Can Intelligent Prefetching Make Web Browsing Faster? *

A Synthesis Project Report
by
Jitendra D. Padhye
e-mail: {jitu@cs.umass.edu}

Abstract

This synthesis project proposes the use of techniques developed in the field of artificial intelligence to predict behavior of clients in networked environments with a client-server architecture. Successful prediction of user behavior allows the system to utilize its resources more efficiently. Predicting future access patterns of clients may allow the system to fetch some of the data from the server in advance. This not only reduces the response time as seen by the client, but it may also help to reduce the burst load on the network.

In our project we have chosen to concentrate on World Wide Web (WWW) based systems, as most of these systems are inherently stateless and asynchronous. This makes the problem of predicting user behavior more difficult, while at the same time it promises greater improvements in performance with successful predictions. As a starting point, we focus our attention on web-based intelligent tutoring systems. We believe that such systems will increasingly augment the traditional classroom model of teaching and learning as the Internet spreads wider and wider. The access times experienced by the users while downloading material from these systems will be a critical factor in their widespread acceptance. In addition, user modeling is an integral part of many intelligent tutoring systems. We hope to exploit these models to predict user access patterns.

To test our ideas, we have built a World Wide Web based intelligent tutoring system called MANIC (Multimedia Asynchronous Networked Individualized CourseWare). The implementation includes a student model server which creates and maintains the models of individual students using this system. The server uses these models to predict future access patterns of the student. This information is used by a proxy to prefetch documents from the web server in an effort to improve response times experienced by the users of the system. All the major components of the system have been implemented and a beta version is currently being used to teach a 1-credit course in the UMass Computer Science Department.

*This work is supported in part by the National Science Foundation under contracts NCR-9508274 and CDA-9502639 and by the Office of Research and Development contract 95-F137600-000. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

One of the most common complaints of Internet users who surf the "World Wide Web", is that at times, it can be "too slow". For most part, the delay in processing a user's browsing request can be attributed to the load on the network and the server(s). If the user's system had some way of anticipating future requests that would be made by the user, it could use that knowledge to prefetch the documents the user is most likely to request, and store them in a local cache (either main memory or local disk). If the user requests the prefetched documents, the delay in serving the user's request would be minimal. Prefetching is widely used to improve performance of file systems, database management systems and CPU instruction and data caches. Is it possible to anticipate future user requests in an asynchronous, stateless environment like the World Wide Web? We believe that this question can be tackled in two different, yet not necessarily independent, ways:

- Create a statistical access pattern model of a given website. This model attempts to capture the logical structure of a website. For example, consider an on-line news service, like the CNN website. The website initially presents the user with a menu of current headlines. Each headline links to a detailed story. At the end of several stories, links are provided into past news archives. Most of the users to such a website will first scan through the index of headlines presented on the "topmost" page in the web hierarchy. After that, most users are likely to follow the links to one of these headlines, to read more about that story. Few users would jump directly to specific items in past archives. Thus, in the context of such websites, it might be possible to make statements like: "80% of the users will choose one of the headline stories as the page they view next". This knowledge can be used for prefetching documents from that website. Note that the number of users to such "public" websites is quite large, which is what makes such statistical models viable. This approach has been recently explored in [Padm96].

- Certain websites have fewer visitors. However, these visitors may access the website heavily. One possible example of such a website is an internal website of a company, that stores data on current stock prices, inventory status and other operational parameters. The managers of the company use the website to access this data to carry out their duties. Each manager may be expected to be interested in certain parts of the website more than the others – e.g., a sales manager is more likely to access data on customer profiles than the manager of the accounting department. Thus, it might be possible to model *individual users*, and use those models for anticipating future requests. This approach has two essential requirements. The first is the ability to track and monitor individual users, and the second is the ability to create and maintain accurate user models. Both these requirements can be satisfied better if a small set of users access the data on a website repeatedly, and use the data to perform some meaningful activity.

The goal of this synthesis project is to test the later of the two prefetching strategies. We believe that an on-line tutoring system is a suitable environment to carry out such an investigation. This is due to several reasons. The number of students visiting an on-line tutoring system is relatively small (or is artificially controlled) compared to large public websites. The students have a specific goal: to learn the material presented on the website. The access patterns of students will differ based on their background and their speed of learning. Thus, modeling behavior of individual students to make prefetch decisions is a feasible alternative. Many intelligent tutoring systems create a model to guide tutoring activities. The same model may be used to make prefetching decisions. Moreover, with the advent of high speed networks and low priced, multimedia-enabled PCs, distance learning is fast becoming a viable adjunct to the standard classroom model of teaching and learning. Many of the future distance learning systems are likely to be based on a WWW architecture. When this happens, the performance of such systems (in terms of response time and resource utilization) will become an important issue. Model-based prefetching can help improve overall performance of web-based distance learning systems. MANIC, a web-based intelligent tutoring system, will help us investigate these issues further. This report describes the design and implementation aspects of MANIC project and reports some of the early results.

The report is organized as follows. Section 2 describes the general architecture of the MANIC environment and how it enables us to carry out research on intelligent prefetching. Section 3 briefly describes the mechanisms used for tracking user accesses in the MANIC system. These details are necessary to understand the architecture of two key components of the MANIC environment, viz. the prefetching, caching proxy and and the student model server that makes the actual prefetch decisions. These two components are described in sections 4 and 5, respectively. Section 6 presents related work, and Section 7 presents our conclusions and plans for future work.

2 MANIC Architecture: An Overview

Currently, MANIC delivers HTML (Hypertext Markup Language) documents and still GIF (Graphics Interchange Format) images synchronized with audio (encoded and delivered using a RealAudio server) over the Internet to students with multimedia capable machines with WWW browsers. This architecture provides a classroom like learning experience (unlike [Rebe96]), and also allows for asynchronous learning (unlike [Lieb96]). RealAudio codes audio for potential delivery over 28.8Kbps dialup lines and hence MANIC course materials can be accessed at home, at school, or at work. Our design also has provisions for a video server (MPEG) to be added as well, and we are currently adding this functionality. In this section we first overview the MANIC hardware and software components. We begin by describing the "raw" classroom materials.

2.1 Classroom Materials

MANIC makes classroom materials, including the HTML of the instructor's class notes (overheads), classroom audio and classroom video available to students over the WWW. The HTML notes are provided by the instructor. Initially, the notes we used were exact duplicates of the paper notes (specifically, a copy of the class overheads used by the instructor) used by in-class students. However, we soon realized that the on-line notes could (and should) be augmented with embedded hyperlinks to related material.

The course that we have already made available on-line via MANIC is a six-hour course on UNIX Network Programming (taught by J. Kurose in May 1996 in the studios of the Video Instructional Program (VIP) at the University of Massachusetts and distributed via the National Technological University). Since the course was taught in the VIP studios, we were provided with professional-quality recorded audio and video of the live classroom activities, which were then digitized.

The on-line MANIC-based version of this six-hour course is currently being used as half of a 1-credit Fall 1996 on-campus mini-course on Network Programming and Java. The course material (overheads and audio) is on-line at `http://www-aml.cs.umass.edu/~amldemo/`.

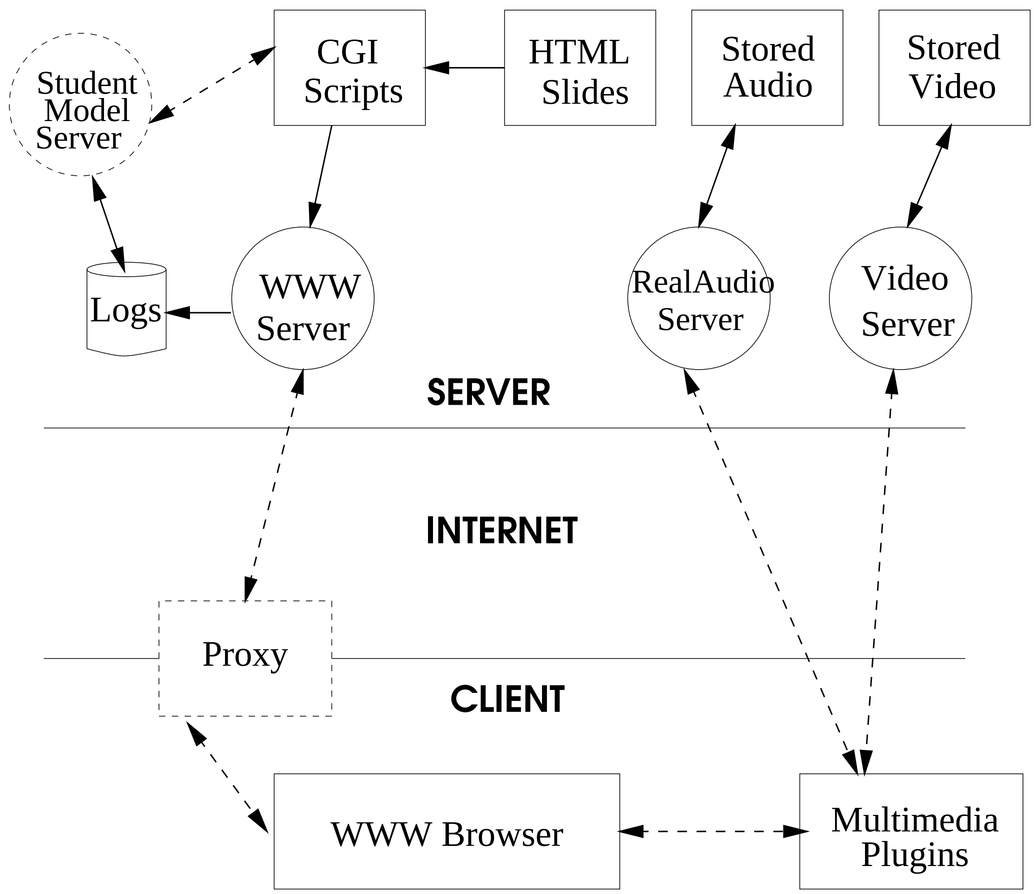

Figure 1: A high level overview of the MANIC architecture

2.2 Components of MANIC Environment

The MANIC architecture is shown in Figure 1. On the server side, MANIC is built around a standard WWW server, with customized common gateway interface (CGI) scripts that generate, "on the fly", the HTML document to be sent to the client. As we will see, the ability to generate/customize a document on the fly allows the server to dynamically and individually tailor a document for an individual student. Currently, our primary use of this tailoring is to highlight selected portions of a document in synchrony with the audio.

The client accesses the course material using a WWW browser (such as Netscape Navigator or Internet Explorer) and the RealAudio and other multimedia plugins. The WWW server sends HTML text to the client browser. An audio stream is also sent to the client plugin by our separate RealAudio server. Audio playout and HTML presentation are synchronized and controlled at the client computer using the synchronized multimedia feature provided by the RealAudio server/player. This interaction is described in more detail in [Kuro96].

The client side components of MANIC are all "off the shelf" pieces of software. Indeed, one of our design criteria in building MANIC was that we *not* develop specialized client-side software beyond some simple Javascript programs. Instead, we leveraged WWW browser and plug-ins (such as the RealAudio client, which is available from RealAudio as a free download) as much as possible. This not only minimized our own software development effort, but aided in portability as well –

companies that make browsers and plug-ins naturally want to make their products available on as many platforms as possible. Using off-the-shelf client software allows any student with a multimedia PC, WWW browser, and an Internet connection to use MANIC without having to download any MANIC-specific software. In retrospect, this decision to leverage existing client side software and focus our development efforts at the server side was a good one.

The two key components enabling prefetching of documents are the proxy and the student model server. Both these components are optional, and the system is a fully functional tutoring system without either or both of them. The student model server keeps track of activities of individual students and constructs a model of the student's behavior. This model can be used to customize the course material and tailor it to individual student's need (done on-the-fly) and make prefetching decisions. These decisions, (if the student model server is present) are communicated to the proxy via a special protocol that piggybacks on top of the normal HTTP (HyperText Transfer Protocol) communication. In absence of a student model server, the proxy automatically serves as a caching proxy [Bern96].

3 User Tracking in MANIC Environment

In this section we will briefly describe some of the implementation details of the MANIC architecture. We will restrict ourselves only to those aspects of implementation that are necessary to understand the next two sections, v.i.z. the proxy and the student model server. A more detailed coverage of the implementation details is provided in [Kuro96]. Moreover, we will assume that the reader is familiar with the HTTP 1.0 protocol. This protocol is described in more detain in [Bern96].

One of the primary requirements to implement user modeling and prefetching is that the system be able to track its users. The HTTP protocol is a stateless protocol [Bern96] and hence special mechanisms like cookies [Gund96] are necessary to track users. The MANIC architecture makes extensive use of single-session cookies. When a user registers with the MANIC web server, he/she is assigned a unique user id. Users are required to provide this user id via an HTML form every time they log on to the MANIC website. The server and the browser then negotiate a cookie containing this user id and other parameters. This cookie is valid until the user logs out or terminates his/her current browser program. The browser sends this cookie every time it sends a URL request to the MANIC web server. The URLs to most material on the MANIC website point to executable CGI scripts, instead of plain documents[1]. The data in the cookie field is made available to the CGI script by the web server via a special environmental variable. The CGI scripts use the user id to appropriately update the user access records, as they generate the required HTML output. The HTML output may be customized for the particular user. One example of such user specific action is to send prefetch advice to the proxy based on the model of that particular user.

4 The Proxy

HTTP proxies are not a new phenomenon, they have been around as long as web itself. Proxies for other protocols like FTP and gopher are also in use. An HTTP proxy accepts a browser's requests for fetching documents and fetches them on browser's behalf. There are two principle motivations behind using a proxy to obtain documents from a server instead of requesting them directly:

[1]This is true of all the material for which we wish to know the user access patterns.

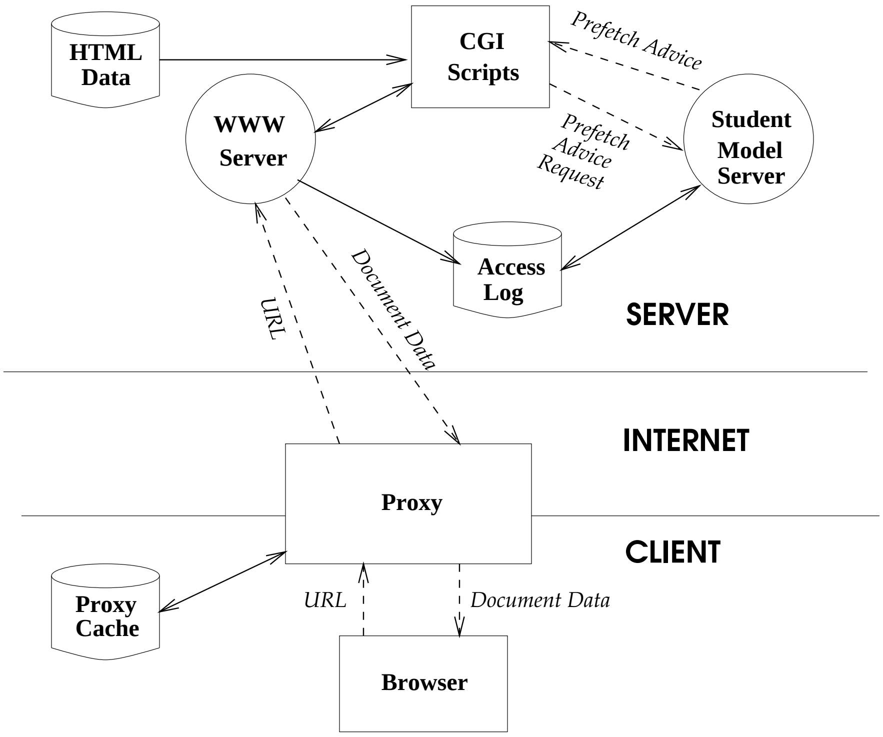

Figure 2: Prefetching under MANIC

1. **Anonymity:** Although the HTTP protocol does not transmit a user id (unless cookies are in use [Bern96]), it does reveal the Internet address of the machine on which the browser is running, as it is essential for communication using TCP/IP protocol. The user may not want to reveal the address of his/her workstation, for reasons of security and privacy. This can be achieved by using a proxy. Since the user communicates only with the proxy and as only the proxy communicates with the WWW server, the identity of a user's workstation is never revealed. From the viewpoint of the WWW server, all requests appear to originate from the proxy itself.

2. **Caching:** If a proxy locally caches some or all of the documents fetched by it, it is possible to reduce the latency experienced by the user(s) when the same document is requested again. This feature can be particularly useful if several users route their access through the same proxy. In addition to reducing the latency perceived by the users, this also reduces the traffic on the network. Sophisticated architectures such as the Harvest System [Bowm94] have been developed towards this end. HTTP protocol provides several features to facilitate caching.

The performance of a cache can be improved by judicious use of prefetching. One goal of the MANIC project, as mentioned earlier, is an attempt to test the feasibility of prefetching in a restricted, WWW-based environment. Hence, apart from satisfying the two objectives mentioned above, the proxy developed for the MANIC project is also able to prefetch documents from the MANIC web server. Unlike the prefetching agent described in [Padm96], the MANIC proxy, by itself, makes no prefetching decisions. In addition, (and unlike [Padm96]) we would like several users to share a common proxy.

There are many motivations behind making prefetching decisions at a centralized student model server and not at the individual proxies:

- The student models are constructed on the basis of student access logs, which are maintained by CGI scripts running at the MANIC web server. These logs can potentially get very large. Transporting them to the proxy is not a good solution.
- Constructing student models from the log can be a computationally expensive process. Since we expect some of the users to run their own personal proxies, it is not clear if sufficient computing power would always be available.
- A single proxy may be used by many users who may access many different intelligent tutoring systems. It is much easier to standardize a protocol to communicate only the prefetch decisions (in form of a list of URLs) instead of trying to standardize a protocol to communicate and evaluate either the access logs or the student models.

We now summarize the design requirements for a prefetching proxy for the MANIC environment. The design and implementation of the student model server is described in the next section.

1. First and foremost, the operation of the proxy must be completely transparent to the user. Apart from a possible change in the response time, the user must not see any difference between the operation of MANIC system with or without the presence of the proxy. This allows us to admit users who for some reason are not able to access the site through a proxy.
2. When the proxy is in use, it must handle connections to web servers other than the MANIC server in a seamless way, without the need for any user intervention. This allows us to include links to outside documents in material on the MANIC web server.
3. When the proxy serves a document to the user from its local cache, it must not create an inconsistency in the access logs maintained by the MANIC CGI scripts. For example, when several users share a proxy, it is possible that the needs of a certain user may be satisfied entirely out of the local cache of the proxy (as other users had accessed the documents just before this user requested them). If the web server is not notified of the user's actions, the access log for the user would be empty, and hence an incorrect model of user behavior would be constructed for this user. Thus, the server must be made aware of requests satisfied out of proxie's cache.
4. The prefetching requests made by the proxy must not be logged by the MANIC CGI scripts as actual accesses made by any user. It is important to note that requirements 3 and 4 are specific to the MANIC environment. The HTTP standard does not provide for any such consistency management.
5. The HTTP protocol requires that a proxy does not cache the outputs of CGI programs. These are recognizable as such by the lack of `Content-length` field in the header of the document received from a web server. However, in the MANIC environment, almost *all* documents are produced as outputs of CGI programs. Hence, the proxy must cache any documents received from the MANIC website unless explicitly notified otherwise by the server. However, when documents outside the MANIC environment are accessed, standard HTTP conventions must be followed to make caching decisions.

```
Browser Request:
GET http://www.cs.umass.edu/~jitu/ HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/3.0 (X11; I; OSF1 V3.0 alpha)
Host: www.cs.umass.edu
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

-------------------------------------------------------
Proxy Request to :www.cs.umass.edu, port 80:
GET /~jitu/ HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0 (X11; I; OSF1 V3.0 alpha)
Host: www.cs.umass.edu
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

-------------------------------------------------------
```

Figure 3: Trace of Browser-Proxy communication while requesting a document from a non-MANIC server.

To meet the above requirements we have designed protocol that piggybacks on top of the normal HTTP communication between the proxy and the server. Various messages belonging to our protocol are carried in the extra fields in the HTTP protocol headers. The protocol requires that the proxy be aware of the Internet name or address of the MANIC web server and all its aliases. This information is supplied to the proxy when the proxy is started up. We believe that this requirement is not unreasonable, as the users need to know the the name of the MANIC web server in order to be able to connect to it. In the remainder of this section, we describe the protocol and various modes of operation of the proxy.

When the proxy receives a request for a data item (specified by its URL) that is currently not within its cache, it forwards the request to the appropriate web server. If the request is for an item on a non-MANIC server, the proxy simply reformulates the request according to the HTTP standard and sends the request to the server. An example of such a request is shown in Figure 3. If the request is for an item on a MANIC server, the proxy reformulates the request according to the HTTP standards and adds to it an extra header field called `Pragma`, and sets its contents to `log_aadvice=1`. The contents of this field are available to the CGI program (in this case `slides.cgi`) running on the MANIC web server. An example of such a request is shown in Figure 4. The CGI program interprets these contents to mean that it should log the request in the proper user log (the user id is jitu1, as can be seen in the `Cookie` field), and send the requested data back to the proxy. This ensures that the user access log of user `jitu1` is updated consistently. We term such requests as `LOG_AND_SEND` requests.

When proxy receives a document that is not in its cache, it checks to see if the document can be cached. For documents downloaded from non-MANIC web servers, it follows the standard HTTP conventions to determine whether the document may be cached. For documents requested from MANIC website, these rules are relaxed in one important respect: the output of CGI programs on MANIC website is cached. This is forbidden under HTTP standards. It is essential to make this exception, as almost all documents on the MANIC website are produced by CGI programs. Apart from this single exception, the proxy follows the HTTP standard in all other respects.

When the proxy receives a request for an item that is already present in its local cache, it immediately serves the document to the requesting client. If the document is from a non-MANIC

```
Browser Request:
GET http://www-aml.cs.umass.edu/~amldemo/courseware/cgi-bin/slides.cgi?slide=1&b
uttons=N&highlight=1 HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/3.0 (X11; I; OSF1 V3.0 alpha)
Host: www-aml.cs.umass.edu
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Cookie: courseware_user_mode=stop; courseware_highlighting=on; courseware_user_id=jitu1

-------------------------------------------------------
Proxy Request to :www-aml.cs.umass.edu, port 80:
GET /~amldemo/courseware/cgi-bin/slides.cgi?slide=1&buttons=N&highlight=1 HTTP/1
.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0 (X11; I; OSF1 V3.0 alpha)
Host: www-aml.cs.umass.edu
Pragma: log_advice=1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Cookie: courseware_user_mode=stop; courseware_highlighting=on; courseware_user_id=jitu1

-------------------------------------------------------
```

Figure 4: Trace of Browser-Proxy communication while requesting a document from a MANIC server.

web site, no further action is taken. However, if the document is from the MANIC web server, simply serving the document from the cache may make the access logs maintained by the MANIC CGI scripts inconsistent. Thus, it is essential that the server be notified of the user's action. One easy solution to this problem is to send the request to the server anyway and ignore the received reply. This, however, is wasteful of network bandwidth and sever resources required to generate the HTML text. We can overcome the problem by informing the server that the request is only for the purpose of updating access logs and no real data transmission is required. This is accomplished by setting the `Pragma` field to contain `log_advice=2`. The CGI programs interpret any such request to be a request for updating user logs only and do not generate any HTML output. We term such types of requests as `LOG_ONLY` requests.

The CGI scripts that generate the HTML output on MANIC web servers are capable of taking several user-specific actions. One of these is to advise the proxy of anticipated user intentions. The proxy can then use this information to prefetch the documents that the user is most likely to request next and remove from its cache the documents the user is not likely to request in near future. The information about a user's intention is obtained by contacting the student model server. Whenever the CGI script processes either a `LOG_AND_SEND` or a `LOG_ONLY` request by a certain user, it contacts the student model server. The student model server responds back with two lists. One is a list of URLs that the user is likely to request in near future and the other is a list of URLs that the user is not likely to request in near future. Based on this response, the CGI script adds one or more `Prefetch` and `Delete` fields in the HTTP header of the response it is about to send back to the client. Each `Prefetch` field contains a URL from the first list while each `Delete` field contains a URL from the second list. An example of such a response is shown in Figure 5. When the proxy receives such a response, it removes these entries from the header and passes on the response to the client if necessary (i.e., if the response was for a `LOG_AND_SEND` type request). It then removes from its cache any entries contained in the `Delete` fields[2]. Then, the proxy retrieves documents

[2]This may not be a particularly good idea. More about this shortly.

```
Response for the URL
:http://www-aml.cs.umass.edu/~amltest/courseware/cgi-bin/slides.cgi?slide=4&buttons=DPN&highlight=1:
HTTP/1.0 200 Document follows
Date: Thu, 07 Nov 1996 02:59:48 GMT
Server: NCSA/1.5.1
Content-type: text/html
Prefetch: http://www-aml.cs.umass.edu/~amltest/courseware/cgi-bin/slides.cgi?slide=5&buttons=DPN&highlight=1
Delete: http://www-aml.cs.umass.edu/~amltest/courseware/cgi-bin/slides.cgi?slide=1&buttons=N&highlight=1

.... < document contents start here> ....
-------------------------------------------------------
```

Figure 5: Use of `Prefetch` and `Delete` fields

pointed at by the URLs mentioned in the `Prefetch` fields and stores them in the local cache. If any of these "prefetches" are from the MANIC web server, it is essential that none of them get logged as requests by any individual user, as no user has actually requested these documents. This is accomplished by setting the `Pragma` field in the header of the corresponding HTTP requests to contain `log_advice=3`. Such requests are termed as `NO_LOG` requests. When the CGI scripts on the MANIC web server process a request of this type, they do not record the request in any user log.

Apart from the cache manipulations required by the advice sent in `Prefetch` and `Delete` fields, the proxy manages its cache using Least Frequently Used (LFU) replacement policy.

This completes our description of the operation of the proxy. Several points about its design and operation are worth noting:

- The protocol to communicate prefetching and delete decisions introduces very little overhead, as it piggybacks on normal HTTP communication between the proxy and the web server. The size of additional fields required is negligible compared to size of most documents in the MANIC environment.
- The use of `Pragma` field to keep user access logs consistent also introduces very little overhead. In the case of `LOG_AND_SEND` and `NO_LOG` requests, the communication piggybacks on normal HTML communication between the web server and the proxy. Only `LOG_ONLY` requests require extra communication. However, even in that case, only a minimal amount of data (i.e., only the HTTP headers) are sent across the network.
- The proxy works seamlessly while accessing non-MANIC websites. For accesses to such websites, the proxy behavior conforms to HTTP standard.
- No change is required either in the web server or browser functionality.
- The proxy, in itself, has no "intelligence". All the model-based prefetching decisions are made by the student model server. We describe its design and operation in the next section.
- The proxy is completely stateless. This is in accord with the HTTP protocol requirements. In the terminology of OSI networking model, the proxy can be thought of as an entity at the session layer, while the "state" of the user interaction with MANIC website is an application-level artifact.

5 The Student Model Server

The student model server is the only "intelligent" component of the MANIC architecture. The server creates models of individual students based on their access patterns and other data. These models can then be used to make prefetching decisions and customize course instruction. When discussing a student model server, we have to focus on three things: representation of the domain of course material, the construction of the student model and uses the model is put to. We omit many implementation details, which can be found in [Stern96].

The domain of the course material is represented in terms of individual topics. Each topic has certain "prerequisite" topics. Prerequisite topics must be covered before the current topic is perused. Associated with each topic can be one or more quizzes, which are used to test a student's knowledge about that topic. In current version of MANIC, quiz selection is "linear", i.e., the student is allowed to take the next quiz as soon as he/she manages to pass the current one. In future, the selection of quizzes will also depend upon the slides seen by the student so far. The course material is represented as a linear list of slides. Each topic is associated with a certain contiguous subset of the slides.

The main purpose of a student model is to record a student's understanding of each topic presented in the course. As soon as the student's understanding of a topic is considered satisfactory, the student is considered qualified to start the next topic. In traditional [Gree91] systems, the model updates are based on the question/answer paradigm. The student is asked a certain set of questions related to each topic and if the answers are satisfactory, then the student is considered to have understood the topic. We believe that in the MANIC environment, we can glean additional information about a student's behavior by detecting his/her browsing patterns. For example, if a student is repeatedly going back to slides describing a particular networking protocol, we may construe that he/she is having difficulties in understanding that protocol.

As noted earlier, the student model can be put to two main uses. It can be used to make prefetching decisions and it can also be used to customize course material. Prefetching decisions are made by detecting user access patterns. Currently, the server can detect two types of pattern. The first pattern occurs when the student is listening to the audio without any interruptions. If this pattern is detected, the student model server uses its knowledge of course layout and requests the proxy to prefetch the next slide in the stream while the student is reviewing the current one. It also advises the proxy to delete from its cache the document that appears five steps earlier. For example, when the student is viewing slide 18, the student model server will advise the proxy to prefetch slide 19, and delete from its cache slide 13. Another pattern that can be detected by the student model server is that of a student "browsing" the material without listening to the audio. Currently, the server can only detect "linear" browsing, and the actions taken are similar to when the first type of pattern is detected. The student model also tracks the number of quizzes the student has taken and his/her score in each of them. Every time a student takes a quiz, the student model server determines the number of questions the student has answered correctly, and decides whether the number is sufficient to consider that the student has passed the quiz. When the student next requests to take a quiz, the server selects the quiz that immediately follows the last quiz the student has "passed".

As we gather more data about student's behavior and browsing patterns, we will implement a more sophisticated model. Additionally, we will develop course material that is more non-linear (i.e., has more "paths" through it, instead of a single, linear one like the current course material). With these improvements, we will be able to take better advantage of the "intelligence" and flexibility offered by the concept of student modeling.

6 Related Work

Prefetching and caching are not novel ideas. A significant amount of work exists on detecting and exploiting file access patterns for improving performance of networked and non-networked applications. Prefetching is also used as a standard performance enhancement technique for instruction processing at CPU level. Many of these techniques are described in standard textbooks such as [Tane92] and [Silb93]. Recently some of these ideas have been revisited in an effort to improve performance of mobile computing applications. The work of [Tait91] and [Nobl95] is significant in this regard.

Caching, especially "community" caching, has been recognized as viable way to reduce traffic on Internet backbone. The Harvest caching system described in [Bowm94] is an example of such a scheme. The Harvest system is an integrated set of tools for replicating and caching objects stored on various nodes across the Internet. It incorporates many protocols such as HTTP, NNTP and FTP. HTTP caching is also discussed in [Glas94] and [Luot94].

Prefetching as a technique to improve performance in WWW environment is a fairly new idea. [Padm96] describe an architecture in which access patterns of users on a large, public website (the homepage of Digital Inc.) was monitored to construct a model of site access patterns (i.e. statements like "80% of the users accessing page X access page Y immediately afterwards"). The model, coupled with "prefetching agents" attached to individual browser clients is used to prefetch documents in an effort to reduce response times. However, no effort is made to share the cache between multiple clients and hence the net result is an increase in traffic on the underlying network.

Modeling the behavior of users of a WWW site is of particular interest to companies involved in Internet commerce. A recent presentation by a Nilsen executive [Bhat96] discusses various aspects of this problem from a commercial viewpoint. Many ways of modeling student behavior have been explored by the researchers in intelligent tutoring system community. A good survey of AI based techniques for modeling student behavior may be found in [Beck96]. A comprehensive discussion on the use of student models to provide individualized instructions is presented in [Gree91]. In [Ship96] the authors describe a prefetching scheme based on student modeling that is very similar to ours. However, there is little mention of "community caching" and non-linear paths through course material.

7 Conclusions and Future Work

As mentioned earlier, all major components of the MANIC architecture have been successfully implemented. A beta version of the system is currently being used to teach a 1 credit course (CS591I) on socket programming and Java. Early reports from class instructors and students indicate that the system is being received very well. At the time of writing this report, little data is available on the effectiveness of prefetching and student modeling, as the system has been in operation only for a few weeks. Also, almost all the students enrolled in the course access the course material via the on-campus LAN. The availability of high bandwidth communication links reduces the need for either caching or prefetching. Instead, we plan to gather the traces of user activity on the course website. These traces, coupled with models of delays occurring on the Internet will be used to drive simulations of our prefetching algorithm. There is already evidence ([Padm96], [Pitk94]) available that prefetching and caching significantly reduce access times for WWW users. We believe that our prefetching scheme will be able to demonstrate comparable performance.

We learned several important lessons during the design and implementation of the MANIC system, which provide us with guidelines for future work. We now outline our plans for future work:

- **Cache replacement policy:** The current version of MANIC proxy uses LFU replacement policy for cache management, coupled with the `Delete` advice provided by the student model server. However, when a proxy is shared by multiple users (e.g., a community proxy), it might not always be advisable to follow the `Delete` advice given by the student model server. A student model server provides advice on the basis of a model of a *single* student. Even if a particular student is not likely to request a particular document in the proxy cache, it may be likely that some other user, connected to the same proxy, might request that document in the future. It might be the case that under the community proxy scenario, it is best to ignore the `Delete` advice from the student model server, and let LFU replacement policy handle all document replacements. A more detailed analysis of user access patterns is required to devise a good replacement policy.

- **When NOT to prefetch:** Just as the proxy may need to ignore the `Delete` advice provided by the student model server, there can be times when even the `Prefetch` advice may have to be ignored. One possible reason to ignore prefetching advice might be the scarcity of network bandwidth. If several users are connected to the same proxy, the machine running the proxy may not have sufficient incoming bandwidth. Under such circumstances, some of the prefetch advice may have to be ignored to prevent the original user requests from getting delayed. Alternatively, we may be able to come up with a mechanism where the proxy undertakes prefetching if and only if no user requests are pending. Currently we are investigating such an approach.

- **Incorporating video into the MANIC architecture:** Currently the MANIC system delivers audio synchronized with text display. The next logical step is to include video into this architecture. We believe that seeing the instructor and his/her gestures (even at low resolution and low frame rate) can improve a student's attention span. However, delivering video increases bandwidth requirements significantly, even at low resolution and low frame rates. In addition, instead of two media streams (text and audio), three media streams must be synchronized (audio, video and text) with each other. Human perception can tolerate very low jitter in audio-video synchronization (lip-synch). We are currently investigating the state-of-the-art regarding this issue.

- **Prefetching in everyday web browsing:** If prefetching is to be used in everyday web browsing, the prefetching capabilities would have to be included in individual browsers. Also, prefetching decisions would have to be based on "site models" instead of the "user model" approach described in this project. Some work towards this end has been reported in [Padm96]. Three important questions would have to be answered before prefetching can be used in web browsers as a general performance enhancement technique:

  1. How to compute effective models of site access patterns?
  2. How should these models be represented and communicated to the browsers? A standard protocol would have to be developed for this purpose.

3. How should the browser modify its prefetching and cache management strategy in the face of scares network bandwidth?

Analysis of long term access patterns of documents on MANIC web site may help provide answers to some of these questions.

- **Prefetching other data types:** This work, as well as the work described in [Padm96] and [Ship96], has focused on prefetching text and still images form web servers. However, prefetching is a much more general and powerful technique and can be used for other data types such streaming audio and video as well. Several buffering strategies have been proposed to reduce jitter in continuous media data streams and improving synchronization between multiple data streams. We plan to investigate model-based prefetching as an extention of these buffering strategies.

- **Using multi-media databases to improve data retrieval:** Currently all the audio and text data on MANIC web sites is stored as simple Unix files. This way of managing the data is clearly not scalable as the amount of material available on the site grows. Multimedia databases provide an efficient way to store, index, retrieve and manipulate audio/video data along with text and still images. For example, it may be possible to create a coherent, "personalized" presentation of certain topics by retrieving audio/video clips related to those topics and then merging them together with text and still images. Under the current architecture, the excessive file manipulation needed for such a "composition" makes the idea infeasible. We are currently looking into the state-of-the-art in the field of multimedia databases.

- **Improving the functionality of the student model server:** The student model server, being the single "intelligent" component of the MANIC architecture, provides maximum scope for future work. A vast amount of literature is available on intelligent tutoring systems. Some of the recent work has also focused on networked intelligent tutoring system. We will experiment with several different ways of modeling student behavior and compare their performance in terms of predicting future access patterns of students. However, the primary use of these models will be to customize the instructional material delivered to a student. Currently, the MANIC system provides a fairly linear path through all the available course material. However, in the future, the student can be guided through the course in an individualized manner based on his/her performance. For example, the server may store three versions of each "topic" covered in the course – a high paced one for students with prior exposure to similar material, a normal one for general students and a more detailed one for students having difficulties in grasping related topics or with lack of prior training in the area. The student model server would then select the version that best suites the needs of the student. Alternatively, the "composition" techniques described above can also be used to create individualized course material.

  The student model server can also be made aware of certain system parameters such as available outgoing bandwidth at the audio/video/WWW servers. This data can be blended with student models to modulate prefetching advice given to proxies of some "multimedia-hungry" students to improve bandwidth availability. For example, due to limited bandwidth available on UMass Internet connection, we will have to restrict video delivery to on-campus students.

- **An annotations server:** The ability to make annotations on the course material is an important learning tool. The annotations can be classified into three categories: personal annotations, annotations visible to a certain group and "global" annotations. Personal annotations would be made by individual students and the text of those would be visible only to the student making the annotations. Group annotations can be made by any person in a specific group and all members of that group will have access to them. Global annotations can be used by the course instructor to clarify difficult topics. Note that the annotations can use HTML syntax and include links to other slides in the course material or to outside sites (e.g., RFCs on various networking protocols). Currently we are designing an "Annotation Server" that will support such note-taking capabilities.

8 Acknowledgments

Project MANIC has benefited from the direction of Jim Kurose, Krithi Ramamrithm and Jack Stankovic. Jack and Krithi were kind enough to fund me while I worked on this stuff and probably didn't do much else ... The idea of intelligent prefetching was originally suggested by Mia Stern. Jesse Steinberg and Hu Imm Lee implemented a large number of the CGI scripts supporting the MANIC website. They were also responsible for the tedious task of encoding the lecture audio and the slides in a format suitable for delivery via the WWW. Gary Wallace is our resident Unix Buddha (complete with a beard!), who puts up with our endless stream of complaints. Zhi-Li Zhang and Sue Moon took the initiative to test our software in real-world setting. The CSCF personnel have provided invaluable technical support throughout this project. Special thanks are due to Subhabrata Sen, whose workstation serves as the MANIC web server and who patiently suffered through inconvenience of several development and testing cycles.

Although I have presented the entire MANIC system in this paper, I have concentrated my work on the prefetching aspects of the project. As a consequence, I have implemented the proxy and the caching mechanism. Mia Stern implemented the the port server and the student model server, including the policy for prefetching advice decisions and the quiz suggestion decisions.

References

[Beck96] J. Beck, M. Stern and E. Haugsjaa, "Applications of AI in Education", *ACM Crossroads*, pp. 11-15, Fall 1996.

[Bern96] T. BernersLee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", *Network Working Group, RFC 1945*, May 1996.

[Bhat96] Manish Bhatia, "Web Audience Measurement: Issues, Challenges and Solutions", *IPQC conference on Performance Measurement for Web Sites*, San Francisco, Sept. 30th. 1996. `http://www.nielsenmedia.com/sfpres`,

[Bowm94] C. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz and D. Wessels, "Harvest: A Scalable, Customizable Discovery and Access System", *Technical Report CU-CS-732-94*, University of Colorado-Boulder, 1994.

[Eklu95] J. Eklund, "Cognitive Models for Structuring Hypermedia and Implications for Learning from the World-Wide Web", *AUSWEB95- The First Australian WorldWideWeb Conference*, `http://www.scu.edu/ausweb95/papers/`, 1995.

[Glas94] S. Glassman, "A caching relay for the world wide web", *Proceedings of the First International World Wide Web Conference*, Amsterdam 1994. Elsevier.

[Gree91] J. Greer and G. McCalla, Editors, "Student Modeling: The key to Individualized Knowledge-Based Instruction", *NATO ASI Series, Series F, Volume 125*, Springer-Verlag, 1991.

[Gund96] S. Gundavaram, *CGI Programming on the World Wide Web*, O'Reilly & Associates, Inc., Sebatopol, CA, 1996.

[Hann95] R. Hannafin and H. Sullivan, "Learner Control in Full and Lean CAI Programs", *Educational Technology Research and Development*, volume 43, number 1, 1995, pp. 19-30.

[Kuro96] J.F. Kurose, H.I. Lee, J. Padhye, J. Steinberg and M. Stern, "MANIC: Multimedia Asynchronous Networked Individualized Courseware," Technical Report 96-75, Department of Computer Science, University of Massachusetts, Amherst MA 01003. Oct. 1996.

[Lieb96] J. Liebeherr et. al., "gwTTS: A grounds-wide Tele-Tutoring System", `http://www.cs.virginia.edu/ gwtts/final2.html`.

[Luot94] A. Luotonen and K. Altis, "World-wide web proxies", *Proceedings of the First International World Wide Web Conference*, Amsterdam 1994. Elsevier.

[Nobl95] B. Noble, M. Price and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing", *Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, MI., April 1995.

[Padm96] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency", *Proceedings of ACM SIGComm*, 1996, pp. 22-36.

[Pitk94] "A Simple yet robust caching algorithm based on dynamic access patterns", J. Pitkow and M. Recker, *http://www.vuw/ac.nz/˜mini/www/www-caching/caching.html*, 1994.

[Rebe96] A. Rebelsky, "Improving WWW-Aided Instruction: A Report from Experience", *Proceedings of ED-TELECOM 96*, June 17-22, 1996, pp. 276-281.

[Silb93] A. Silberschatz and P. Galvin, "Operating System Concepts", *Addison-Wesley*, 1993.

[Ship96] F. Shipman, C. Marshall and R. Furuta, "Creating Educational guided paths over the world-wide web", *Educational Telecommunications*, pp. 326-331, 1996.

[Stern96] Mia K. Stern, "Network Implementation of an Intelligent Tutoring System", *Synthesis Project Report*, Department of Computer Science, University of Massachusetts, Amherst, 1996.

[Stin95] A. Stinner, "Contextual Settings, Science Stories and Large Context Problems: Toward a More Humanistic Science Education", *Science Education*, volume 79, number 5, 1995, pp. 555-581.

[Tait91] C. Tait and D. Duchamp, "Detection and Exploitation of File Working Sets", *Proc. 11th Intl. Conf. on Distributed Computing Systems*, pp.2-9 Arlington TX, May 1991

[Tane92] A. Tannenbaum, "Modern Operating Systems", *Prentice Hall*, 1992.