM OF A REBUILT WEB TREE	31
ALGORITHM 6-2. NO PREFETCHING SIMULATION ALGORITHM.....	32
ALGORITHM 6-3. BREATH-FIRST PREFETCHING ALGORITHM.....	33
ALGORITHM 6-4. USER DIRECTED PREFETCH ALGORITHM.....	35
TABLE 7-1. NO PREFETCHING SIMULATION RESULT.....	36
TABLE 7-2. BREATH-FIRST PREFETCH SIMULATION RESULT	37
TABLE 7-3. USER DIRECTED PREFETCH SIMUALTION RESULT FOR HIT RATE.....	39
TABLE 7-4. USER DIRECTED PREFETCH SIMULATION RESULT FOR SERVER NET LOAD	40
TABLE 7-5. USER DIRECTED PREFETCH SERVER NETWORK LOAD DISTRIBUTION	41

Chapter One

1. Introduction

The World Wide Web (WWW) can be viewed as a large distributed database, which stores an immense amount of information. Since its introduction, more and more people are using it as an efficient way of gathering information from various sources. A WWW User Survey has reported that, "84% of the users report that they consider access to the Web is indispensable" [LF98]. Since users expect the retrieved information to arrive quickly, long user delay may cause user discontent, which reduces the value of the WWW. Therefore, it is important to keep the user latency to a minimum, in order to encourage the use of the WWW.

The user latency depends on the transfer time of the data and the propagation delay. Since the workload of the Internet has been growing exponentially, both hardware and software must be built to address the increasing demand on it. Previous research has been done to improve user latency by caching and prefetching web documents. Both techniques are widely adopted by the proxy server, which is an agent that downloads the web documents on behalf of the user. The following sections give a brief introduction to the proxy server and its overall operation.

1.1 Proxy Server Overview

A proxy server may be configured to play two roles in a network. First, it can be used as an agent which forwards a user request to the appropriate web server. Secondly, it can also be used as a firewall which prevents unauthorized outsiders from gaining access to the internal network. The following diagram depicts the network setup of the proxy server.

Figure 1-1: A Proxy Server As A Firewall

The proxy server is an interface between the browser and the web server, and acts on behalf of the browser to forward Hypertext Transfer Protocol (HTTP) requests to the web server. The web server will then reply to the proxy with the document and a header indicating the return code; size of the reply; expiration date of the document; and other information for the browser to process the content in the reply. The proxy will then forward the information to the browser. The use of a proxy is common practice, because it acts as a gateway to the outside world and resolves many security issues. The proxy server normally is installed with a cache which can be shared among browsers, thus the cache can be better utilized among users.

1.2 The HTTP Protocol

The Internet is shared by many types of computers, such as Unix, Macintosh, and PC, so there is a need to use a machine independent protocol to represent the requests and the replies. The primary protocol that is used is called Hypertext Transfer Protocol (HTTP). HTTP is an ASCII based protocol which can be used to represent requests and replies. The following example shows a typical HTTP request from a client, and a response from a web server:

1) Request

```
GET http://www.datek.com HTTP/1.0
Referer: http://home.netscape.com/bookmark_4_05/datek.htm
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.07[jen] (X11;SunOS 5.5.1_sun4u;Navo)
```

Pragma: no-cache
Host: www.datek.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1, *.utf-8

2) Response

HTTP/1.0 200 OK
Data: Sat, 26 Jun 99 15:43:25 GMT
Last-Modified: Sat, 26 Jun 99 15:43:25 GMT
Server: 1
Content-type: text/html
Content-length: 300
Connection: close
[... content of the reply ...]

As shown above, the first line of the request indicates that the request is a GET method, which retrieves the home page of www.datek.com. Whereas, the reply shows the return code of 200, which indicates that the request has been served successfully, and the content of the requested document is appended in the response.

1.3 Caching

Since users may revisit information that has been requested before, information can be cached in the proxy server to reduce the transfer time and propagation delay. The cache can also be shared among multiple users, thus the caching proxy server only needs to fetch documents from the Internet once. All subsequent requests from users can be served by retrieving a copy of the data in the cache. While a non-sharing caching proxy is not as efficient as a sharing caching proxy, one study has shown that under some situations, the caching proxy is limited to a 30-50% of improvement [ASA*96].

The following shows a picture of cache sharing:

Figure 1-2. Shared Caching Proxy

1.4 Prefetching

Besides caching, another approach to further reduce user access latency is by prefetching. After a user downloads a web page, the user takes time to digest the information on the current page, thus there is usually a period of time before the user makes another request. The prefetching algorithm utilizes the user “digesting” time to prefetch web documents of interest to the user, so that these documents reside in the local cache before they are requested. By means of prefetching, web documents are downloaded before the user requests them. The prefetching approach also reduces the chances of having to wait on invalid URL requests like an obsolete link, which may happen due to the on-going changes to the web.

This thesis presents a prefetching technique which allows the user to select the data to be prefetched. A prototype was built to implement this algorithm and an analysis was done to evaluate the effect of various contributing factors, which may alter the performance of the prefetching technique. The following sections give a detailed background on this topic, and is followed by the design overview of the User Directed Prefetching Proxy. This thesis concludes with an analytical result of a simulated model, which supports that the User Directed Prefetch approach can greatly reduce server load.

Chapter Two

2. Related Work

A lot of research has been done to reduce user latency. The focus of this research can be categorized as: 1) caching, and, 2) prefetching web documents. Both techniques require installing a software module along with the proxy server. The following sections describe some of the previous work that has been done on caching and prefetching.

2.1 Caching

By means of caching, the web documents are stored locally to the user, thus reducing transfer time and propagation delay. To achieve this, any requested documents that do not already exist in the cache will be downloaded from the web server, and stored in a specified cache which is local to the user. As a result, subsequent requests from users can be more quickly served by retrieving a copy of the data locally stored in the cache.

There are two fundamental questions to caching: "What to cache?" and, "When to cache?". The answers to these questions are not trivial and are driven by different philosophies and network properties. For example, some may argue that large documents should not be cached, because the cache can make use of the same amount of space to store smaller documents, which will increase the hit rate. In that case, it is better off to store it in a cache even though it is not as frequently accessed as other

documents. However, the counter argument is that the cost of downloading the document from the server again is higher than the cost of keeping it in the cache, and therefore large documents should be cached in some situations.

2.1.1 LRU Algorithm

Since the size of the cache is limited, there is a need to discard documents that have expired or are not being requested, so that the cache is better utilized. At first glance, a simple Least Recently Used (LRU) algorithm, which discards the documents least recently referenced, seems to be sufficient. This algorithm is not suitable for web documents, however, despite the fact that the LRU algorithm is widely adopted in some file systems with fixed page size. This is due to a fundamental difference between the property of a file system memory page and a web document. While a memory page normally has a fixed size of 512 K, a web document comes in variable lengths. Therefore the penalty of a cache miss is not constant for a proxy cache. In that case, the algorithm may discard many small documents in order to gather space for a few large documents. As a result, the hit rate is greatly reduced.

To solve this problem, there is an improved version of the LRU algorithm, which is known as LRU-MIN.

2.1.2 LRU-MIN Algorithm

The difference between the LRU and LRU-MIN is that the LRU-MIN algorithm walks through the cache, gathers a list of least recently used documents and discards

documents, beginning from the largest one, while the LRU algorithm blindly discards the least recently used documents regardless of their sizes. Thus, a smaller document has a higher priority to be retained in the cache. The following is the LRU-MIN algorithm in four steps [ASA⁺96]:

1. Let $T = \text{size}(D)$, where D is the new document which we would like to store in the cache.
2. Get a list of documents of size $> T$
3. Run LRU on this list, until the list is empty or the free space is enough for D
4. If the list is empty and the free space is $< T$, set T to $T/2$ and repeat from step 2

2.1.3 Shared Caching Proxy

As shown in Figure 1-2, a Shared Caching Proxy is a caching proxy that is shared by multiple users. The advantage is that, for popular documents, only one copy is stored in the cache. As long as the document exists in the cache, requests from users can be served by retrieving a copy of the shared data in the cache. A research shows that stand-alone caching proxy does not improve latency [LF98]. Furthermore, the benefit of a cache is limited, because users tend to retrieve new data from the web. While a non-shared caching proxy is not as efficient as a shared caching proxy, as the number of client sharing the cache increases, the hit rate increases. However, one study suggests that the increase in hit rate is limited [DMF97]. Another study has shown that under some situations, the caching proxy is limited to a 30-50% improvement of hit rate [ASA⁺96].

2.2 Prefetching

Prefetching a document means that the document is downloaded to the local cache before the user actually reads the information on the page. While caching is targeted to reduce user latency on subsequent requests of a web document, prefetching reduces user latency on the first user request of the document. Compared to those on caching, the research on prefetching is relatively recent. This section gives a tour of some of these approaches and how they are different from each other.

2.2.1 Web Crawler

The Web Crawler is also known as: Web Robot, Web Wanderers or Web Spiders. The Web Crawler recursively downloads all documents referred by the embedded links of the current document. Each Web Crawler may use a different prefetching strategy to traverse the Web. Although the Web Crawler reduces latency, it may greatly increase server load due to the amount of information that it prefetches. Therefore, operators must carefully evaluate the impact during configuration and operation of the Web Crawler in order to allocate the bandwidth efficiently among applications. [Lie95]

2.2.2 Top 10 Prefetching

Another prefetching approach is to prefetch the top ten most popular documents. The idea is that since some documents are more popular than others, the algorithm provides

the client a way to identify popular documents and prefetch them without heavily increasing the server load.

In this approach, each participating web server calculates the ten most requested documents based on access logs. The proxy server also calculates a list of popular web servers. By combining the information from the proxy and the web server, the proxy server regularly prefetches the top ten documents from a configured number of popular web servers. As a result, when the client requests a popular document, the local copy in the proxy server is included in the reply. The advantage of this approach is that the client prefetches documents that are statistically proven to be the most popular. Therefore, the bandwidth is not wasted on prefetching unwanted web pages. Also, since the number of prefetching documents is limited to ten, unlike the Web crawler, the increase in server load is under control. The result shows that it can achieve a 40% hit rate with only a 10% increase in server load [MC98]. The following figure shows the interaction between the users, the proxy server and the web servers in Top 10 Prefetching:

Figure 2-1. Top Ten Prefetching

2.2.3 Intelligent Prefetching

This approach prefetches documents based on the user's previous access pattern [Lie95, DCW96]. It makes use of an intelligent agent, which prefetches anticipated requests based on the user's browsing history. Although the user's access pattern may not be deterministic, users normally traverse through hyperlinks. Based on this observation, this algorithm applies the user's access pattern to prefetch embedded links and embedded images of the current page. One concern about this approach is that it is almost impossible to predict an user's interest and that the user's interest may change. Some users are random walkers, i.e. their behavior is hard to predict. Since people use the web as a means to search for new information, previous patterns may not be useful. Studies suggest that only 3% of user requests are initiated from the hot-list or

the window history, and that only 2% of the web documents are either printed or saved [CP95].

2.2.4 Breath-first Prefetching

One of the concerns of prefetching is that it may prefetch documents prematurely, in other words, a lot earlier than the first time a user refers to it. In the worse case scenario, the prefetched information may expire before the user requests it. To solve this problem, this approach prefetches documents that are hyper-linked by the user's current browsing page. Since users often traverse the web by selected links referred in the current page, the chances of a premature prefetching decision is reduced. Unlike the Web Crawler, which is a recursive prefetching algorithm, this approach only downloads links that are directly referred to by the current page. Therefore, the increase in server load is limited. However, the traffic is still 4.12 times higher than a normal caching proxy, thus it requires a high bandwidth to achieve a good performance [CY97].

Chapter Three

3. Motivation

A lot of research has been done on reducing user latency through caching. As mentioned before, the hit rate of a caching proxy server is limited to 30%-50% [ASA*96]. The effectiveness of a caching proxy also depends on how many users are sharing it: The advantages of caching is reduced as the number of shared users decreases [DMF97, LF98].

One study has shown that even a simple Breath-first Prefetching proxy can have a hit rate 1.67 times higher than a caching proxy [CY97].

The problem with Intelligent Prefetching is that it takes time for an intelligent agent to learn about an user's behavior, and also, even once the behavior is learnt, the user's interest may have subsequently changed. Moreover, information may be prematurely prefetched. Another concern is traffic: Traffic induced by a simple Breath-first Prefetching proxy is 2.81 times higher than a normal proxy's, and 4.12 times higher than a caching proxy's [CY97]. The issues faced by certain home users are also relevant factors of this study. Home users that are still using a slow modem do not benefit from using a shared cache nor a high traffic prefetching scheme, so this study will demonstrate that the following prefetching technique will benefit users with low bandwidth high latency Internet access. Seeing the limitation of the caching proxy, our research is focused on developing a prefetching algorithm to eliminate some of the

disadvantages of the prefetching proxy mentioned. The goal is to introduce a prefetching technique that will allow the user to select what documents are to be prefetched, so that the bandwidth of the connection can be fully optimized.

∴

Chapter Four

4. Assumption

This section describes the assumptions of the research and discusses the validity of the assumptions.

4.1 Prefetch Text and Image

The design will only prefetch text or image files, through HTTP. This is because 80% of web requests are of HTTP type, the remaining 20% include cgi, ftp, etc. [CP95]. It is not relevant in this study to prefetch cgi scripts, because the processing time for a cgi script is prohibitive. Furthermore, studies [KMR95, WAB⁺96, and, Bra96] show that most of the HTTP requests are text documents (HTML, plain text, postscript files), and images (GIF, JPEG, RGB, X bitmap). HTTP constitutes most of the WWW traffic growth.

4.2 Embedded Links Traversal

Fifty-two percent of document requests are made through selecting embedded hyperlinks, and 41% are from the "back" command [CP95]. Therefore the evaluation presented in this thesis is based on the assumption that the user traverses the web site by selecting one of the embedded links in the current page, which points to pages within

the same web site. Since users do not generally request at random and jump back and forth between sites, the assumption will be deemed valid.

4.3 Infinite Cache

Since the focus of this research is prefetching, the effect of manipulating cache size will not be evaluated here. The evaluation assumes that the cache is infinitely large, and so the user may back-track without penalty.

4.4 Undefined Browsing Strategy

There are three types of user browsing strategies:

1. Directed search - user knows exactly which page to download.
2. General purpose - user is familiar with the site, and
3. Random - user does not have a specific objective.

This research will not focus on a specific strategy to test, since users often switch back and forth between strategies [CP95].

Chapter Five

5. Design Overview

To provide a mechanism for users to select the information which they want prefetched. A personal proxy server is designed to provide this capability. A personal proxy server is a scaled down version of a proxy server. It is installed on the user's machine and provides features like local caching and HTTP request forwarding to the web browser. Although a personal proxy server may also provide some extended features to the user, it mainly acts as a request forwarding agent, while access control, logging, and monitoring, is still handled by the proxy server.

5.1 Design Walk-through

There are two processes running on the user's workstation: the web browser, and the personal proxy server. The personal proxy server contains a local cache, which stores the documents that are retrieved from the proxy server. It also displays the status of the user's requests on a link status window. This window contains a list of documents that the user has selected to prefetch. The color of the requests reflects whether the request is in progress, and finished successfully or not. The following is a diagram which shows the Design Walk-through. It describes the components of the setup and the communication between them.

Figure 5-1. Design Walk-through of User Directed Prefetch Proxy

When the user selects an URL, the browser sends an HTTP request to the personal proxy (1). The personal proxy then registers the request to the request list, which is shown on a separate link status window (2). In this window, a request may be shown in three different colors: red, yellow, and green. Red indicates that the request has been forwarded and a failure has occurred. This may occur if the URL link has become obsolete, or that there is a communication failure during the download. A new request is shown in yellow, which indicates that the request is in process. A green light indicates that the request is being serviced and the result is stored in the cache; duplicated requests are discarded.

Before the proxy prefetches the document, a fake return code is returned to the browser so that the browser will not wait for the reply of the request document (3). This fake return code is code 204 (No Content). To the web browser, this means that the request is served successfully, but that the browser should remain on the current page. This allows the user to continue browsing the current page and to select other links to be prefetched. These links will also be added to the request list and forwarded to the proxy server concurrently (4). When the server replies (5), the proxy will update the cache [Kol99]. (6); look up the requested link from the request list; and update the color of the URL link status window according to the return code (7). All information related to the status window is stored persistently on the user's workstation, so that the status of these requests are shown on the next user's session. If a selected link shows up as green in the link status window, the user can select the link again from the browser. This will trigger an HTTP request to the proxy server (8). When the proxy sees the request and sees that the document is already in the cache, it will forward it to the browser (9).

As already stated, there is a separate window displaying the status of downloading links. A snapshot (see Figure 6-2) of the window is illustrated as follows: Indicated in red is an invalid link, which was selected by the user, and while the rest of the links were successfully downloaded, the yellow link indicates that there is a link being requested by the proxy.

No Name

Netscape: SCS – Course Pages

File Edit View Go Communicator

Back

Reload

Home

Search

Netscape

Print

Security

Bookmarks

Location: <http://www.scs.carleton.ca/courses>

Personal Bookmarks

Members

WebMail

Connections

E

Home Directories Search

Carleton University
School of Computer Science

[Undergraduate Course Web Pages](#)

[Computer Science Clinic](#)

[Index Page](#)

Figure 5-2. User Directed Prefetch Status Window

5.2 Design Issues

Since the personal proxy server is interoperable with all web browsers, the browser window can be a browser window for the Netscape Navigator, the Internet Explorer, etc. In order to communicate with the personal proxy server, the user only needs to change the proxy server IP address in the browser, to the address defined by the personal proxy server. As well, the user can switch between browsers during a user session without affecting the operation of the personal proxy server.

The choice of language is Java, which makes the personal proxy server platform independent. Furthermore, there are a lot of added features for processing an URL that is ready to be integrated for future enhancement of the proxy. Java also makes the implementation of multithreading easier. The proxy server is multi-threaded because the design adopts a "thread per request concurrency model". Thus, each URL request is mapped to a thread so that if there is an URL request that has a very long propagation delay, the latency for the other URL requests are not affected. This also eliminates the need for queuing up the requests if a single threaded model is adopted.

Chapter Six

6. Evaluation

One of the greatest challenges in this research is to evaluate the User Directed Prefetching technique. It requires setting a proper benchmark, gathering evaluation results and comparing the results with other approaches.

All of the referenced research in this thesis has used one of the methods below to evaluate the performance of the proposed solutions:

1) Real experimentation

The proposed solution is implemented as a real prefetching proxy server and is applied to an experimental site.

2) Simulation using log traces

Log traces from a real web server is collected, and the performance of the prefetching proxy is evaluated analytically based on the log traces.

To evaluate the performance based on a real experiment, a stable version of the proxy mentioned above must be implemented and placed in an actual environment, so that benchmarking can be done. The second approach is used to evaluate the technique under a simulated environment. This is because the implementation of the prototype is designed to handle a subset of usage, outlined in the assumption, and is not suitable to

place in an experimental site composed of various requests and requiring proper error handling.

6.1 Limitation

The probability of a user selecting an appropriate page is based on how well the web site is designed. This thesis does not evaluate the web site and ignores this factor; thus the log traces are provided by a web server, rather than a proxy server, and all requests are made to a single web site. As a result of these criteria, the user's pattern is not diversified, meaning the result may be different for users with different request patterns.

6.2 Evaluation Goal

The goal is to build a simulation which measures various performance metrics for different approaches. In the experiment, three approaches are evaluated: No Prefetching, Breath-first Prefetching, and User Directed Prefetching. While the No Prefetching case represents the worse case scenario, the Breath-first Prefetching represents the best case scenario. It is important to understand that there is no prefetching algorithm that is perfect. The Breath-first Prefetching algorithm is chosen to represent the best case scenario because this algorithm prefetches based on embedded links traversal, which is similar to the User Directed Prefetch algorithm. Moreover, it is known to have a relatively high hit rate.

6.3 Simulation

For the simulation, the trace-base approach was chosen to evaluate the performance based on actual user log traces. The results are gathered based on the following aspects:

- 1) hit rate (i.e. the number of requested documents that existed in the cache during the observation period.)
- 2) server network load (i.e. total bytes transferred, in order to show that the method works without inducing a heavy degradation on the server).

6.4 Web Traces

Since there is no way to trace which documents an user anticipates to prefetch, and what they finally request, the analysis can only be based on HTTP request log traces. The trace used here was collected by NASA Kennedy Space Center web server in Florida [NAS95]. The log was collected in August 1995, and contains 3,461,612 HTTP requests. The following shows a sample log entry:

```
user1.floyd.ses.carleton.ca - - [05/Aug/1995:17:43:50 -0400] "GET /shuttle/countdown  
/liftoff70.html HTTP/1.0" 200 5220.
```

It contains information on hostname, timestamp, HTTP request, return code and size of the reply. To simulate prefetching, all embedded links in an HTML document must be known, then the Breath-first algorithm can prefetch the embedded links.

6.5 URL Links Dependency

As shown in the previous section, the log traces do not give information on what the embedded links are, and only show individual HTTP requests. We could take each HTML request, go to the web site, find out the page, parse the page, and find out all embedded web links, and prefetch, based on the information gathered. However, the log traces were collected in 1995, and the “road map” of the web site has changed since then: Some documents requested in the log may no longer exist in the web site. The alternative is to find out the relationship between log entries, and determine whether the requested link is an embedded link in a previous request. Since the information given in the log traces is very limited, our methodology must conjecture the missing information from the log traces. The simulation uses an approach that is suggested in another paper [Bes96]. The following introduces the concept of embedded dependency and traversal dependency:

1. An **embedded dependency** theorizes that if the user is reading document D_i , the probability of the user requesting D_j within a short period of time is 1, then D_j is embedded in D_i . Thus, the user will always request D_j after D_i .
2. A **traversal dependency** correlates that if the user is at document D_i , the probability of the user requesting D_j within a period of time is $0 < x < 1$. Thus, the user may request D_j after visiting D_i .

To determine embedded dependency, if the user always request D_j after D_i within a very short period of time, we can conclude that D_j is embedded in D_i . However, a lot of the

embedded links are icons which could have been cached at the client cache. Therefore, even if D_j is embedded in D_i , the log traces may not correctly reflect that. To reduce the complexity of the simulation the prototype was built only to prefetch traversal links, and embedded dependency was not considered in the simulation. Coincidentally, large images on the NASA web site are not commonly embedded in an HTML document, but on a separate directory, which users need to download with a separate request. So, ignoring the embedded dependency does not impact the evaluation result greatly.

6.6 Session Delineation

Since users may go for coffee breaks, it is not fair to evaluate the performance with a long break between HTTP requests. This is because the algorithm has no knowledge about session breaks, and therefore, it will naively prefetch all traversal links of the current page during the long break. Foreseeing this, the log is sorted by users, and the mean and standard deviation of the time between HTTP requests issued by the same user is calculated. The idea is that, if the time between two consecutive requests from the same client is longer than 1.5 standard deviation, then the second request is treated as the beginning of the next session. However, the calculation of the NASA log shows that the mean session length is 18.01 minutes and the standard deviation is 178.39 minutes, which is too high to be used as a guideline to delineate sessions. As a result, we delineate the browsing session by 30 minutes, which means any request issued after 30 minutes is treated as a request from a new browsing session.

In summary, if the user browses for ten minutes, then goes for a coffee break and comes back for another ten minutes, it is counted as two browsing sessions. This definition is important when the log traces of a web server are used for evaluation. As well, this definition helps determine the length of period which the algorithm can be utilized to prefetch documents.

6.7 Web Tree Reconstruction

Now that page dependency has been defined, the following describes how to rebuild a web tree based on the log traces.

6.7.1 Log Traces Preprocessing

Since requests from users were interleaved, the input file contains more than 3 million data, which is too great to be sorted by users. Accordingly, it was broken into smaller files and then each one was sorted by users. Since the overhead of processing cgi script can be very high, the proxy does not forward cgi scripts request during prefetch. As a result, all cgi scripts requests were filtered from the log traces. The simulation also filters all failed requests. Therefore, the log only contains HTTP traces with a return code of 200 (OK) and 304 (not modified).

The traces are then fed to a program to rebuild the web tree. The web tree is a hash table of nodes, each containing: information of the HTTP request; the number of times it has been visited; and a list of links that has been requested after the HTTP request.

Figure 6-1. URL Web Tree

For the above example, the web tree contains a document, www.abc.com/a.html, of 3045 bytes. This page was requested 17 times by users. It contains two traversal links: www.abc.com/b.html and www.abc.com/c.html, which were requested 4 and 12 times respectively. Therefore, the probability of requesting [b.html](http://www.abc.com/b.html) after [a.html](http://www.abc.com/a.html) is $4/17$.

whereas the probability of requesting c.html after a.html is 12/17. This implies that the probability of not going further into the web tree along this branch is 1/17.

This web tree is built based on the following algorithm:

```

for each session
  for each request
    if the node entry for the current HTTP request does not exist
      create a node entry for the current HTTP request
    increment the referenceCount for the node
    look up previous requested document
    add current request document to the traversal links of previous request document
    add reference count for the previous node
  end for
end for

```

Algorithm. 6-1 Web Tree Rebuild Algorithm

The following gives an example of how a web tree is being built:

1. *user1.floyd.scs.carleton.ca* - - - [05/Aug/1995:17:43:50 -0400] "GET /shuttle/countdown/*liftoff1*.html HTTP/1.0" 200 4770
2. *user1.floyd.scs.carleton.ca* - - - [05/Aug/1995:17:48:24 -0400] "GET /shuttle/countdown/*liftoff2*.html HTTP/1.0" 200 5220
3. *user1.floyd.scs.carleton.ca* - - - [05/Aug/1995:17:53:40 -0400] "GET /shuttle/countdown/*liftoff3*.html HTTP/1.0" 200 8974
4. *user2.floyd.scs.carleton.ca* - - - [05/Aug/1995:18:13:37 -0400] "GET /shuttle/countdown/*liftoff2*.html HTTP/1.0" 200 5220

5. *user2.floyd.scs.carleton.ca* - - [05/Aug/1995:18:39:42 -0400] "GET /shuttle/countdown/**liftoff4.html** HTTP/1.0" 200 1904

The first record is the liftoff1.html document. Since it is the first entry, other nodes are not updated. The second record is liftoff2.html. When a new node is added to the tree, the parent node liftoff1.html is looked up from the existing tree, and liftoff2.html is added to the children list of liftoff1.html. The third record is liftoff3.html, which will be added to the children list of liftoff2.html in a similar manner. The fourth record is liftoff2.html, again. Since this node already exists, the reference count of this node is incremented by 1. The results are shown below:

Figure 6-2. Logical Diagram of a Rebuilt Web Tree

After the web site has been built, the same log traces are used to simulate the cases of

1. Random, 2. Smart Fetch, 3. Pretetch, and 4. User Directed Pretetch.

6.8 No Prefetching Simulation

Since No Prefetching is involved, the simulation is a simple algorithm that takes in the traces, and calculates the hit rate and total bytes transferred. As discussed in Section 4, this algorithm assumes the cache size is infinitely large. The cache in the personal proxy server is not shared among users, so the algorithm clears the cache before each browsing session. The following shows the algorithm that is used in the no prefetching simulation:

```
read in the NASA HTTP log traces
filter out unsuccessful log traces and cgi request
loop through each request
    if (new session)
        clear cache
        session count = session count + 1
    total request count = total request count + 1
    if requested document not exist in cache
        increment number of transferred bytes
        add requested document to cache
    else if request document exist in cache
        total hit rate = total hit rate + 1
```

Algorithm 6-2. No Prefetching Simulation Algorithm

6.9 Breath-first Prefetching Simulation

Like the No Prefetching simulation, the Breath-first Prefetching simulation requires knowledge about the web tree structure, so that it can prefetch the traversal links while the user is reading the information on the browser. This is where the pre-built web tree

comes into place. As mentioned, the log traces are pre-processed to construct a web tree. As shown in the following algorithm, while the user is reading, the prefetch algorithm will traverse the web tree and prefetch according to the Breath-first Prefetching algorithm. Again, the cache size is assumed to be infinitely large and is cleared before each browsing session. The following shows the algorithm:

```

read in the pre-built web tree
read in the NASA HTTP log traces
filter out unsuccessful log traces and cgi request
loop through each request
    session length = current request time - previous request time
    if (new session)
        clear cache
        session count = session count + 1
    else
        // prefetch document
        prefetch length = allowed time to prefetch
        loop through each traversal links in under the current document
            if prefetch length > time to prefetch next document
                decrement prefetch length
                increment prefetch bytes
        total request count = total request count + 1
        if requested document not exist in cache
            increment number of transfered bytes
            add requested document to cache
        else // request document exist in cache
            total hit rate = total hit rate + 1

```

Algorithm 6-3. Breath-first Prefetching Algorithm

6.10 User Directed Prefetch

The goal of the User Directed Prefetch is to better utilize the bandwidth to prefetch the correct page without overloading the connection. In other words, to find out the hit rate and the number of bytes transferred if the user were allowed to prefetch documents: 1. 2. 3. 4. The number of documents for the user to prefetch is limited in order to analyze how the algorithm reduces server network load. If the user is allowed to prefetch any number of documents, then the expected result will be very similar to the Breath-first Prefetch, and we will be unable to analyze the algorithm's reduction of server network load.

Since web pages have expiration dates, and if the algorithm prefetches prematurely, then the cache is not being used efficiently and the data may expire before the user reference. However, this model guarantees that this will not happen. This is because, the algorithm only prefetches immediate traversal links, and these links are very likely to be the next requested document.

In the simulation algorithm, a random probability between 0 and 1 is generated between each user request. The user is considered to select the correct prefetch link, if the generated probability is less than the configured probability. Therefore, the algorithm will look ahead in the log traces, and prefetch the next document referred by the user. On the other hand, if the random probability is greater than the configured probability, then the simulator will prefetch another link from the list of traversal links. As outlined above, there is a limit on the number of documents that a user may prefetch. While this

configurable prefetching threshold is not reached, the simulation will prefetch the remaining documents according to the Breath-first Prefetch algorithm. The following shows the algorithm used in the User Directed Prefetching simulation:

```

read in the pre-built web tree
read in the NASA HTTP log traces
filter out unsuccessful log traces and cgi request
loop through each request
    if (new session)
        clear cache
        session count = session count + 1
    else
        prefetch length = allowed time to prefetch
        loop through traversal links under the current document
            randomly generate the probability of user selecting a correct link
            if prefetch length > time to prefetch next document
                if (generated probability <= specified probability)
                    prefetch the next request document // user has select the correct page
                else
                    prefetch an unprefetched traversal link
                    decrement prefetch length
                    increment prefetch bytes
        total request count = total request count + 1
        if requested document not exist in cache
            increment number of transfered bytes
            add requested document to cache
        else // request document exist in cache
            total hit rate = total hit rate + 1

```

Algorithm 6-4. User Directed Prefetch Algorithm

Chapter Seven

7. Simulation Results

A simulation of each of the stated prefetching algorithms was implemented. Each simulation was run with various parameters using the filtered NASA log traces as an input. These traces were composed of 1,497,635 requests, made during 142,999 users' sessions. The number of prefetched bytes, transfer bytes and cache hits are logged during the simulation. The following sections describe the results of each of the simulations followed by an analysis of the results.

7.1 No Prefetching

The following table shows the performance of the No Prefetching simulation. Since the cache is cleared before each user session, a cache hit only occurs when the user requests a document that was previously requested during the same browsing session. As expected, the cache hit is fairly low.

Total request	Cache hit	Hit rate	Number of sessions	Number of requests	Bytes transferred
1,497,635	174,359	11.64	142,999	1,497,635	24,205,674,440

Table 7-1. No Prefetching Simulation Result

7.2 Breath -first Prefetch

The following table shows the results of the Breath-first Prefetching. As shown below, the hit rate shows an overall increase. The number of prefetched bytes depends on the time between user requests and the network access speed, so the simulation was run three times at various speeds. The results show that the hit rate is increased, while the transferred bytes are decreased.

	ISDN Kbps	28.8 Kbps	14.4 Kbps
Bytes transferred	9,392,930,471	18,579,804,155	21,941,619,687
Bytes prefetched	579,605,908,492	137,189,108,472	63,335,005,126
Total bytes sent	588,998,838,963	155,768,912,627	85,276,624,813
Number of document prefetched	21,238,957	7,336,374	4,284,848
Cache hit	1,159,056	799,056	618,912
Total request	1,497,635	1,497,635	1,497,635
Hit rate	77.392%	53.355%	41.326%

Table 7-2. Breath-first Prefetch Simulation Result

7.3 User Directed Prefetch

The following table shows the performance of the User Directed Prefetch Algorithm. Similar to the Breath First Prefetch simulation, the network access speed is a tunable parameter in this experiment. On top of this, there is another parameter - (probability) that represents how well a user selects a page. A probability of zero means that the list of documents that the user submits to the prefetch queue does not contain the document that the user will request next. On the other hand, a probability of one means that the list of documents that the user submits to the prefetch queue contains the document that

the user will request next. Moreover, the number of pages that allows the user to prefetch is limited. This restriction was set to one-to-four during the simulation. Therefore, the simulation was run with various network access speeds, select correct probability, and a prefetch limit, for a total of $(3 * 10 * 4) = 120$ times.

7.3.1 Hit Rate

The following table can be viewed as three sections, each containing results of a specific network access speed. Each section contains simulation results of prefetching one-to-four web documents. The percentage shown is the hit rate. The highlighted result shows that the configuration gives a similar hit rate result, when compared to the Breath First Prefetching. That is, the hit rate is close to 77% for ISDN, 53% for 28.8Kbps, and 41% for 14.4Kbps.

	ISDN				28.8Kbps				14.4Kbps			
prob.	A-1	A-2	A-3	A-4	B-1	B-2	B-3	B-4	C-1	C-2	C-3	C-4
0.1	31%	41%	49%	55%	26%	32%	37%	41%	22%	27%	31%	34%
0.2	35%	48%	57%	63%	28%	38%	44%	48%	25%	32%	36%	<u>40%</u>
0.3	39%	55%	64%	70%	31%	43%	49%	<u>53%</u>	27%	36%	<u>41%</u>	44%
0.4	43%	60%	69%	74%	35%	47%	<u>53%</u>	57%	30%	39%	44%	47%
0.5	47%	65%	73%	76%	38%	51%	56%	59%	32%	<u>43%</u>	47%	49%
0.6	52%	70%	75%	<u>78%</u>	41%	<u>54%</u>	58%	60%	35%	45%	49%	50%
0.7	57%	73%	<u>77%</u>	79%	45%	56%	60%	61%	38%	47%	50%	51%
0.8	62%	75%	78%	79%	48%	58%	60%	61%	<u>41%</u>	49%	50%	51%
0.9	67%	<u>77%</u>	78%	79%	<u>52%</u>	59%	61%	61%	44%	50%	50%	51%
1	72%	<u>77%</u>	78%	79%	56%	60%	61%	61%	47%	50%	51%	51%

Table 7-3. User Directed Prefetch Simulation Result for Hit Rate

7.3.2 Server Load

Similar to the previous table, the following table shows the server network load, which is shown as a percentage of the Breath-first Prefetch server network load described in Table 7-2. The configuration that was highlighted in the previous table is also highlighted in the table below:

Server Load ISDN					28.8Kbps				14.4Kbps			
prob.	A-1	A-2	A-3	A-4	B-1	B-2	B-3	B-4	C-1	C-2	C-3	C-4
0.1	6%	8%	10%	12%	20%	24%	27%	30%	34%	39%	42%	45%
0.2	6%	8%	10%	12%	19%	23%	27%	30%	34%	38%	41%	44%
0.3	6%	8%	10%	12%	19%	22%	26%	29%	33%	37%	40%	43%
0.4	6%	7%	9%	12%	19%	22%	25%	28%	33%	36%	40%	43%
0.5	5%	7%	9%	11%	18%	21%	25%	28%	32%	36%	39%	42%
0.6	5%	7%	9%	11%	18%	21%	25%	28%	32%	35%	39%	42%
0.7	5%	7%	9%	11%	18%	21%	24%	28%	32%	35%	39%	42%
0.8	5%	7%	9%	11%	17%	20%	24%	28%	31%	34%	39%	42%
0.9	5%	6%	9%	11%	17%	20%	24%	27%	31%	34%	38%	42%
1	5%	6%	9%	11%	17%	20%	24%	27%	30%	34%	38%	41%

Table 7-4. User Directed Prefetch Simulation Result for Server Network Load

It is clear from the results that the server network load is greatly reduced in the User Directed Prefetch. In the best case, the server network load of the simulation running with the fastest prefetching speed, ISDN, is only 8% of the Breath First simulation

network load. Even in the worst case scenario, the 14.4Kbps User Directed Prefetch simulation, the server network load is 44% of the Breath First simulation network load.

The following table shows the distribution of the server network load for the User Directed Prefetch with four prefetching documents. It is clear that the reduction in the server load is due to the reduction of the number of prefetching bytes. Moreover, it shows that the transferred bytes are also reduced with the increasing probability.

	ISDN	28.8Kbps			14.4Kbps				
Prob.	A-Prefetch	A-Transfer	A-Total	B-Prefetch	B-Transfer	B-Total	C-Prefetch	C-Transfer	C-Total
0.1	10%	176%	12%	19%	113%	30%	25%	104%	45%
0.2	10%	137%	12%	20%	100%	30%	26%	97%	44%
0.3	10%	109%	12%	20%	92%	29%	27%	92%	43%
0.4	10%	90%	12%	21%	86%	28%	27%	89%	43%
0.5	10%	78%	11%	21%	82%	28%	27%	86%	42%
0.6	10%	72%	11%	21%	80%	28%	27%	85%	42%
0.7	10%	68%	11%	21%	78%	28%	27%	84%	42%
0.8	10%	66%	11%	21%	78%	27%	27%	83%	42%
0.9	10%	65%	11%	21%	77%	27%	27%	83%	42%
1	10%	65%	11%	21%	77%	27%	27%	83%	41%

Prefetch & Transfer Bytes User Prefetch Vs Breath First Prefetch

Fig. 7-5. User Directed Prefetch Server Network Load Distribution

Chapter Eight

8. Conclusion

As pointed out in Section 3, the goal of the User Directed Prefetch is to provide an alternative prefetching technique that eliminates the problems that other prefetching techniques introduced.

This research demonstrates that unlike the Intelligent Prefetching, the User Directed Prefetch does not require a long time to learn the user's access pattern. As well, since the user selects the links from the current page, it is unlikely the algorithm will prefetch documents too in advance, before the user needs it. As a result, the cache is better utilized to store documents that are more likely to be requested before its expiration time. Table 7-3, User Directed Prefetch Simulation Result for Hit Rate, also shows that Intelligent Prefetching is less attractive for users with low speed transfer bandwidth. For example, for an user with 14.4Kbps transfer rates, who is allowed to prefetch four documents from the current link, if the probability for the user to select the correct page is 0.2, then the user has already achieved the same hit rate as the Breath First Prefetching. It is shown that the User Directed Prefetch can greatly reduce the server load.

The disadvantage of the Web Crawler, and Breath-first Prefetching is that they are relatively selfish algorithms, which almost blindly prefetch documents and greatly increase the server network load. As seen in Table 7-4, User Directed Prefetch

Simulation Result for Server Network Load. with User Directed Prefetch, the work load can be reduced by at least 50-90%, while maintaining the same hit rate as the Breath-first Prefetch. For example, in the case with ISDN, the hit rate is 77% in Breath-first Prefetch, and in Table 7-4, the User Directed Prefetch can achieve the same hit rate with just 6%-11% of Breath-first Prefetch's workload. Further analysis shows that the decrease in the work-load is due to the significant reduction on the amount of prefetching bytes.

In conclusion, we have shown that the User Directed Prefetch successfully eliminates some of the disadvantages of other prefetching techniques. It is a non-greedy algorithm, which allows users to fairly distribute bandwidth among processes in the workstation.

References

- [ASA⁺96] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox.
“Caching Proxies: Limitations and Potentials.” 4th International
Conference on the World-Wide Web. 1996: 119 – 134.
<<http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html>>
- [Bes96] A. Bestavros. “Speculative Data Dissemination and Service to Reduce
Server Load, Network Traffic and Service Time in Distributed
Information Systems.” Proceedings of the Twelfth International
Conference on Data Engineering. March. 1996: 180 – 187.
<<http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/icde/Bestavros96.html>>
- [Bra96] T. Bray. “Measuring The Web Computer Networks and ISDN Systems.
May. 1996: 993 – 1005.
<http://www5conf.inria.fr/fich_html/papers/P9/Overview.html>
- [CP95] L. D. Catledge and J. E. Pitkow. “Characterizing Browsing Strategies in
the World-Wide Web.” Computer Networks and ISDN Systems. Apr.
1995: 1065 – 1073.
<<http://www.fhg.de/www/www95/proceedings/papers/80/userpattern/80/UserPatterns/Paper4.formatted.html>>

- [CY97] K. Chinen and S. Yamaguchi. "An Interactive Prefetching Proxy Server for Improvement of WWW Latency." INET Conference Proceedings. 1997.
<http://www.isoc.org/inet97/proceedings/A1/A1_3.HTM>
- [DCW96] G. V. Dias, G. Cope, and R. Wijayarathne. "A Smart Internet Caching System." INET Conference Proceedings. 1996.
<http://info.isoc.org/isoc/whatis/conferences/inet/96/proceedings/a4/a4_3.htm>
- [DMF97] B. M. Duska, D. Marwood, and M. J. Feeley. "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches." USENIX Symposium on Internet Technologies and Systems Proceedings. Dec. 1997: 23 – 35.
<<http://www.cs.ubc.ca/spider/marwood/Projects/SPA/wwwap/wwwap.html>>
- [KMR95] T.T. Kwan, R. E. McGrath, and D. A. Reed. "User Access Patterns to NCSA's World Wide Web Server". Technical Report, University of Illinois at Urbana—Champaign, Department of Computer Science. Feb 1995.
- [K99] R. Kolar. "Smart Cache." 1999.
<http://www.netmag.cz/apps/nase/smartcache_e.html>

- [LF98] B. Liu and E. A. Fox. "Web Traffic Latency: Characteristics and Implications." *Journal of Universal Computer Science*. Sept. 1998: 763 – 778.
<http://www.iicm.edu/jucs_4_9/web_traffic_latency_characteristics/paper.html>
- [Lie95] H. Lieberman. "Letizia: An Agent That Assists Web Browsing." *Proceedings of the International Joint Conference on Artificial Intelligence*. Aug. 1995: 924 – 929.
<<http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-AAAI/Letizia.html>>
- [MC98] E. P. Markatos and C. E. Chronaki. "A Top 10 Approach for Prefetching the Web." *Internet Conference Proceedings*. 1998.
<http://www.isoc.org/inet98/proceedings/I1/I1_2.htm>
- [NAS95] "NASA-HTTP log traces." 1995.
<<http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html>>
- [WAB'96] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, and L. A. Rowe. "An Investigation of Documents from the World Wide Web." *Computer Networks and ISDN Systems*. May. 1996: 963 – 980.
<http://www5.conf.inria.fr/fich_html/papers/P7/Overview.html>

Appendix A: Personal Proxy Server

The following is a list of programs that implemented the personal proxy server, which is described in Section 5:

1. Proxy.java – A multi-threaded server that handles all URL requests. This is the main program of the person proxy server.
2. RequestHandler.java – Each URL request is handled by a separate request handler that is responsible for calling various utilities to download the requested document and updating the status window.
3. SelectedLinkRequest.java – This program is responsible for parsing an HTTP request.
4. SelectedLinksCanvas.java – This program is responsible for updating the status window.
5. PersistentRequestList.java – All information in the status window is written in a file so that the information is preserved for the next browsing session on the same site.

Appendix A-1: Proxy.java

```
// Proxy.java
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.net.*;
import java.util.*;
import java.io.*;
import java.io.IOException;

public class Proxy extends Applet implements ActionListener, Runnable {

    static final int PORT = 8081; // port used by the web browser
    static final int MAX_THREADS = 25; // max threads for each URL request
    private ServerSocket serverSocket_ = null; // socket for the browser
    private ThreadGroup httpRequestGroup_ = null; // thread group for requests
    private Thread selfThread_ = null; // main thread for the server itself

    private SelectedLinksCanvas selectedLinksCanvas_ = null; // GUI

    public void init() {

        // set the layout of the link status window
        setLayout(new BorderLayout());
        Panel panel = new Panel();
        add("North", panel);

        ScrollPane sp = new ScrollPane();
        selectedLinksCanvas_ = new SelectedLinksCanvas();

        sp.setSize(500, 600);
        sp.add(selectedLinksCanvas_);
        add("Center", sp);

        // Create a thread group to handle the requests
        httpRequestGroup_ = new ThreadGroup("HTTP-requests");

        // set up the proxy server that talks to netscape and smart cache
        serverSocket_ = new ServerSocket(PORT);
    }

    public void actionPerformed() {
        try {
            while (true) {
                Socket clientSocket = serverSocket_.accept();
                Thread t = new Thread(this, clientSocket);
                t.start();
            }
        } catch (IOException e) {
            // handle exception
        }
    }

    public void run() {
        // start the server
    }
}
```

```

        System.out.println( "Proxy::Init: " + e );
        System.exit(-1);
    }

    System.out.println("Server Started");

}

public void run() {
    try {
        while(true) {
            System.out.println("Proxy::run: Number of active threads: " +
                               httpRequestGroup_.activeCount() );
            if (httpRequestGroup_.activeCount() < MAX_THREADS ) {
                // listen from the web browser
                Socket clientSocket = serverSocket_.accept();
                try {

                    // browser talked to us, set up streams to I/O netscape
                    BufferedReader fromNetscape = new BufferedReader(
                        new InputStreamReader(
                            clientSocket.getInputStream()));
                    DataOutputStream toNetscape = new DataOutputStream(
                        new BufferedOutputStream(
                            clientSocket.getOutputStream()));

                    // create an object to handle the request and start the
                    // thread of the request processor
                    RequestHandler tmp =
                        new RequestHandler(clientSocket,
                                           fromNetscape,
                                           toNetscape,
                                           selectedLinksCanvas_);
                    new Thread(httpRequestGroup_, tmp).start();
                } catch (IOException e) {
                    System.out.println("Proxy::run(): " + e);
                    if (clientSocket != null) {
                        clientSocket.close();
                    }

                    // if all request handlers are busy
                }
                Thread.currentThread().sleep(100);
                System.out.println("Proxy::run: Threads: " +

```

```

        } catch ( InterruptedException e ) {
            System.out.println( "Proxy::run: Error in sleep: " + e );
            System.exit(-1);
        }
    }
}
} catch ( IOException e ) {
    System.out.println("Proxy::run: " + e);
} finally {
    try {
        if (serverSocket_ != null) {
            serverSocket_.close();
        }
    } catch ( IOException e ) {
        System.out.println("Proxy::run: " + e);
    }
}
}
}

```

```

public void start() {
    if (selfThread_ == null) {
        selfThread_ = new Thread(this);
        selfThread_.start();
    }
}

```

```

public void actionPerformed(ActionEvent e) {
    //
}

```

```

static class MyAdapter extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}

```

```

public static void main(String args[] ) {
    //
}

```

```

    Frame proxyFrame = new Frame("Prefetching URL Links");
    Proxy myProxy = new Proxy();

```

```

    myProxy.init();
    myProxy.start();
}

```

```
//default, don't really need this line
proxyFrame.setLayout(new FlowLayout());

proxyFrame.add("Center", myProxy);
proxyFrame.pack();
proxyFrame.setSize(600, 800);
proxyFrame.show();

proxyFrame.addWindowListener(new MyAdapter());
}
}
```

Appendix A-2: RequestHandler.java

// RequestHandler.java

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
import java.text.*;

class RequestHandler extends Thread {
    static final int PORT = 8080; // Port used by the cache
    static final String PROXYHOSTNAME = "localhost";
    static final Color RED = new Color(255, 0, 0);
    static final Color YELLOW = new Color(255, 255, 0);
    static final Color GREEN = new Color(0, 255, 0);
    static final String CLASSNAME = "RequestHandler";

    static boolean log_ = true; // turn on logging
    private static int requestCount_; // URL request counter
    private Socket netscapeToProxySocket_ = null; // socket from browser
    private Socket forwardingSocket_ = null; // socket to cache
    private boolean methodSearched_ = false; // false if URL method is unknown
    private StringTokenizer httpTokenizer_; // used when parsing the URL
    private StringTokenizer headerTokenizer_; // used when parsing the URL
    private SelectedLinkRequest httpRequest_ = null; // stored the parsed URL

    private Vector header_; // stored the HTTP header
    private boolean readyToNetscape_ = false; // not ready to send data to browser
    private boolean duplicatedRequest_ = false; // submitted request is not duplicated

    private DataOutputStream logFile_; // log file

    private BufferedReader fromNetscape_; // reader from browser
    private DataOutputStream toNetscape_; // writer to browser
    private DataInputStream fromSmartCache_; // reader from cache
    private PrintWriter toSmartCache_; // writer to cache
    private SelectedLinksCanvas selectedLinksCanvas_; // GUI
    private static DateFormat formatter_ =
        new SimpleDateFormat("EEE, dd MMM yy HH:mm:ss GMT", Locale.US);

    public RequestHandler(
        Socket browserSocket,
        Socket cacheSocket,
```

```

        BufferedReader    fromNetscape,
        DataOutputStream  toNetscape,
        SelectedLinksCanvas selectedLinksCanvas ) throws IOException {

// initialization
selectedLinksCanvas_ = selectedLinksCanvas;
netscapeToProxySocket_ = socket;
fromNetscape_ = fromNetscape;
toNetscape_ = toNetscape;
requestCount_++;

// setup log file for logging
if ( log_ ) {
    // create log directory if it does not exist
    if ( !(new File(".") + File.separator + "log").exists()) {
        new File("log").mkdirs();
    }
    // open log file
    logFile_ = new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream( "." + File.separator + "log" +
                File.separator + requestCount_)));
}
}

public void SW_error( String functionName, String logMessage ) {

    // print out error messages
    try {
        System.out.println requestCount_ + "logging " + logMessage );
        if ( log_ ) {
            logFile_.writeBytes requestCount_ + "-" + CLASSNAME + ":" +
                functionName + ":" +
                logMessage + "\r\n" );
        }
    } catch ( IOException e ) {
        System.out.println "RequestHandler::SW_error: log IO exception" );
    }

    // Handle requests from the browser
    public boolean readFromNetscape() {
        boolean returnValue = false;
        Vector persistentList = null;
    }
}

```



```

while( true ) {

    // read HTTP request line by line
    String str = fromNetscape_.readLine();

    // if I interrupt connection by hitting escape
    // the str will be null and give me null exception
    // I need to handle it by returning i.e. abort this thread
    if (str == null) {
        return false;;
    }

    if ( str.equals( null ) ) {
        System.out.println( "Echoing to Server: NULL is hitted!" );
        break;
    }

    String urlString = null;

    // check if the request is a GET, which means that the
    // user is requesting the proxy server to get a document
    // from the Web
    if ( !methodSearched_ ) {
        methodSearched_ = true;
        httpTokenizer_ = new StringTokenizer( str );
        String methodToken = next( httpTokenizer_ );
        if (methodToken.equals( "GET" ) ) {
            // preload page and image in that page
            String urlToken = next( httpTokenizer_ );
            if ( !( urlToken.toLowerCase().endsWith("gif" ) ) &&
                !( urlToken.toLowerCase().endsWith("jpg" ) ) &&
                !( urlToken.toLowerCase().endsWith("mpg" ) ) ) {
                synchronized(this) {
                    httpRequest_ = new SelectedLinkRequest( urlToken, Color.yellow );
                    PersistentRequestList persistentRequestList =
                        new PersistentRequestList();

                    // check if request exist already
                    SelectedLinkRequest existingRequest =
                        persistentRequestList.getElement( httpRequest_ );
                    if ( existingRequest == null ) // request not exist
                        persistentList =
                            ( new PersistentRequestList() ).addElement( httpRequest_ );
                    // update status on window, the link shows in Yellow

```

```

        selectedLinksCanvas_.update( httpRequest_.getHost(), persistentList );
    } else { // request exist
        if ( existingRequest.getStatusColor().equals( GREEN ) ||
            existingRequest.getStatusColor().equals( RED ) ) {
            // request was served before, reshow the status
            readyToNetscape_ = true;
            if ( existingRequest.getStatusColor().equals( RED ) ) {
                //
                httpRequest_.setStatusColor( Color.red );
                persistentList =
                    ( new PersistentRequestList() ).updateElement( httpRequest_ );
                selectedLinksCanvas_.forceUpdate( httpRequest_.getHost(),
                    persistentList );
            }

        } else if ( existingRequest.getStatusColor().equals( YELLOW ) ) {
            // I am already serving that request. exit the loop
            duplicatedRequest_ = true;
        }
    } // if request exist
} else { // get non html
    readyToNetscape_ = true;
}

} else { // non GET method
    readyToNetscape_ = true;
}

}

if ( !duplicatedRequest_ ) {
    toSmartCache_.println( str );
    toSmartCache_.flush();
    String out = str + "END";
    if ( out.equals( "END" ) ) {
        // finish reading from netscape
        break;
    }
} else {
    // finish reading from netscape
    break;
}

while still have data from netscape
catch ( IOException e ) {
    System.out.println( "RequestHandler:readFromNetscape: IOException" );
}

```

```

    }

    return true;
}

public void readFromSmartCache() throws IOException {
    // read from smart cache
    boolean interpretHeader = false;
    Vector persistentList = null;

    while ( true ) {
        byte inputBuffer[] = new byte[512];
        int inputSize;
        inputSize = fromSmartCache_.read(inputBuffer);
        if (inputSize == -1) {
            break;
        }
        if ( httpRequest_ != null && !interpretHeader ) {

            interpretHeader = true;
            String header = null;
            header = new String( inputBuffer );
            headerTokenizer_ = new StringTokenizer( header );
            String httpVersionToken = next( headerTokenizer_ );
            String httpRc = next( headerTokenizer_ );

            synchronized(this) {
                // check the return code
                if ( httpRc.equals("200") || httpRc.equals("304") || httpRc.equals("302") ) {

                    // the return code is OK
                    // update color to green in persistent list
                    if ( !readyToNetscape_ ) {
                        try {
                            // give some time to view the update
                            System.out.println
                                ("RequestHandler:readFromSmartCache: sleep 5 seconds" );
                            sleep(10000);
                        } catch ( InterruptedException e ) {
                            System.out.println
                                ("RequestHandler:readFromSmartCache: sleep exception" );
                        }

                        // update the status color to green in the status window

```

```

httpRequest_.setStatusColor( Color.green );
persistentList =
    (new PersistentRequestList()).updateElement( httpRequest_ );

} else { // NOT OK re update color to red in persistent list
    httpRequest_.setStatusColor( Color.red );
    persistentList =
        ( new PersistentRequestList()).updateElement(httpRequest_ );
}

if ( readyToNetscape_ ) {
    selectedLinksCanvas_.forceUpdate( httpRequest_.getHost(), persistentList );
} else {
    selectedLinksCanvas_.update( httpRequest_.getHost(), persistentList );
}
}

if ( readyToNetscape_ ) {
    toNetscape_.write(inputBuffer, 0, inputSize);
} else {
    break;
}

} // while still have data from smartCache. send it to netscape

if (readyToNetscape_) {
    toNetscape_.flush();
}

}

public void run() {
    String myName = "run";
    SW_error( myName, "Running" );

    try
    {
        connectToSmartCache();
        SW_error( myName, "Cannot connect to smartcache" );
    }
    try
    {
        netscapeToProxySocket_.close();
    }
    catch (IOException e2) {
        SW_error( myName, "final IO exception" );
    }
}

```

```

    }
    return;
}

if (!readFromNetscape()) {
    return;
}

// if the request has fully download read from smart cache
// otherwise immediate reply

if ( !readyToNetscape_ || duplicatedRequest_ ) {
    // Sending 204 header to Netscape
    sendImmediateResponse();
}

if ( !duplicatedRequest_ ) {
    readFromSmartCache();
}

} catch (IOException e1) {
    System.out.println("RequestHandler::run: io exception");
} finally {
    try {
        if (netscapeToProxySocket_ != null) {
            netscapeToProxySocket_.close();
        }
        if (forwardingSocket_ != null) {
            forwardingSocket_.close();
        }
    } catch (IOException e2) {
        System.out.println("RequestHandler::run: final IO exception");
    }
}

if (netscapeToProxySocket_ != null) {
    netscapeToProxySocket_.close();
}

if (forwardingSocket_ != null) {
    forwardingSocket_.close();
}

} catch (IOException e2) {
    System.out.println("RequestHandler::run: final IO exception");
}

```

```

}

public boolean connectToSmartCache() {
    boolean rc = false;
    try {
        InetAddress proxyAddr = InetAddress.getByName(PROXYHOSTNAME);
        forwardingSocket_ = new Socket(proxyAddr, PORT);

        fromSmartCache_ =
            new DataInputStream(
                new BufferedInputStream(
                    forwardingSocket_.getInputStream()));
        toSmartCache_ =
            new PrintWriter(
                new BufferedWriter(
                    new OutputStreamWriter(
                        forwardingSocket_.getOutputStream()), true);

        System.out.println("RequestHandler::connectToSmartCache_ success");
        rc = true;
    }
    catch ( IOException e ) {
        try {
            if (forwardingSocket_ != null) {
                forwardingSocket_.close();
            }
        }
        catch ( IOException e4 ) {
        }
        System.out.println("RequestHandler: connect to smart cache failed");
        rc = false;
    }
    return rc;
}

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    }
    else {
        return "";
    }
}

```

```

// create an HTTP header
public void makeHeader(int rc, String contentType, String location,
                      int size, long lastModified, long expires) {
    header_ = new Vector();
    Date d = new Date();
    header_.addElement("Date: " + formatter_.format(d));
    if (lastModified!=0) {
        header_.addElement("Last-Modified: " +
                           formatter_.format(new Date(lastModified)));
    }
    if (expires>d.getTime()) {
        header_.addElement("Expires: " +
                           formatter_.format(new Date(expires)));
    }
    if (location!=null) {
        header_.addElement("Location: " + location);
    }
    header_.addElement("Server: " + 1);
    if (contentType!=null) {
        header_.addElement("Content-type: " + contentType);
    }
    if(size>0) {
        header_.addElement("Content-length: " + size);
    }
    header_.addElement("Connection:close");
    header_.addElement("Proxy-Connection: close");
}

// send an HTTP header with empty content to browser
// we only need to send the status code in the header
public void sendHeader(int returnCode) {
    try {
        toNetscape_.writeBytes("HTTP/1.0 " + returnCode + " OK\r\n");
        System.out.println("HTTP/1.0 " + returnCode + " OK");
        int header_cache=header_.size();
        for(int i=0;i<header_cache;i++) {
            toNetscape_.writeBytes((String)header_.elementAt(i));
            System.out.println((String)header_.elementAt(i));
            toNetscape_.writeBytes("\r\n");
        }
        toNetscape_.writeBytes("\r\n");
        toNetscape_.flush();
    } catch (IOException ioe) {}
    try {
        toNetscape_.close();
    }
}

```

```

    } catch (Exception e) {
    }
}

}

// send a header to the browser. This will let the browser
// know that the URL request has been acknowledged
public void sendImmediateResponse() {
    String myName = "sendImmediateResponse";

    makeHeader(204, "text/html", null, 3, new Date().getTime(), 0);
    sendHeader(204);
    try {
        if (netscapeToProxySocket_!=null) {
            netscapeToProxySocket_.close();
        }
        if (duplicatedRequest_) {

            if (forwardingSocket_!=null) {
                forwardingSocket_.close();
            }
        }
    } catch (IOException e) {
        SW_error( myName, "final IO exception");
    }
}
}
}

```


Appendix A-3:

SelectedLinkRequest.java

```
// SelectedLinkRequest.java

import java.util.*;
import java.awt.*;
import java.io.*;

public class SelectedLinkRequest implements Serializable {

    private String url_;
    private Color  statusColor_;
    private String[] parsedResult_; // parsed output of the URL

    public SelectedLinkRequest( String url, Color statusColor ) {
        setUrl(url);
        setStatusColor(statusColor);
        parsedResult_ = parseUrl();
    }

    public String getUrl() {
        return url_;
    }

    public Color getStatusColor() {
        return statusColor_;
    }

    public String getHost() {
        return parsedResult_[0];
    }

    public String getPort() {
        return parsedResult_[1];
    }

    public String getDir() {
        return parsedResult_[2];
    }

    public String getIndex() {
        return parsedResult_[3];
    }
}
```

```

    return parsedResult_[3];
}

public String getProtocol() {
    return parsedResult_[4];
}

public String[] getParsedResult() {
    return parsedResult_;
}

public void setUrl( String url ) {
    url_ = url;
}

public void setStatusColor( Color statusColor ) {
    statusColor_ = statusColor;
}

// As mentioned in the acknowledgement, the following code is
// partially obtained from the SmartCache [Kol99]
public String[] parseUrl() {
    /* parse the URL request into the following
       web host
       web port
       directory storing the document
       file name of the document
       protocol used
    */

    String[] res = new String[5];
    res[3] = "";
    int i, j;
    // search for :
    i = url_.indexOf(':', 0);
    // break request by colon
    // http:
    res[4] = url_.substring(0, i); // http
    int seven = res[4].length()+3;
    if (seven==7) {
        // the format is http:// 7 characters
        char c;
        c = res[4].charAt(0);
        if (c=='h' || c=='H') res[4]=null;
    }
}

```

```

// find index to the next '/'
i = url_.indexOf('/',seven);
if(i == -1) {
    the host name is not specified
    url_+="7";
    i=url_.length()-1;
}

// find index to the next ':'
j = url_.indexOf(':', seven);
if (j!=-1 && j<i) {
    // port and host name are specified
    // substring from 7 to next, i.e. host name, port
    res[0] = url_.substring(seven, j);
    res[1] = url_.substring(j+1, i);
    if (res[1].equals("80")) {
        res[1] = null;
    }
} else {
    res[0] = url_.substring(seven, i);
}

j = url_.length()-1;
char v[];
v = new char[j+1];
url_.getChars(i,j+1,v,0);

// skip the following characters
loop1:for(int zz=i;zz<=j;zz++) {
    switch(v[zz]) {
        case '/':
        case ':':
        case '=':
        case '?':
            i=zz; break loop1;
        case '#':
            uri =url_.substring(0,zz); break loop1;
    }
}

// parse directory, filename
i=url_.lastIndexOf('/');
res[2] = url_.substring(i+1);
res[3] = url_.length()-res[3];res[3]=url_.substring(j+1);

```

```
    return res;
}

}
```

...

Appendix A-4:

SelectedLinksCanvas.java

```
// SelectedLinksCanvas.java
```

```
import java.awt.*;
import java.applet.Applet;
import java.util.*;
import java.io.*;
```

```
class SelectedLinksCanvas extends Canvas {
    private Vector requestQueue_ = null; // request queue
    int charHeight_; // window height
    int charWidth_; // window width
    Font defaultFont_; // font used
    String webServer_ = null; // web server of the URL requests

    public SelectedLinksCanvas() {

        // set fonts used in status window
        String[] fontNames = getToolkit().getFontList();
        defaultFont_ = new Font(fontNames[0], Font.PLAIN, 16);
        FontMetrics fm =
            Toolkit.getDefaultToolkit().getFontMetrics( defaultFont_ );
        charHeight_ = fm.getHeight() + 3;
        charWidth_ = fm.getMaxAdvance() + 4;
        System.out.println("charwd " + charWidth_);
        requestQueue_ = new Vector();
        setSize( charWidth_ * 48 + 600,
            charHeight_ * requestQueue_.size() * 10 + 10);

        // update the canvas only if the current is empty or
        // the input is the same site as the current
        public void update( String webServer, Vector requestQueue ) {
            System.out.println("SelectedLinksCanvas: update: " +
                "current " + webServer_ + "new " + webServer);

            if (webServer_ == null)
                // the status panel is empty
                webServer_ = webServer;
            else if (webServer_ != webServer)
                requestQueue_ = requestQueue;
        }
    }
}
```

```

        repaint();
    } else if (webServer_.equals(webServer)) {
        // the status panel has the current http site
        requestQueue_ = requestQueue;
        repaint();
    }
}

```

```

// update the canvas forcefully
public void forceUpdate( String webServer, Vector requestQueue ) {
    webServer_ = webServer;
    requestQueue_ = requestQueue;
    repaint();
}

```

```

// repaint the window
public void updateCanvas() {
    repaint();
}

```

```

// paint the window
public void paint(Graphics g) {

```

```

    setSize( charWidth_ * 48 + 600,
             charHeight_ * requestQueue_.size() * 10 + 10);

```

```

    int y = 20;
    for (int i = 0; i < requestQueue_.size(); i++) {
        SelectedLinkRequest selectedLinkRequest_ =
            ( SelectedLinkRequest ) requestQueue_.elementAt( i );
        g.setColor( selectedLinkRequest_.getStatusColor() );
        g.drawString( selectedLinkRequest_.getUrl(), 10, y );
        y += charHeight_;
    }
}

```

Appendix A-5:

PersistentRequestList.java

```
// PersistentRequestList.java
```

```
import java.awt.*;
import java.util.*;
import java.util.zip.*;
import java.io.*;
```

```
public class PersistentRequestList {
    Vector persistentList_;
    final static int SWAP_LEVEL1_DIRS = 4;
    final static int SWAP_LEVEL2_DIRS = 4;
    final static int MAX_FILENAME = 80;
```

```
    public void PersistentRequestList() {
    }
```

```
    // If the request has not been served before, we need to add a
    // new element in the persistent status log
```

```
    public Vector addElement(SelectedLinkRequest request) {
        try {
            String[] result = request.getParsedResult();
            String cacheInfoLog = getLocalDir("store", result[0], result[1], result[4]);
            StringTokenizer cacheInfoLogTokenizer =
                new StringTokenizer(cacheInfoLog);
```

```
            // create directory if needed
            // the directory may not exist if the user has never request
            // the URL from that particular web server
            // the format of the directory is level1/level2/level3
            String level1Directory =
                new String(cacheInfoLogTokenizer.nextToken()
                    + File.separator +
                    cacheInfoLogTokenizer.nextToken() + File.separator +
                    cacheInfoLogTokenizer.nextToken());
            if (!File.exists(level1Directory))
                File.mkdirs(level1Directory);
```

```
            String level2Directory = new String(level1Directory + File.separator +
                cacheInfoLogTokenizer.nextToken() + File.separator +
                cacheInfoLogTokenizer.nextToken());
```

```

if (!(new File(level2Directory).exists())) {
    new File(level2Directory).mkdirs();
}
String level3Directory = new String(level2Directory + File.separator +
    cacheInfoLogTokenizer.nextToken(File.separator));
if (!(new File(level3Directory).exists())) {
    new File(level3Directory).mkdirs();
}

// read in vector
try {
    ObjectInputStream in =
        new ObjectInputStream(
            new FileInputStream(cacheInfoLog));
    persistentList_ = (Vector) in.readObject();
    in.close();
} catch (FileNotFoundException e) {
    System.out.println(
        "PersistentRequestlist::addElement: file not exist, creating vector");

    persistentList_ = new Vector();
} catch (EOFException e) {
    System.out.println("PersistentRequestlist::addElement:EOF creating vector");
    persistentList_ = new Vector();
}

// check if the object is unique in the list
// i.e. add it only if it does not exist
boolean requestExist = false;
SelectedLinkRequest modifyRequest;
for (int i=0; i<persistentList_.size(); i++) {
    modifyRequest = (SelectedLinkRequest) persistentList_.elementAt(i);
    if (modifyRequest.getUrl().equals(request.getUrl())) {
        requestExist = true;
        break;
    }
}

if (!requestExist) {
    //safety checks
    if (persistentList_ == null) {
        System.out.println("PersistentRequestList::addElement: " +
            "null persistentrequestlist");
    }
}

```



```

        System.out.println("PersistentRequestList::addElement: " +
            "null request");
    }

    // add element
    persistentList_.addElement(request);

    // output elements back to the file
    ObjectOutputStream out =
        new ObjectOutputStream(
            new FileOutputStream(cacheInfoLog));
    out.writeObject(persistentList_);
    out.close();
    System.out.println("PersistentRequestList::addElement: " + cacheInfoLog +
        "added" + request.getHost() + cacheInfoLog);
}
} catch (Exception e) {
    e.printStackTrace();
}
return persistentList_;
}

// return the whole persistent list from file
public Vector getPersistentList(SelectedLinkRequest request) {
    try {
        String[] result = request.getParsedResult();
        String cacheInfoLog = getLocalDir(
            "store", result[0], result[1], result[4]);
        System.out.println("PersistentRequestList::getPersistentList " + cacheInfoLog);

        ObjectInputStream in =
            new ObjectInputStream(
                new FileInputStream(cacheInfoLog));
        persistentList_ = (Vector) in.readObject();
        in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return persistentList_;
}

// assume that element already exist in persistent list
// return the element
public SelectedLinkRequest getElement(SelectedLinkRequest request) {
    for (int i = 0; i < persistentList_.size(); i++) {
        SelectedLinkRequest element = (SelectedLinkRequest) persistentList_.get(i);
        if (element.getHost().equals(request.getHost()) &&
            element.getPort().equals(request.getPort()) &&
            element.getPath().equals(request.getPath())) {
            return element;
        }
    }
    return null;
}
}

```

```

String[] result = request.getParsedResult();
String cacheInfoLog = getLocalDir("store", result[0], result[1], result[4]);
System.out.println("PersistentRequestList::updateElement " + cacheInfoLog);
// read in persistent list
ObjectInputStream in =
    new ObjectInputStream(
        new FileInputStream(cacheInfoLog));
persistentList_ = (Vector) in.readObject();
in.close();

```

```

// update element in persistent list
SelectedLinkRequest modifyRequest;
for (int i=0; i<persistentList_.size(); i++) {
    modifyRequest = (SelectedLinkRequest) persistentList_.elementAt(i);
    if (modifyRequest.getUrl().equals(request.getUrl())) {
        modifyRequest.setStatusColor(request.getStatusColor());
        break;
    }
}
ObjectOutputStream out =
    new ObjectOutputStream(
        new FileOutputStream(cacheInfoLog));
out.writeObject(persistentList_);
out.close();
} catch (Exception e) {
    e.printStackTrace();
}
return persistentList_;
}

```

```

public boolean contains(SelectedLinkRequest request) {

```

```

    boolean re = false;

```

```

    try
    {
        String[] result = request.getParsedResult();
        String cacheInfoLog = getLocalDir("store", result[0], result[1], result[4]);
        System.out.println("PersistentRequestList::contains " + cacheInfoLog);
        if (new File(cacheInfoLog).exists()) {
            // persistent file exists, so read vector from file
            ObjectInputStream in =
                new ObjectInputStream(
                    new FileInputStream(cacheInfoLog));
            persistentList_ = (Vector) in.readObject();
            SelectedLinkRequest tempRequest;

```

```

        for (int i=0; i<persistentList_.size(); i++) {
            currentRequest = (SelectedLinkRequest) persistentList_.elementAt(i);
            if (currentRequest.getUrl().equals(request.getUrl())) {
                rc = true;
                break;
            }
        }
        in.close();
    }
} catch (Exception e) {
    e.printStackTrace();
}
return rc;
}

```

```

// get a particular element from the Persistent list
public SelectedLinkRequest getElement(SelectedLinkRequest request) {

```

```

    boolean found = false;
    SelectedLinkRequest currentRequest = null;

```

```

    try {
        String[] result = request.getParsedResult();
        String cacheInfoLog = getLocalDir("store", result[0], result[1], result[4]);
        System.out.println("PersistentRequestList::getElement " + cacheInfoLog +
            request.getUrl());
        if ((new File(cacheInfoLog)).exists()) {
            // persistent file exist, so read vector from persistent file

```

```

            ObjectInputStream in =
                new ObjectInputStream(
                    new FileInputStream(cacheInfoLog));

```

```

            persistentList_ = (Vector) in.readObject();
            for (int i=0; i<persistentList_.size(); i++) {
                currentRequest = (SelectedLinkRequest) persistentList_.elementAt(i);
                if (currentRequest.getUrl().equals(request.getUrl())) {
                    found = true;
                    break;
                }
            }

```

```

            in.close();

```

```

        } catch (Exception e) {
            e.printStackTrace();

```

```

    }
    if (found) {
        return currentRequest;
    } else {
        return null;
    }
}

}

public void removeElement(SelectedLinkRequest request) {

// get the local directory of the cacheLogInfo
public String getLocalDir(String cacheDir, String host, String port, String proto) {
    StringBuffer result = new StringBuffer(MAX_FILENAME);
    result.append(cacheDir);
    int i;
    CRC32 crc = new CRC32();

    host = host.toLowerCase();
    crc.update(host.getBytes());
    long unit = ((long) 1<<32)/(SWAP_LEVEL1_DIRS*SWAP_LEVEL2_DIRS);
    i=(int)(crc.getValue()/unit);
    result.append(File.separator+(i/SWAP_LEVEL2_DIRS)+
        File.separator+(i-SWAP_LEVEL1_DIRS
            * (i/SWAP_LEVEL2_DIRS))+File.separator+host);
    if (port!=null) {
        result.append("_");
        result.append(port);
    }
    if (proto!=null) {
        result.append("^");
        result.append(proto);
    }
    result.append(File.separator + "cacheLogInfo");
    return result.toString();
}

// for debugging only
public static void main(String[] args) {
    SelectedLinkRequest request = new SelectedLinkRequest(
        "http://www.nes.aunz.edu/index.html", Color.red);
    PersistentRequestList c = new PersistentRequestList();
    SelectedLinkRequest request1 =
        new SelectedLinkRequest(

```

```
        "http://www.ncsa.uiuc.edu/index.html", Color.red);  
    SelectedLinkRequest req2 = c.getElement(request);  
    System.out.println( "req2" + req2.getStatusColor() );  
    }  
}
```

Appendix B: Traces Statistics

The following is a list of programs that calculates the session length and standard deviation of the log traces:

1. `SessionLength.java` – Calculates the mean session length.
2. `StandardDeviation.java` – Calculates the standard deviation of the session length.

Appendix B-1: SessionLength.java

```
// SessionLength.java
```

```
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class SessionLength {
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormat sm_dateFormatter =
        (DateFormat) sm_dateFormat;
    public long m_totalSessionLength = 0;
    public long m_sessionCount = 0;
```

```
    public SessionLength () {
    }
```

```
    public void readInput(String filename) {
        try {
```

```
            BufferedReader currentInput =
                new BufferedReader(new FileReader(filename));
            PrintWriter currentOutput =
                new PrintWriter(new BufferedWriter(
                    new FileWriter(filename + ".sessionLength")));
```

```
            StringTokenizer nextHttpTokenizer = null;
            String nextLine = null;
            String nextClient = null;
            Date nextTime = null;
            String nextUrl = null;
```

```
            StringTokenizer currentHttpTokenizer = null;
            String currentLine = null;
            String currentClient = null;
            Date currentTime = null;
            String currentUrl = null;
            String tmpTime = null;
            String returnCode = null;
```

```
long sessionLength;
```

```
// filter out initial traces which does not have  
// a successful HTTP return code from server  
// or it is a CGI  
do {
```

```
    // read the input traces
```

```
    currentLine = currentInput.readLine();
```

```
    if (currentLine == null) {
```

```
        break;
```

```
    }
```

```
    // tokenize a line of log traces
```

```
    currentHttpTokenizer = new StringTokenizer(currentLine);
```

```
    currentClient = next(currentHttpTokenizer); // host ID
```

```
    next(currentHttpTokenizer); // -
```

```
    next(currentHttpTokenizer); // -
```

```
    tmpTime = next(currentHttpTokenizer); // time
```

```
    try {
```

```
        currentTime = sm_dateFormatter.parse(tmpTime);
```

```
    }
```

```
    catch (ParseException e) {
```

```
        System.out.println("Error" + currentLine);
```

```
        System.out.println("Error in parsing timestamp " + tmpTime);
```

```
    }
```

```
    next(currentHttpTokenizer); // timezone
```

```
    next(currentHttpTokenizer); // method e.g. GET
```

```
    currentUrl = next(currentHttpTokenizer); // url
```

```
    next(currentHttpTokenizer); // http version
```

```
    returnCode = next(currentHttpTokenizer); // return code
```

```
    while ((returnCode.equals("200")) &&
```

```
        ((returnCode.equals("304")) ||
```

```
        (currentUrl.startsWith("/cgi-bin/") ||
```

```
        (currentUrl.startsWith("/htbin/")));
```

```
        // begin to read the input traces
```

```
        nextLine = currentInput.readLine();
```

```
        while (nextLine != null) {
```

```
            // read in next request
```

```
            nextHttpTokenizer = new StringTokenizer(nextLine);
```

```
            nextClient = next(nextHttpTokenizer); // host ID
```

```
            next(nextHttpTokenizer); // -
```

```
            next(nextHttpTokenizer); // -
```

```
            tmpTime = next(nextHttpTokenizer); // time
```



```

try {

    // parse the time stamp
    nextTime = sm_dateFormatter.parse(tmpTime);
}
catch (ParseException e) {
    System.out.println("Error: " + currentLine);
}

next(nextHttpTokenizer); // timezone
next(nextHttpTokenizer); // method e.g. GET
nextUrl = next(nextHttpTokenizer); // url
next(nextHttpTokenizer); // http version
returnCode = next(nextHttpTokenizer); // return code

if (((returnCode.equals("200")) ||
    (returnCode.equals("304"))) &&
    (!nextUrl.startsWith("/cgi-bin/")) &&
    (!nextUrl.startsWith("/htbin/"))) {
    // traces that has a successful HTTP return code
    // from server or it is a CGI

    if (nextClient.equals(currentClient)) {

        // same session
        sessionLength = nextTime.getTime() - currentTime.getTime();
        m_totalSessionLength += sessionLength;
        currentOutput.println(sessionLength);
        m_sessionCount++;

    }

    // advance index
    currentUrl = nextUrl;
    currentClient = nextClient;
    currentTime = nextTime;

    // read the next line
    nextLine = currentInput.readLine();

    // print it
    currentOutput.print(nextLine);
}
}

```

```
currentOutput.close();
```

```
    }  
    catch (IOException e) {  
        System.out.println("IOException in readInput");  
    }  
}
```

```
// get the next token  
public String next(StringTokenizer stringTokenizer) {  
    if (stringTokenizer.hasMoreTokens()) {  
        String s = stringTokenizer.nextToken();  
        return s;  
    } else {  
        return "";  
    }  
}
```

```
public static void main(String argv[]) {  
    SessionLength length = new SessionLength();  
    length.readInput("/opt2/xaa.sorted");  
    length.readInput("/opt2/xab.sorted");  
    length.readInput("/opt2/xac.sorted");  
    length.readInput("/opt2/xad.sorted");  
    length.readInput("/opt2/xae.sorted");  
    length.readInput("/opt2/xaf.sorted");  
    length.readInput("/opt2/xag.sorted");  
    length.readInput("/opt2/xah.sorted");  
    length.readInput("/opt2/xai.sorted");  
    length.readInput("/opt2/xaj.sorted");  
    length.readInput("/opt2/xak.sorted");  
    System.out.println("total " + length.m_totalSessionLength);  
    System.out.println("number sessions " + length.m_sessionCount);  
    System.out.println("mean " + length.m_totalSessionLength / length.m_sessionCount);  
}
```

Appendix B-2: StandardDeviation.java

```
// StandardDeviation.java
```

```
import java.util.*;
import java.io.*;
import java.text.*;

public class StandardDeviation {
    public long m_totalDifference = 0;
    public long m_numTotalDifference = 0;
    public long m_sessionCount = 0;

    public StandardDeviation () {
    }

    public void readInput(String filename) {
        String currentLine = null;
        long sessionLength = 0;
        long difference = 0;
        long mean = 1583039; // calculated mean from the SessionLength.java
        try {
            BufferedReader currentInput =
                new BufferedReader(new FileReader(filename + ".sessionLength"));

            currentLine = currentInput.readLine();

            // calculate the standard deviation
            while (currentLine != null) {
                sessionLength = Long.parseLong(currentLine);
                difference = Math.abs(sessionLength - mean);
                difference = difference * difference;
                if (Long.MAX_VALUE - m_totalDifference <= difference) {
                    m_numTotalDifference++;
                    m_totalDifference = m_totalDifference - Long.MAX_VALUE + difference;
                } else {
                    m_totalDifference += difference;
                }

                m_sessionCount++;
                currentLine = currentInput.readLine();
            }

        } catch (IOException e) {
            System.out.println("IOException: " + e.getMessage());
        }
    }
}
```

```

        e.printStackTrace();
    }
}

```

```

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

```

```

public static void main(String argv[]) {
    StandardDeviation standardDeviation = new StandardDeviation();
    standardDeviation.readInput("/opt2/xaa.sorted");
    standardDeviation.readInput("/opt2/xab.sorted");
    standardDeviation.readInput("/opt2/xac.sorted");
    standardDeviation.readInput("/opt2/xad.sorted");
    standardDeviation.readInput("/opt2/xae.sorted");
    standardDeviation.readInput("/opt2/xaf.sorted");
    standardDeviation.readInput("/opt2/xag.sorted");
    standardDeviation.readInput("/opt2/xah.sorted");
    standardDeviation.readInput("/opt2/xai.sorted");
    standardDeviation.readInput("/opt2/xaj.sorted");
    standardDeviation.readInput("/opt2/xak.sorted");
    System.out.println("totalDifference " + standardDeviation.m_totalDifference);
    System.out.println("totalDifference " + standardDeviation.m_numTotalDifference);
    System.out.println("number sessions " + standardDeviation.m_sessionCount);
    long variance = (standardDeviation.m_totalDifference /
        standardDeviation.m_sessionCount +
        Long.MAX_VALUE / standardDeviation.m_sessionCount
        + standardDeviation.m_numTotalDifference);
    System.out.println("variance " + variance);
    System.out.println("standard deviation " +
        Math.sqrt((double) variance));
}

```

Appendix C: Simulation

The following is a list of programs that carried out the simulation:

1. Node.java – A Structure that represents a tree node used in the simulation.
2. NodeContent.java – Stores the information on a tree node.
3. BuildTree.java – Read the NASA log traces and build a tree representation of it.
4. ReadTree.java – Read the Tree and print it out, used for debugging.
5. NoPrefetch.java – Simulation for no prefetching.
6. BFS.java – BFS simulation.
7. UserPrefetchMultipleDoc.java – Simulation for prefetching multiple documents.
8. UserPrefetchOneDoc.java – Simulation for prefetching one document.

Appendix C-1: Node.java

```
// Node.java

import java.util.*;
import java.io.*;

public class Node implements Serializable {

    // content of this node
    public NodeContent m_nodeContent;

    // children of this node
    public ArrayList m_children = new ArrayList(100);

    public Node(String url, int size) {
        m_nodeContent = new NodeContent(url, 0, size);
    }

    // print function
    public String toString() {
        String str = "";
        str = str + m_nodeContent;
        str = str + "[ ";
        NodeContent child = null;
        for(int i=0; i<m_children.size(); i++) {
            child = (NodeContent) m_children.get(i);
            str = str + child + " ";
        }
        str = str + "]" + "\n";
        return str;
    }
}
```

Appendix C-2: NodeContent.java

```
// NodeContent.java
```

```
import java.util.*;
import java.io.*;
```

```
public class NodeContent implements Serializable {
    public String m_url; // URL
    public int m_visitCount = 0; // how many times it is visited
    public int m_size = 0; // size of this document in bytes

    // constructor
    public NodeContent(String url, int visitCount, int size) {
        m_url = url;
        m_visitCount = visitCount;
        m_size = size;
    }

    // print function
    public String toString() {
        String str = "[" + m_url + " " + m_visitCount + " " + m_size + "]";
        return str;
    }
}
```

Appendix C-3: BuildTree.java

```
// BuildTree.java

import java.util.*;
import java.io.*;
import java.text.*;

public class BuildTree {
    // hash table to store the web tree node
    public Hashtable m_webTree = new Hashtable(5000);

    // whether the web tree is loaded or not
    public Hashtable m_loadedWebTree = null;
    public BufferedReader m_currentInput = null;
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormat sm_dateFormatter = (DateFormat) sm_dateFormat;

    public BuildTree() {
    }

    public void readInput(String filename) {
        try {
            m_currentInput = new BufferedReader(new FileReader(filename));
            String line = m_currentInput.readLine();
            StringTokenizer httpTokenizer = null;
            String currentClient = null;
            String previousClient = null;
            Date currentTime = null;
            Date previousTime = null;
            String tmpTime = null;
            String previousUrl = null;
            String currentUrl = null;
            Node nodeFound = null;
            NodeContent child = null;
            boolean childFound = false;
            String returnCode = null;
            String size = null;
            int pageSize = 0;
            long sessionLength = 0;
            int pageNumber = 0;
        }
    }
}
```



```

while (line != null) {
    httpTokenizer = new StringTokenizer(line);
    currentClient = next(httpTokenizer); // host ID
    next(httpTokenizer); // -
    next(httpTokenizer); // -
    tmpTime = next(httpTokenizer); // time
    try {
        currentTime = sm_dateFormatter.parse(tmpTime);
    }
    catch (ParseException e) {
        System.out.println("Error in parsing timestamp " + tmpTime);
    }
    next(httpTokenizer); // timezone
    next(httpTokenizer); // method e.g. GET
    currentUrl = next(httpTokenizer); // url
    next(httpTokenizer); // http version
    returnCode = next(httpTokenizer); // return code
    size = next(httpTokenizer); // size
    if ((!currentUrl.startsWith("/cgi-bin/") && (!currentUrl.startsWith("/htbin/"))
    && ((returnCode.equals("200")) || (returnCode.equals("304")))) {
        nodeFound = (Node) m_webTree.get(currentUrl);
        if (nodeFound == null) {
            pageSize = 0;
            if (!size.equals("-")) {
                try {
                    pageSize = Integer.parseInt(size);
                }
                catch (NumberFormatException e) {
                    pageSize = 0;
                }
            }
            nodeFound = new Node(currentUrl, pageSize);
            String hashString = currentUrl;
            m_webTree.put(hashString, nodeFound);
        }
        // increment current node count
        nodeFound.m_nodeContent.m_visitCount++;

        // increment count for parent
        if (previousUrl != null) &&
            (previousClient.equals(currentClient)) {
            sessionLength = currentTime.getTime() - previousTime.getTime();
            if (sessionLength <= 1800000) { // 30*60*1000 half an hour in millisecond
                nodeFound = (Node) m_webTree.get(previousUrl);
                if (nodeFound == null)

```

```

        System.out.println("Error: previous node not stored in hashTable");
    } else {
        childFound = false;
        for (int i=0; i<nodeFound.m_children.size(); i++) {
            child = (NodeContent) nodeFound.m_children.get(i);
            if (currentUrl.equals(child.m_url)) {
                child.m_visitCount++;
                childFound = true;
                break;
            }
        }
        if (!childFound) { // create Child
            pageSize = 0;
            if (!size.equals("-")) {
                try {
                    pageSize = Integer.parseInt(size);
                }
                catch (NumberFormatException e) {
                    pageSize = 0;
                }
            }
            child = new NodeContent(currentUrl, 1, pageSize);
            nodeFound.m_children.add(child);
        }
    }
}

previousUrl = currentUrl;
previousClient = currentClient;
previousTime = currentTime;

line = m_currentInput.readLine();

m_currentInput.close();

catch (IOException e) {
    System.out.println("Error in reading input file while building tree");
}

public void processInput(String filename) {
    m_currentInput = new BufferedReader(new FileReader(filename));
    String line = m_currentInput.readLine();
}

```

```

        while (line != null) {
        }
        line = m_currentInput.readLine();
    }
    catch (IOException e) {
        System.out.println("error in reading input file while processing");
    }
}

```

```

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

```

```

public String toString() {
    return m_webTree.toString();
}

```

```

public void dumpTreeObject(String filename) {
    try {
        ObjectOutputStream out =
            new ObjectOutputStream(
                new FileOutputStream(filename));
        out.writeObject(m_webTree);
        out.close();
    }
    catch (IOException e) {
        System.out.println("dumpTreeObject IO error");
        e.printStackTrace();
    }
}

```

```

public void clean() {
    m_webTree.clean();
}

```

```

public void loadTreeObject(String filename) {
    try {
        ObjectInputStream in =
            new ObjectInputStream(
                new FileInputStream(filename));
    }
}

```

```

        m_loadedWebTree = (Hashtable) in.readObject();
        in.close();
    }
    catch (IOException e) {
        System.out.println("loadTreeObject IO error");
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        System.out.println("loadTreeObject ClassNotFoundException");
    }
}

```

```

public void printTreeObject() {
    System.out.println(m_loadedWebTree);
}

public static void main(String argv[]) {
    BuildTree tree = new BuildTree();
    tree.readInput("/opt2/xaa.sorted");
    tree.readInput("/opt2/xab.sorted");
    tree.readInput("/opt2/xac.sorted");
    tree.readInput("/opt2/xad.sorted");
    tree.readInput("/opt2/xae.sorted");
    tree.readInput("/opt2/xaf.sorted");
    tree.readInput("/opt2/xag.sorted");
    tree.readInput("/opt2/xah.sorted");
    tree.readInput("/opt2/xai.sorted");
    tree.readInput("/opt2/xaj.sorted");
    tree.readInput("/opt2/xak.sorted");

    tree.dumpTreeObject("/opt2/tree.dat");
}

```

Appendix C-4: ReadTree.java

```
// ReadTree.java

import java.io.*;
import java.util.*;

public class ReadTree {
    Hashtable m_loadedWebTree = null;

    public void loadTreeObject(String filename) {
        try {
            ObjectInputStream in =
                new ObjectInputStream(
                    new FileInputStream(filename));
            m_loadedWebTree = (Hashtable) in.readObject();
            in.close();
        }
        catch (IOException e) {
            System.out.println("loadTreeObject IO error");
            e.printStackTrace();
        }
        catch (ClassNotFoundException e) {
            System.out.println("loadTreeObject ClassNotFoundException");
        }
    }

    public void printTreeObject() {
        System.out.println(m_loadedWebTree);
    }

    public static void main(String argv[]) {
        ReadTree tree = new ReadTree();
        tree.loadTreeObject("/opt2/tree.dat");
        tree.printTreeObject();
    }
}
```

Appendix C-5: NoPrefetch.java

```
// NoPrefetch.java
```

```
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class NoPrefetch {
    public BufferedReader m_currentInput = null;
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormat sm_dateFormatter =
        (DateFormat) sm_dateFormat;
    public ArrayList m_urlCache = new ArrayList(20);
    public long m_totalHitRate = 0;
    public long m_totalRequest = 0;
    public long m_totalBytes = 0;
    public long m_sessionCount = 0;
    public int m_numLongTotalBytes = 0;
    public NoPrefetch() {
    }

    public void readInput(String filename) {
        try {
            StringTokenizer httpTokenizer = null;
            String currentClient = null;
            String previousClient = null;
            Date currentTime = null;
            Date previousTime = null;
            String tmpTime = null;
            String currentUrl = null;
            String returnCode = null;
            String size = null;
            int pageSize = 0;
            long sessionLength = 0;

            boolean urlFound = false;
            String cachedUrl = null;

            m_currentInput = new BufferedReader(new FileReader(filename));
            String line = m_currentInput.readLine();
            while (line != null)
```

```

httpTokenizer = new StringTokenizer(line);
currentClient = next(httpTokenizer); // host ID
next(httpTokenizer); // -
next(httpTokenizer); // -
tmpTime = next(httpTokenizer); // time
try {
    currentTime = sm_dateFormatter.parse(tmpTime);
}
catch (ParseException e) {
    System.out.println("Error in parsing timestamp " + tmpTime);
}
next(httpTokenizer); // timezone
next(httpTokenizer); // method e.g. GET
currentUrl = next(httpTokenizer); // url
next(httpTokenizer); // http version
returnCode = next(httpTokenizer); // return code
size = next(httpTokenizer); // size

if ((!currentUrl.startsWith("/cgi-bin/")) &&
    (!currentUrl.startsWith("/htbin/")) &&
    ((returnCode.equals("200")) || (returnCode.equals("304")))) {
    if (previousClient != null) {
        sessionLength = currentTime.getTime() - previousTime.getTime();
        if ((!previousClient.equals(currentClient)) ||
            (sessionLength > 1800000)) { // 30*60*1000
            // calculate average for previous session

            m_urlCache.clear();
            m_sessionCount++;
        }
    }
    m_totalRequest++;

    urlFound = false;
    for (int i=0; i<m_urlCache.size(); i++) {
        cachedUrl = (String) m_urlCache.get(i);
        if (cachedUrl.equals(currentUrl)) {
            urlFound = true;
            break;
        }
    }

    if (!urlFound)
        pageSize = 0;
    if ("size equals" != null)

```

```

        try {
            pageSize = Integer.parseInt(size);
        }
        catch (NumberFormatException e) {
            System.out.println("cannot parse size " + size);
            pageSize = 0;
        }
    }
    if (m_totalBytes + pageSize >= Long.MAX_VALUE) {
        m_numLongTotalBytes++;
        m_totalBytes = m_totalBytes - Long.MAX_VALUE + pageSize;
    } else {
        m_totalBytes += pageSize;
    }
    String cacheUrl = currentUrl;
    m_urlCache.add(cacheUrl);
} else {
    m_totalHitRate++;
}

previousClient = currentClient;
previousTime = currentTime;

}
line = m_currentInput.readLine();
} // while
m_currentInput.close();
}
catch (IOException e) {
    System.out.println("Error in reading input file while building tree");
}
}

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

// static void main(String argv[]) {
//     NoFetch tree = new NoFetch();

```



```

tree.readInput("/opt2/xaa.sorted");
tree.readInput("/opt2/xab.sorted");
tree.readInput("/opt2/xac.sorted");
tree.readInput("/opt2/xad.sorted");
tree.readInput("/opt2/xae.sorted");
tree.readInput("/opt2/xaf.sorted");
tree.readInput("/opt2/xag.sorted");
tree.readInput("/opt2/xah.sorted");
tree.readInput("/opt2/xai.sorted");
tree.readInput("/opt2/xaj.sorted");
tree.readInput("/opt2/xak.sorted");
System.out.println( "totalHitRate " + tree.m_totalHitRate);
System.out.println( "totalRequest " + tree.m_totalRequest);
System.out.println( "numLongTotalBytes " + tree.m_numLongTotalBytes);
System.out.println( "totalBytes " + tree.m_totalBytes);
System.out.println( "sessionCount " + tree.m_sessionCount);
}
}

```

Appendix C-6: BFS.java

```
// BFS.java
```

```
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class BFS {
```

```
    // hash table storing the web tree nodes
    public Hashtable m_webTree = new Hashtable(5000);
    public BufferedReader m_currentInput = null;
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormatter sm_dateFormatter = (DateFormatter) sm_dateFormat;
    public ArrayList m_urlCache = new ArrayList(20); // a cache of the URL documents
    public long m_totalHitRate = 0; // hit rate
    public long m_totalRequest = 0; // number of request
    public long m_totalBytes = 0; // number of transferred bytes
    public long m_totalPrefetchedBytes = 0; // number of prefetched bytes
    public long m_sessionCount = 0; // number of sessions
    public int m_numLongTotalBytes = 0; // in case number is too large
    public int m_numLongTotalPrefetchedBytes = 0; // in case number is too large
    // public double m_modemSpeed = 202375;
    // ISDN PRI basic 1544 kbps = 193 kbytes/second
    // public double m_modemSpeed = 16384; // ISDN basic 128 kbps = 16 kbytes/second
    // public double m_modemSpeed = 3686; // 28.8 kbps = 3.6 kbytes/second
    public double m_modemSpeed = 1843; // 14.4 kbps = 1.8 kbytes/second
    public int m_prefetchedNodeCount = 0; // number of prefetched doc
```

```
    public BFS() {
        ;
    }
```

```
    public void init(int speed)
    {
        m_modemSpeed = speed;
        m_totalHitRate = 0;
        m_totalRequest = 0;
        m_totalBytes = 0;
        m_totalPrefetchedBytes = 0;
        m_sessionCount = 0;
    }
```

```

m_numLongTotalBytes = 0;
m_numLongTotalPrefetchedBytes = 0;
m_prefetchedNodeCount = 0;
}

```

// load the web tree which was stored as a hash table

```

public void loadTreeObject(String filename) {
    try {
        ObjectInputStream in =
            new ObjectInputStream(
                new FileInputStream(filename));
        m_webTree = (Hashtable) in.readObject();
        in.close();
    }
    catch (IOException e) {
        System.out.println("loadTreeObject IO error");
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        System.out.println("loadTreeObject ClassNotFoundException");
    }
}

```

// print the web tree

```

public void printTreeObject() {
    System.out.println(m_webTree);
}

```

process the traces

```

public void processInput(String filename) {
    StringTokenizer currentHttpTokenizer = null;
    StringTokenizer nextHttpTokenizer = null;
    String currentClient = null;
    String nextClient = null;
    Date currentTime = null;
    Date nextTime = null;
    String tmpTime = null;
    String currentUrl = null;
    String nextUrl = null;
    String returnCode = null;
    String currentLane = null;
    String nextLane = null;
    Node currentNode = null;
}

```

```

Node nextNode = null;
long sessionLength = 0;
long prefetchLength = 0;
NodeContent prefetchNode = null;
NodeContent cachedUrl = null;
int i = 0;
boolean urlFound = false;

try {
    m_currentInput = new BufferedReader(new FileReader(filename));
    // read current stuff
    do {
        currentLine = m_currentInput.readLine();
        if (currentLine == null) {
            break;
        }

        // tokenize 1 line of log traces
        currentHttpTokenizer = new StringTokenizer(currentLine);
        currentClient = next(currentHttpTokenizer); // host ID
        next(currentHttpTokenizer); // -
        next(currentHttpTokenizer); // -
        tmpTime = next(currentHttpTokenizer); // time
        try {
            currentTime = sm_dateFormatter.parse(tmpTime);
        }
        catch (ParseException e) {
            System.out.println("Error in parsing timestamp " + tmpTime);
        }
        next(currentHttpTokenizer); // timezone
        next(currentHttpTokenizer); // method e.g. GET
        currentUrl = next(currentHttpTokenizer); // url
        next(currentHttpTokenizer); // http version
        returnCode = next(currentHttpTokenizer); // return code
    } while((!returnCode.equals("200")) &&
            (!returnCode.equals("304")) &&
            (currentUrl.startsWith("/cgi-bin/") &&
            (currentUrl.startsWith("/htbin/")));
    currentNode = (Node) m_webTree.get(currentUrl);
    m_totalBytes += currentNode.m_nodeContent.m_size;
    m_totalRequest++;
    nextLine = m_currentInput.readLine();
    while (nextLine != null) {
        // read in next request
        nextHttpTokenizer = new StringTokenizer(nextLine);
        nextClient = next(nextHttpTokenizer); // host ID

```

```

next(nextHttpTokenizer); // -
next(nextHttpTokenizer); // -
tmpTime = next(nextHttpTokenizer); // time
try {
    nextTime = sm_dateFormatter.parse(tmpTime);
}
catch (ParseException e) {
    System.out.println("Error in parsing timestamp " + tmpTime);
}
next(nextHttpTokenizer); // timezone
next(nextHttpTokenizer); // method e.g. GET
nextUrl = next(nextHttpTokenizer); // url
next(nextHttpTokenizer); // http version
returnCode = next(nextHttpTokenizer); // return code
nextNode = (Node) m_webTree.get(nextUrl);

// only process requests that were served successfully
// and filter out any CGI requests
if (((returnCode.equals("200")) ||
    (returnCode.equals("304"))) &&
    (!nextUrl.startsWith("/cgi-bin/")) &&
    (!nextUrl.startsWith("/htbin/"))) {

    // length between 2 requests
    sessionLength = nextTime.getTime() - currentTime.getTime();
    if ((!currentClient.equals(nextClient)) ||
        (sessionLength > 1800000)) { // 30*60*1000
        // if new session (either different user or exceeded
        // session length) clear cache and increment session count
        m_urlCache.clear();
        m_sessionCount++;
    } else {
        // prefetch document because the next request belong
        // to the currentClient as well

        // time allow me to prefetch the referred links
        // This is equals the session lenght minus the time to
        // prefetch the current node i.e. we only prefetch if
        // there are time after we have prefetch the current page
        prefetchLength = sessionLength -
            (long) (currentNode.m_nodeContent.m_size * m_modemSpeed / 1000);

        go through the list of children and prefetch while there
        is time
    }
}

```

```

for (i=0; i<currentNode.m_children.size(); i++) {

    prefetchNode = (NodeContent) currentNode.m_children.get(i);

    if (prefetchLength - prefetchNode.m_size / m_modemSpeed * 1000 < 0) {
        // no time to prefetch more links
        break;
    }
    if (!m_urlCache.contains(prefetchNode)) {

        m_urlCache.add(prefetchNode);
        prefetchLength -= (long) (prefetchNode.m_size / m_modemSpeed * 1000);
        m_prefetchedNodeCount++;
        if (Long.MAX_VALUE - m_totalPrefetchedBytes <= prefetchNode.m_size) {
            m_numLongTotalPrefetchedBytes++;
            m_totalPrefetchedBytes = m_totalPrefetchedBytes - Long.MAX_VALUE
                + prefetchNode.m_size;
        } else {
            m_totalPrefetchedBytes += prefetchNode.m_size;
        }

    }
}

// now that we have download the current page and done all
// the prefetching, we can download the NEXT request
m_totalRequest++;

// check if the document is in the cache
urlFound = false;
for (int j=0; j<m_urlCache.size(); j++) {
    cachedUrl = (NodeContent) m_urlCache.get(j);
    if (cachedUrl.m_url.equals(nextUrl)) {
        urlFound = true;
        break;
    }
}
if (!urlFound) {
    // transfer the file
    if (Long.MAX_VALUE - m_totalBytes <= nextNode.m_nodeContent.m_size) {
        m_numLongTotalBytes++;
        m_totalBytes = m_totalBytes - Long.MAX_VALUE
            + nextNode.m_nodeContent.m_size;
    } else {
        m_totalBytes += nextNode.m_nodeContent.m_size;
    }
}

```

```

    }
    Node cachedUrlNode = (Node) m_webTree.get(currentUrl);
    m_urlCache.add(cachedUrlNode.m_nodeContent);
} else {
    m_totalHitRate++;
}

currentClient = nextClient;
currentTime = nextTime;
currentUrl = nextUrl;
currentNode = nextNode;
}

// advance to the next request
nextLine = m_currentInput.readLine();

}
m_currentInput.close();
}
catch (IOException e) {
    System.out.println("error in reading input file while processing");
}
}

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

public String toString() {
    return m_webTree.toString();
}

public static void main(String argv[]) {
    BFS tree = new BFS();

    int speed = Integer.parseInt(argv[0]);
    System.out.println("Speed " + speed);
}

```

```

tree.loadTreeObject("/opt2/tree.dat");

tree.init(speed);
tree.processInput("/opt2/xaa.sorted");
tree.processInput("/opt2/xab.sorted");
tree.processInput("/opt2/xac.sorted");
tree.processInput("/opt2/xad.sorted");
tree.processInput("/opt2/xae.sorted");
tree.processInput("/opt2/xaf.sorted");
tree.processInput("/opt2/xag.sorted");
tree.processInput("/opt2/xah.sorted");
tree.processInput("/opt2/xai.sorted");
tree.processInput("/opt2/xaj.sorted");
tree.processInput("/opt2/xak.sorted");
System.out.println( "averagehitRate " + tree.m_totalHitRate);
System.out.println( "averageRequest " + tree.m_totalRequest);
System.out.println( "numLongTotalBytes " + tree.m_numLongTotalBytes);
System.out.println( "averageBytes " + tree.m_totalBytes);
System.out.println( "sessionCount " + tree.m_sessionCount);
System.out.println( "prefetchedNodeCount " + tree.m_prefetchedNodeCount);
System.out.println( "numLongTotalPrefetchedBytes "
    + tree.m_numLongTotalPrefetchedBytes);
System.out.println( "averagePrefetchedBytes " + tree.m_totalPrefetchedBytes);

}
}

```


Appendix C-7:

UserPrefetchMultipleDoc.java

// UserPrefetchMultipleDoc.java

```
import java.util.*;
import java.io.*;
import java.text.*;

public class UserPrefetch {
    // hash table storing the web tree nodes
    public Hashtable m_webTree = new Hashtable(5000);
    public BufferedReader m_currentInput = null;
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormat sm_dateFormatter = (DateFormat) sm_dateFormat;
    public ArrayList m_urlCache = new ArrayList(20); // a cache of the URL documents
    public long m_totalHitRate = 0; // hit rate
    public long m_totalRequest = 0; // number of request
    public long m_totalBytes = 0; // number of prefetched bytes
    public long m_totalPrefetchedBytes = 0; // number of sessions
    public long m_sessionCount = 0; // number of sessions
    public int m_numLongTotalBytes = 0; // in case number is too large
    public int m_numLongTotalPrefetchedBytes = 0; // in case number is too large
    public int m_prefetchedNodeCount = 0; // number of prefetched doc
    // public double m_modemSpeed = 202375;
    // ISDN PRI basic 1544 kbps = 193 kbytes/second
    // public double m_modemSpeed = 16384; // ISDN basic 128 kbps = 16 kbytes/second
    // public double m_modemSpeed = 3686; // 28.8 kbps = 3.6 kbytes/second
    public double m_modemSpeed = 1843; // 14.4 kbps = 1.8 kbytes/second
    public int m_prefetchedNodeCount = 0;
    public double m_userProbability = 0.8; // probability of users selected the correct doc
    // number of node that is selected correctly
    public long m_selectCorrectCount = 0;
    // number of node that is selected correctly and is prefetched
    public long m_selectCorrectPrefetchedCount = 0;

    public UserPrefetch() {
        // ...
    }
}
```

```

public void init(int speed, double userProbability) {
    m_modemSpeed = speed;
    m_userProbability = userProbability;
    m_totalHitRate = 0;
    m_totalRequest = 0;
    m_totalBytes = 0;
    m_totalPrefetchedBytes = 0;
    m_sessionCount = 0;
    m_numLongTotalBytes = 0;
    m_numLongTotalPrefetchedBytes = 0;
    m_prefetchedNodeCount = 0;
    m_selectCorrectCount = 0;
    m_selectCorrectPrefetchedCount = 0;
}

```

```

// load the web tree which was stored as a hash table
public void loadTreeObject(String filename) {
    try {
        ObjectInputStream in =
            new ObjectInputStream(
                new FileInputStream(filename));
        m_webTree = (Hashtable) in.readObject();
        in.close();
    }
    catch (IOException e) {
        System.out.println("loadTreeObject IO error");
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        System.out.println("loadTreeObject ClassNotFoundException");
    }
}

```

```

// process the traces
public void processInput(String filename) {
    StringTokenizer currentHttpTokenizer = null;
    StringTokenizer nextHttpTokenizer = null;
    String currentClient = null;
    String nextClient = null;
    Date currentTime = null;
    Date nextTime = null;
    String tmpTime = null;
    String currentUrl = null;
    String nextUrl = null;
    String tmpUrl = null;
    String tmpNode = null;
}

```

```

String currentLine = null;
String nextLine = null;
Node currentNode = null;
Node nextNode = null;
long sessionLength = 0;
long prefetchLength = 0;
NodeContent prefetchNode = null;
NodeContent cachedUrl = null;
int i = 0;
boolean urlFound = false;
boolean selectCorrect = false;
Node cachedUrlNode = null;
try {
    m_currentInput = new BufferedReader(new FileReader(filename));
    // read current stuff
    do {
        currentLine = m_currentInput.readLine();
        if (currentLine == null) {
            break;
        }
        currentHttpTokenizer = new StringTokenizer(currentLine);
        currentClient = next(currentHttpTokenizer); // host ID
        next(currentHttpTokenizer); // -
        next(currentHttpTokenizer); // -
        tmpTime = next(currentHttpTokenizer); // time
        try {
            currentTime = sm_dateFormatter.parse(tmpTime);
        }
        catch (ParseException e) {
            System.out.println("Error in parsing timestamp " + tmpTime);
        }
        next(currentHttpTokenizer); // timezone
        next(currentHttpTokenizer); // method e.g. GET
        currentUrl = next(currentHttpTokenizer); // url
        next(currentHttpTokenizer); // http version
        returnCode = next(currentHttpTokenizer); // return code
        while ((returnCode.equals("200")) &&
            (!returnCode.equals("304")) &&
            (currentUrl.startsWith("/cgi-bin/") &&
            !currentUrl.startsWith("/html/"))) {
            currentNode = (Node) m_webTree.get(currentUrl);
            m_totalBytes += currentNode.m_nodeContent.m_size;
            m_totalRequest++;
            nextLine = m_currentInput.readLine();
            while (nextLine != null) {

```

```

// read in next request
nextHttpTokenizer = new StringTokenizer(nextLine);
nextClient = next(nextHttpTokenizer); // host ID
next(nextHttpTokenizer); // -
next(nextHttpTokenizer); // -
tmpTime = next(nextHttpTokenizer); // time
try {
    nextTime = sm_dateFormatter.parse(tmpTime);
}
catch (ParseException e) {
    System.out.println("Error in parsing timestamp " + tmpTime);
}
next(nextHttpTokenizer); // timezone
next(nextHttpTokenizer); // method e.g. GET
nextUrl = next(nextHttpTokenizer); // url
next(nextHttpTokenizer); // http version
returnCode = next(nextHttpTokenizer); // return code
nextNode = (Node) m_webTree.get(nextUrl);

if (((returnCode.equals("200")) ||
    (returnCode.equals("304"))) &&
    (!nextUrl.startsWith("/cgi-bin/")) &&
    (!nextUrl.startsWith("/htbin/"))) {

    sessionLength = nextTime.getTime() - currentTime.getTime();
    if ((!currentClient.equals(nextClient)) ||
        (sessionLength > 1800000)) { // 30*60*1000
        m_urlCache.clear();
        m_sessionCount++;
    } else {
        // prefetch document because the next request belong
        // to the currentClient as well

        // time allow me to prefetch the referred links
        // This is equals the session lenght minus the time to
        // prefetch the current node i.e. we only prefetch if
        // there are time after we have prefetch the current page
        prefetchLength = sessionLength -
            (long) (currentNode.m_nodeContent.m_size / m_modemSpeed * 1000);

        // go through the list of children and prefetch while there
        // is time
        boolean previousCorrect = false;
        for (int nodePrefetched=0; nodePrefetched<4; nodePrefetched++)
            double r = Math.random();
    }
}

```

```

if (chance <= m_userProbability) {
    selectCorrect = true;
} else {
    selectCorrect = false;
}
if (selectCorrect && !previousCorrect) {
    previousCorrect = true;
    m_selectCorrectCount++;
    // try to prefetch the document which the user has selected
    // save check
    boolean noError = false;
    for (int k=0; k<currentNode.m_children.size(); k++) {
        prefetchNode = (NodeContent) currentNode.m_children.get(k);
        if (prefetchNode.m_url.equals(nextNode.m_nodeContent.m_url)) {
            noError = true;
            break;
        }
    }
    if (!noError) {
        System.out.println("invalid child currentNode " +
            currentNode + " nextNode " + nextNode);
    } else {
        if (!m_urlCache.contains(prefetchNode)) {
            // end save check
            if (prefetchLength - prefetchNode.m_size / m_modemSpeed * 1000 >= 0)
            {
                m_selectCorrectPrefetchedCount++;

                // have enough time to prefetch
                m_urlCache.add(prefetchNode);
                prefetchLength -=
                    (long) (prefetchNode.m_size / m_modemSpeed * 1000);
                m_prefetchedNodeCount++;
                if (Long.MAX_VALUE - m_totalPrefetchedBytes <=
                    prefetchNode.m_size) {
                    m_numLongTotalPrefetchedBytes++;
                    m_totalPrefetchedBytes = m_totalPrefetchedBytes -
                        Long.MAX_VALUE + prefetchNode.m_size;
                } else {
                    m_totalPrefetchedBytes += prefetchNode.m_size;
                }
            }
        }
    }
}

```

```

    } else {
        for (i=0; (i<currentNode.m_children.size()); i++) {
            prefetchNode = (NodeContent) currentNode.m_children.get(i);

            if (!((!selectCorrect) &&
                (prefetchNode.m_url.equals(nextNode.m_nodeContent.m_url)))) {
                if (prefetchLength-prefetchNode.m_size/m_modemSpeed * 1000 < 0)
                {
                    break;
                }
                if (!m_urlCache.contains(prefetchNode)) {

                    m_urlCache.add(prefetchNode);
                    prefetchLength -=
                        (long) (prefetchNode.m_size / m_modemSpeed * 1000);
                    m_prefetchedNodeCount++;

                    if (Long.MAX_VALUE - m_totalPrefetchedBytes <=
                        prefetchNode.m_size) {
                        m_numLongTotalPrefetchedBytes++;
                        m_totalPrefetchedBytes = m_totalPrefetchedBytes -
                            Long.MAX_VALUE + prefetchNode.m_size;
                    } else {
                        m_totalPrefetchedBytes += prefetchNode.m_size;
                    }
                    break;
                }
            }
        }
    }
}

```

```

// now that we have download the current page and done all
// the prefetching, we can download the NEXT request
m_totalRequest++;

```

```

urlFound = false;
for (int j=0; j<m_urlCache.size(); j++) {
    fetchedUrl = (NodeContent) m_urlCache.get(j);
    if (fetchedUrl.m_url.equals(nextUrl)) {
        urlFound = true;
        break;
    }
}

```

```

    }

    if (!urlFound) {
        // transfer the file
        if (Long.MAX_VALUE - m_totalBytes <= nextNode.m_nodeContent.m_size) {
            m_numLongTotalBytes++;
            m_totalBytes = m_totalBytes - Long.MAX_VALUE +
                nextNode.m_nodeContent.m_size;
        } else {
            m_totalBytes += nextNode.m_nodeContent.m_size;
        }
        cachedUrlNode = (Node) m_webTree.get(currentUrl);
        m_urlCache.add(cachedUrlNode.m_nodeContent);
    } else {
        m_totalHitRate++;
    }

    currentClient = nextClient;
    currentTime = nextTime;
    currentUrl = nextUrl;
    currentNode = nextNode;
}

nextLine = m_currentInput.readLine();

}
m_currentInput.close();
}
catch (IOException e) {
    System.out.println("error in reading input file while processing");
}

public String nextStringTokenizer(stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

public String toString() {
    return m_webTree.toString();
}

```

```
}
```

```
public static void main(String argv[]) {
    UserPrefetch tree = new UserPrefetch();
    int speed = Integer.parseInt(argv[0]);
    double userProbability = Double.parseDouble(argv[1]);

    System.out.println("Speed " + speed + " user probability " + userProbability);
    tree.loadTreeObject("/opt2/tree.dat");
    tree.init(speed, userProbability);
    tree.processInput("/opt2/xaa.sorted");
    tree.processInput("/opt2/xab.sorted");
    tree.processInput("/opt2/xac.sorted");
    tree.processInput("/opt2/xad.sorted");
    tree.processInput("/opt2/xae.sorted");
    tree.processInput("/opt2/xaf.sorted");
    tree.processInput("/opt2/xag.sorted");
    tree.processInput("/opt2/xah.sorted");
    tree.processInput("/opt2/xai.sorted");
    tree.processInput("/opt2/xaj.sorted");
    tree.processInput("/opt2/xak.sorted");

    System.out.println("averagehitRate " + tree.m_totalHitRate);
    System.out.println("averageRequest " + tree.m_totalRequest);
    System.out.println("numLongTotalBytes " + tree.m_numLongTotalBytes);
    System.out.println("averageBytes " + tree.m_totalBytes);
    System.out.println("sessionCount " + tree.m_sessionCount);
    System.out.println("prefetchedNodeCount " + tree.m_prefetchedNodeCount);
    System.out.println("numLongTotalPrefetchedBytes " +
        tree.m_numLongTotalPrefetchedBytes);
    System.out.println("averagePrefetchedBytes " + tree.m_totalPrefetchedBytes);
    System.out.println("selectCorrectCount " + tree.m_selectCorrectCount);
    System.out.println("selectCorrectPrefetchedCount "
        + tree.m_selectCorrectPrefetchedCount);
}
```


Appendix C-8:

UserPrefetchOneDoc.java

```
// UserPrefetchOneDoc.java
```

```
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class UserPrefetch {
    // hash table storing the web tree nodes
    public Hashtable m_webTree = new Hashtable(5000);
    public BufferedReader m_currentInput = null;
    public static final SimpleDateFormat sm_dateFormat =
        new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss");
    public static final DateFormat sm_dateFormatter =
        (DateFormat) sm_dateFormat;
    public ArrayList m_urlCache = new ArrayList(20);
    public long m_totalHitRate = 0; // hit rate
    public long m_totalRequest = 0; // number of request
    public long m_totalBytes = 0; // number of prefetched bytes
    public long m_totalPrefetchedBytes = 0; // number of sessions
    public long m_sessionCount = 0; // number of sessions
    public int m_numLongTotalBytes = 0; // in case number is too large
    public int m_numLongTotalPrefetchedBytes = 0; // in case number is too large
    public int m_prefetchedNodeCount = 0; // number of prefetched doc
    // public double m_modemSpeed = 202375;
    // ISDN PRI basic 1544 kbps = 193 kbytes/second
    // public double m_modemSpeed = 16384; // ISDN basic 128 kbps = 16 kbytes/second
    // public double m_modemSpeed = 3686; // 28.8 kbps = 3.6 kbytes/second
    public double m_modemSpeed = 1843;
    // 14.4 kbps = 1.8 kbytes/second public int m_prefetchedNodeCount = 0;
    public double m_userProbability = 0.8; // probability of user selected the correct doc
    // number of node that is selected correctly
    public long m_selectCorrectCount = 0;
    // number of node that is selected correctly and is prefetched
    public long m_selectCorrectPrefetchedCount = 0;

    public UserPrefetch() {
        // ...
    }
}
```

```

public void init(int speed, double userProbability) {
    m_modemSpeed = speed;
    m_userProbability = userProbability;
    m_totalHitRate = 0;
    m_totalRequest = 0;
    m_totalBytes = 0;
    m_totalPrefetchedBytes = 0;
    m_sessionCount = 0;
    m_numLongTotalBytes = 0;
    m_numLongTotalPrefetchedBytes = 0;
    m_prefetchedNodeCount = 0;
    m_selectCorrectCount = 0;
    m_selectCorrectPrefetchedCount = 0;
}

```

```

// load the web tree which was stored as a hash table
public void loadTreeObject(String filename) {
    try {
        ObjectInputStream in =
            new ObjectInputStream(
                new FileInputStream(filename));
        m_webTree = (Hashtable) in.readObject();
        in.close();
    }
    catch (IOException e) {
        System.out.println("loadTreeObject IO error");
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        System.out.println("loadTreeObject Class.NotFound error");
    }
}

```

// process the traces

```

public void processInput(String filename) {
    StringTokenizer currentHttpTokenizer = null;
    StringTokenizer nextHttpTokenizer = null;
    String currentClient = null;
    String nextClient = null;
    Date currentTime = null;
    Date nextTime = null;
    String tmpTime = null;
    String currentUri = null;
    String nextUri = null;
    String returnUrl = null;
}

```

```

String currentLine = null;
String nextLine = null;
Node currentNode = null;
Node nextNode = null;
long sessionLength = 0;
long prefetchLength = 0;
NodeContent prefetchNode = null;
NodeContent cachedUrl = null;
int i = 0;
boolean urlFound = false;
boolean selectCorrect = false;
Node cachedUrlNode = null;
try {
    m_currentInput = new BufferedReader(new FileReader(filename));
    // read current stuff
    do {
        currentLine = m_currentInput.readLine();
        if (currentLine == null) {
            break;
        }
        currentHttpTokenizer = new StringTokenizer(currentLine);
        currentClient = next(currentHttpTokenizer); // host ID
        next(currentHttpTokenizer); // -
        next(currentHttpTokenizer); // -
        tmpTime = next(currentHttpTokenizer); // time
        try {
            currentTime = sm_dateFormatter.parse(tmpTime);
        }
        catch (ParseException e) {
            System.out.println("Error in parsing timestamp " + tmpTime);
        }
        next(currentHttpTokenizer); // timezone
        next(currentHttpTokenizer); // method e.g. GET
        currentUrl = next(currentHttpTokenizer); // url
        next(currentHttpTokenizer); // http version
        returnCode = next(currentHttpTokenizer); // return code
        while ((returnCode.equals("200")) &&
            (returnCode.equals("304")) ||
            (currentUrl.startsWith("/cgi-bin/") ||
            (currentUrl.startsWith("/htbin/")))) {
            currentNode = (Node) m_webTree.get(currentUrl);
            m_totalBytes += currentNode.m_nodeContent.m_size;
            m_totalRequest++;
            nextLine = m_currentInput.readLine();
            while (nextLine != null) {

```

```

// read in next request
nextHttpTokenizer = new StringTokenizer(nextLine);
nextClient = next(nextHttpTokenizer); // host ID
next(nextHttpTokenizer); // -
next(nextHttpTokenizer); // -
tmpTime = next(nextHttpTokenizer); // time
try {
    nextTime = sm_dateFormatter.parse(tmpTime);
}
catch (ParseException e) {
    System.out.println("Error in parsing timestamp " + tmpTime);
}
next(nextHttpTokenizer); // timezone
next(nextHttpTokenizer); // method e.g. GET
nextUrl = next(nextHttpTokenizer); // url
next(nextHttpTokenizer); // http version
returnCode = next(nextHttpTokenizer); // return code
nextNode = (Node) m_webTree.get(nextUrl);
if (((returnCode.equals("200")) ||
    (returnCode.equals("304"))) &&
    (!nextUrl.startsWith("/cgi-bin/")) &&
    (!nextUrl.startsWith("/htbin/"))) {

    sessionLength = nextTime.getTime() - currentTime.getTime();
    if (!currentClient.equals(nextClient)) ||
        (sessionLength > 1800000)) { // 30*60*1000
        m_urlCache.clear();
        m_sessionCount++;
    } else {
        // prefetch document
        prefetchLength = sessionLength -
            (long) (currentNode.m_nodeContent.m_size / m_modemSpeed * 1000);
        double chance = Math.random();
        if (chance <= m_userProbability) {
            selectCorrect = true;
        } else {
            selectCorrect = false;
        }

        if (selectCorrect) {
            m_selectCorrectCount++;
            // try to prefetch the document which the user has selected
            // save check
            boolean noError = false;
            for (int k=0; k<currentNode.m_children.size(); k++) {
                prefetchNode = (Node) (currentNode.m_children.get(k));
            }
        }
    }
}

```

```

        if (prefetchNode.m_url.equals(nextNode.m_nodeContent.m_url)) {
            noError = true;
            break;
        }
    }
    if (!noError) {
        System.out.println("invalid child currentNode " +
            currentNode + " nextNode " + nextNode);
    } else {
        // end save check
        if (prefetchLength - prefetchNode.m_size /
            m_modemSpeed * 1000 >= 0) {
            m_selectCorrectPrefetchedCount++;

            // have enough time to prefetch
            m_urlCache.add(prefetchNode);
            prefetchLength -= (long) (prefetchNode.m_size /
                m_modemSpeed * 1000);
            m_prefetchedNodeCount++;
            if (Long.MAX_VALUE - m_totalPrefetchedBytes
                <= prefetchNode.m_size) {
                m_numLongTotalPrefetchedBytes++;
                m_totalPrefetchedBytes = m_totalPrefetchedBytes -
                    Long.MAX_VALUE + prefetchNode.m_size;
            } else {
                m_totalPrefetchedBytes += prefetchNode.m_size;
            }
        }

        //
    }

    } else {
        boolean nodePrefetched = false;
        for (i=0; i<currentNode.m_children.size(); i++) {
            prefetchNode = (NodeContent) currentNode.m_children.get(i);

            if (!selectCorrect) {
                if (prefetchNode.m_url.equals(nextNode.m_nodeContent.m_url)) {
                    if (prefetchLength - prefetchNode.m_size /
                        m_modemSpeed * 1000 < 0) {
                        break;
                    }

                    if (!m_urlCache.contains(prefetchNode)) {

                        m_urlCache.add(prefetchNode);
                        prefetchLength -=

```

```

        (long) (prefetchNode.m_size / m_modemSpeed * 1000);
        m_prefetchedNodeCount++;

        if (Long.MAX_VALUE - m_totalPrefetchedBytes <=
            prefetchNode.m_size) {
            m_numLongTotalPrefetchedBytes++;
            m_totalPrefetchedBytes = m_totalPrefetchedBytes -
                Long.MAX_VALUE + prefetchNode.m_size;
        } else {
            m_totalPrefetchedBytes += prefetchNode.m_size;
        }
        nodePrefetched = true;
    }
}

// now that we have download the current page and done all
// the prefetching, we can download the NEXT request
m_totalRequest++;

urlFound = false;
for (int j=0; j<m_urlCache.size(); j++) {
    cachedUrl = (NodeContent) m_urlCache.get(j);
    if (cachedUrl.m_url.equals(nextUrl)) {
        urlFound = true;
        break;
    }
}

if (!urlFound) {
    // transfer the file
    if (Long.MAX_VALUE - m_totalBytes <= nextNode.m_nodeContent.m_size) {
        m_numLongTotalBytes++;
        m_totalBytes = m_totalBytes - Long.MAX_VALUE +
            nextNode.m_nodeContent.m_size;
    } else {
        m_totalBytes += nextNode.m_nodeContent.m_size;
    }

    cachedUrlNode = (Node) m_webTree.getCurrentUrl();
    m_urlCache.add(cachedUrlNode.m_nodeContent);
} else {
    m_totalHttpRequest++;
}

```

```

    }

    currentClient = nextClient;
    currentTime = nextTime;
    currentUrl = nextUrl;
    currentNode = nextNode;
}

nextLine = m_currentInput.readLine();

}
m_currentInput.close();
}
catch (IOException e) {
    System.out.println("error in reading input file while processing");
}
}

public String next(StringTokenizer stringTokenizer) {
    if (stringTokenizer.hasMoreTokens()) {
        String s = stringTokenizer.nextToken();
        return s;
    } else {
        return "";
    }
}

public String toString() {
    return m_webTree.toString();
}

public static void main(String argv[]) {
    UserPrefetch tree = new UserPrefetch();
    int speed = Integer.parseInt(argv[0]);
    double userProbability = Double.parseDouble(argv[1]);

    System.out.println("Speed " + speed + " user probability " +
        userProbability);
    tree.loadTreeObjects("opt2/tree.dat");
    tree.init(speed, userProbability);
    tree.processInput("opt2/xaa.sorted");
    tree.processInput("opt2/xab.sorted");
    tree.processInput("opt2/xac.sorted");
    tree.processInput("opt2/xad.sorted");
    tree.processInput("opt2/xae.sorted");
}

```

```

tree.processInput("/opt2/xaf.sorted");
tree.processInput("/opt2/xag.sorted");
tree.processInput("/opt2/xah.sorted");
tree.processInput("/opt2/xai.sorted");
tree.processInput("/opt2/xaj.sorted");
tree.processInput("/opt2/xak.sorted");

```

```

System.out.println( "averagehitRate " + tree.m_totalHitRate);
System.out.println( "averageRequest " + tree.m_totalRequest);
System.out.println( "numLongTotalBytes " + tree.m_numLongTotalBytes);
System.out.println( "averageBytes " + tree.m_totalBytes);
System.out.println( "sessionCount " + tree.m_sessionCount);
System.out.println( "prefetchedNodeCount " + tree.m_prefetchedNodeCount);
System.out.println( "numLongTotalPrefetchedBytes " +
    tree.m_numLongTotalPrefetchedBytes);
System.out.println( "averagePrefetchedBytes " + tree.m_totalPrefetchedBytes);
System.out.println( "selectCorrectCount " + tree.m_selectCorrectCount);
System.out.println( "selectCorrectPrefetchedCount " +
    tree.m_selectCorrectPrefetchedCount);
}
}

```