Alleviating the Latency and Bandwidth Problems in WWW Browsing

Tong Sau Loon Vaduvur Bharghavan

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{s-tong,bharghav}@crhc.uiuc.edu

Abstract

This work addresses three problems that are associated with Web browsing: (a) low bandwidth available to the end user who is connected via slow modems or outdoor wireless networks, (b) long and variable latencies in document access, and (c) temporary disconnections of mobile users. Three techniques are used with a variety of heuristics in order to overcome these problems: (a) profiling user and group access patterns, and using these profiles in order to pre-fetch documents, (b) filtering HTTP requests and responses in order to reduce data transmission over bottleneck links, and (c) hoarding documents based on user profiles in order to support limited web browsing even during disconnection. In this paper, we describe the design and implementation of a WWW proxy based system that incorporates the above techniques. We describe our experiences with the proxy system, and present performance results that show a remarkable improvement in the experience of web browsing using this system.

1 Overview

In the last few years, the World Wide Web has had a remarkable effect on computing and communications in general, and Internet traffic in particular. Due to the explosion of the offered network load and the inherently best-effort paradigm of Internet service, WWW users typically notice long latencies and large variations in latency for web accesses. The scarcity of network bandwidth, particularly of the last link for users connected via low bandwidth modems and outdoor wireless networks exacerbates the latency problem, as does the transmission of increasingly graphics-oriented documents over slow networks. Mobile users face additional challenges in terms of frequent disconnections. In order to solve the above problems, we have built a WWW proxy based distributed system which is compatible with existing browsers and protocol standards. This paper presents the design and implementation of our system and shows the remarkable performance improvements we obtained using this system in conjunction with our standard browsing environment.

Three problems motivate this work: (a) low bandwidth available to the end user who is connected via slow modems or wireless networks, (b) high and variable latency due to congestion in the network, best-effort service in the Internet, and transmission of large amounts of data over slow links, and (c) temporary disconnections of mobile users - either involuntary due to fades, or voluntary to save cost and battery power. In order to find effective solutions to the above problems, our work is based on three key observations: (a) user accesses to WWW documents have been shown to follow certain patterns, albeit changing over time [8] - these changing patterns can be learned by monitoring user accesses and used for both pre-fetching and hoarding, (b) most large documents are graphics-intensive and graphics data can tolerate loss - thus filtering images can significantly reduce data transmission without compromising severely on quality, and (c) groups of users with similar interests tend to access similar documents - the commonality of their access patterns can be learned by monitoring their access patterns. It should be noted that none of the above techniques are unique to our work. Caching of documents based on recent history of accesses is provided with most browsers (e.g. Netscape, Explorer). Intelligent pre-fetching based on document hyperlinks and user access patterns have been proposed in several studies [1, 8]. Filtering in order to adapt to dynamic network quality of service has been proposed in related work, both in the context of WWW accesses and in the context of application adaptation in general [12, 25, 7]. Collaborative filtering has been proposed in the context of newsgroups [13] as well as WWW [11]. Hoarding has been proposed in the context of file systems support during disconnected operation [18, 22]. The contribution of this paper is the combination of several mechanisms and the use of multiple heuristics in order to intelligently pre-fetch documents

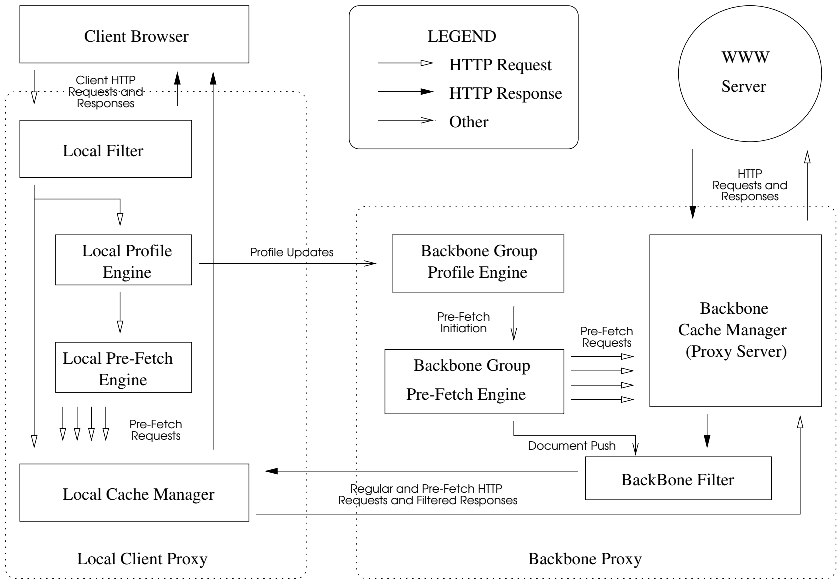

Figure 1: System Model

(based on user profiles, group profiles, and associated heuristics), filter documents (adaptively to varying QoS), and hoard documents anticipating disconnection (based on a hoard database that is learned over time). Performance results show that our system can learn and adapt to changing user behavior quickly, and significantly improve the experience of WWW browsing once the user profile is learned.

The rest of the paper is as follows. Section 2 describes the goals of our system. Section 3 describes the system design, while Section 4 discusses the various heuristics employed in order to improve efficiency. Section 5 provides implementation details, while Section 6 presents performance results. Section 7 compares related work to our approach, and Section 8 concludes the paper.

2 Architecture of the WWW Proxy System

The three key aspects of our system design are pre-fetching documents based on user and group profiles, filtering retrieved documents based on the available network quality of service, and hoarding documents in anticipation of network disconnections (for mobile users). For pre-fetching and hoarding to be effective, the cached copy of the documents must be as close to the client as possible. For filtering to be effective, it must be done as close to the server as possible; in particular, filtering needs to be performed *before* the bottleneck link on the retrieval path of the client.

In the ideal case, a server would have a set of filters associated with a document type. A client request would be accompanied with the measured network quality of service and the choice of filters. The server would then retrieve the document and pass it through the filter before sending it back to the client. The advantage of this design is that only the required data is sent over the network, thereby decreasing the latency of access. Besides, if the user has to pay for receiving data over the network (proportional to the amount of data received), this mechanism can reduce the cost of web access. The disadvantage of this design is that it requires QoS aware servers, and also places the burden of filtering on the server.

In cases where the user is connected to the network via a slow modem link or an outdoor wireless link, the last link typically happens to be the bottleneck link in the retrieval path. In this case, filtering the document before the last link may work just as well in reducing latency, if not cost. Since this does not require any change in the server, we use this model for our system.

The architecture of our WWW proxy system is shown in Figure 1. A WWW browser/client points to a local proxy server, through which all requests are routed. The local proxy server contains a profile management engine, a pre-fetching engine, a filtering engine and a cache management engine. The local proxy server points to a backbone proxy server, which also contains the same modules, but may service multiple users. The backbone proxy thus handles group profiles while the local proxy handles individual user profiles. In order to effectively manage the usage profile, all user accesses must traverse through the local proxy server; thus, the browser's cache is disabled. Profile-based pre-fetch is performed at both the local proxy server and the backbone proxy server (note, that hoarding must be done by the local proxy server). Filtering is done on both queries (e.g. to reduce HTTP headers) and responses (e.g. to clip images). The WWW server is not required to have any special functionality that is specific to our system.

In the following subsections, we first describe the flow of a HTTP request, and then describe the functionality of each module.

2.1 Request/Response Paths

When the user requests a document, the browser/client issues the request to the local proxy. The local proxy passes the request through the filter; the request may be immediately satisfied (e.g. if the request is for a site that is blocked out, such as advertisements), may be modified (e.g. header compression [16]), or may be passed through without modification. In the latter two cases, the request is logged by the profile manager and sent to the cache manager and the pre-fetcher. The pre-fetcher will determine which documents to pre-fetch and issue HTTP requests to the cache manager for these documents. If the cache has the document available, the request is satisfied, otherwise the request is forwarded to the backbone proxy server. After retrieval, the document is passed through the backbone filter and the result is sent to the local proxy server. Backbone proxy servers are also used to manage group profiles. Periodically, a user profile manager at the client sends updates the the profile managers of all the groups to which it belongs. This causes the backbone proxy server to update the group profile and issue pre-fetch requests, if required. As shown in Figure 1, retrieval of documents based on group profiles is accompanied by a push to the local cache of the users who belong to the group.

2.2 Filter Engine

The filter engines are responsible for modification of document requests and responses in order to reduce data traffic over the bottleneck link. Typical examples of filtering include reducing the color depth of images, clipping images, transmitting parts of audio files only, transmitting the first few words of each paragraph in a document, reducing the HTTP request header, suppressing requests to documents from a particular server, etc.

Invocation of the appropriate filter is accomplished by means of a rules database. A rule in the database specifies the invocation of a filter based on its document type, available network quality of service, size of the document, etc. The goal of the rules database is to allow adaptive filtering based on dynamic network conditions and data types. Currently, we have a simplistic mechanism for invoking rules, though the full-fledged implementation is ongoing.

2.3 Profile Engine and Usage Pattern Profile

A usage profile is a representation of a user or group's usage pattern of the web. The profile is learned over time by monitoring the access patterns of users. Using the profile to determine which documents to pre-fetch rather than simply using the document layout can significantly improve the efficiency of pre-fetching. The usage profile is a directed weighted graph, where the nodes represent URLs and edges represent the access path. The weight of a node *u* indicates the frequency of access of the corresponding URL, while the weight of an edge *(u,v)* indicates the frequency of access of the URL *v* immediately following the URL *u*. In order to reflect the changing access patterns of users and the temporal locality of accesses, recent history is given precedence in the weighting heuristic.

Let $n_t(u)$ represent the number of accesses of node u (i.e. the URL corresponding to node u) during the time interval t, $n_t(u,v)$ represent the number of accesses of node v immediately following node u, $w_t(u)$ represent the weight of node u after the time interval t, and $w_t(u,v)$ represent the weight of edge (u,v) after the time interval t. The weights of nodes and edges are computed as follows:

- $w_{t+1}(u) = w_t(u).\alpha_1 + n_t(u).\beta_1$
- $w_{t+1}(u,v) = w_t(u,v).\alpha_2 + n_t(u,v).\beta_2$

where α_1, α_2, β_1 and β_2 are constants which indicate the relative weights of recent history versus past history. These constants, along with the time window t play a key role in determining how effectively the profile adapts to the changing user access patterns. Based on our own experience, we have currently set $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$ and $t = 12$ hours. Note, that the time window t is closely related to the concept of *sessions*, which we describe later in this paper. While the current values of the constants are based on what worked for our usage profiles, we plan to do more experiments in order to determine the weights in the graph. While our weights are determined solely by frequency of access, we plan to use the following parameters for determining weights in the future: percentage of membership that uses an edge or node for group profiles, expected latency, livliness of documents, and size.

Group usage profiles are a natural extension of the idea of exploiting usage profile for predictive pre-fetch of documents. They also fit naturally into collaborative filtering of documents discussed in [13]. Group usage profiles can be inherited by users. Thus, a new user would first enroll in several groups. This will ensure that there are some form of informed speculative pre-fetch service for the user while her own usage pattern is being learned by the local profile engine. In addition, as the interests of the group changes over time, the user will be able to inherit the changes automatically.

The full paper will contain details of the profile computation including the pseudo-code.

2.4 Pre-fetch Engine

The pre-fetch engine will determine which documents to pre-fetch, based on the user profile and the current document request. At the local proxy server, when a HTTP request arrives, a list of documents likely to be used in the future is created. This list is then used to issue HTTP requests to the cache manager. The user profile is updated, and periodically updates are propagated to each group profile to which the user belongs. At the (backbone) proxy server of a group profile, an update of the group profile may trigger pre-fetches. Pre-fetched documents are then pushed to the local proxy servers of the members of the group. The group profile is particularly useful in two scenarios: (a) when a new user's profile is being learned, and (b) for receiving documents that are of interest to the whole group. In conjunction with push-based notifications, group profiles can serve to both pre-fetch and provide updates.

A special case of pre-fetch is *hoarding*. In the case of hoarding, only the node weights are used in order to pre-fetch the documents that the user is most likely to require in the future. As in the case of disconnected file systems [18], this gives rise to the problem of caching versus hoarding. An approach similar to hoard-walking is used in order to determine priority of documents in the cache. Details of this trade-off, as well as the pseudo-code for pre-fetching and hoard-walking are provided in the full paper.

2.5 Cache Manager

Each proxy has a cache manager which maintains a cache of documents available as a result of user requests, pre-fetches, and hoard-walks. When the proxy receives a request, if the document is not available at the cache, the request is forwarded to the next level of proxy. Note, that the browser/client cache is disabled since all user requests must go through to the local proxy at the client in order to built an accurate user profile. The cache manager also accepts document push from the next level of proxy, as a result of group profile pre-fetches. Ongoing work plans to provide a notification mechanism to the end user as a result of this push.

A key issue in cache management is the cache replacement policy. In the full paper, the trade-offs between different cache replacement policies - including least recently used node, least frequently used node, least weighted node, least probability of access node (minimum of the maximum product of edge weights from the current node to any other node), and minimum latency node - are discussed.

In the full paper, the pseudo-code for the cache manager is provided.

3 Performance Enhancing Heuristics

We used several heuristics in order to improve the efficiency of pre-fetching. While many of the heuristics listed below are simple and intuitively obvious, we found that a combination of these heuristics provided a remarkable performance improvement in our daily operation. This section lists the heuristics we used, while the next section describes the performance results.

1. *Web Sessions*: The notion of a web session had one of the greatest performance impacts. Without the notion of a session, the pre-fetch engine re-issued multiple requests for the same documents if a user accessed several pages from a "root" page. With the notion of a session, documents are only fetched once during the session. Subsequent requests are satisfied from the cache unless the user explicitly requests a fresh access using the RELOAD button. At the start of a session, documents with the highest node weights are pre-fetched. This accounts for a high percentage of the cache hits. Additionally, hoard-walking (for mobile computers) periodically refreshes

pages with high node weight (if they were deleted from the cache as a result of pre-fetches). Consequently, a large fraction of typical user accesses are satisfied locally during disconnection.

2. *Amortization of pre-fetches*: Pre-fetches are performed only when the network is "idle". In our system, only two ongoing pre-fetches are allowed at any time, and pre-fetches are not started until a minimum time after the issue of the user request. Additionally, pre-fetches have lower priority than regular requests. We observed that causing a delay in pre-fetches improves the latency of the requests document at the same time improving the latency of subsequent accesses by a factor of around 2.5.

3. *Weighting edges*: Weighting edges as opposed to only using node weights for pre-fetches ensures that the proper usage pattern is captured. When a user visits a URL, the probability of next access should be distributed over all the possible paths that she might take from the URL rather than over all the possible next destinations. For example, consider the following scenario: C was visited 2 times from A and 100 times from B. B was visited 10 times from A. When a user is at A, B should be pre-fetched even though it has a smaller weight than C. In cases when the cache size is small or near capacity, this heuristic becomes significant in terms of the hit ratio.

4. *Dynamically determining a document's dependents*: We distinguish a document from the images that are linked to it - which constitute its dependents. Dependents do not appear in the user profile, because they are accessed automatically upon access of the document, and also because they change frequently over time. The original implementation stored the URLs of the dependents. However, due to the frequent changes in HTML documents, requests were being made for non-existent documents. This heuristic removed all those redundant pre-fetch requests, at the same time keeping the user profile graph small.

5. *Continued download of document*: Even after the user specified "stop", we continue downloading a document in the local proxy server. This allows a user to click on another page while the previous page is begin downloaded in the background. It also serves as a crude form of user specified hoarding.

6. *CGI scripts*: CGI and other dynamic pages are pre-fetched and retained in the cache for a short period of time. Currently, CGI pages are not cached either in browsers or proxies; thus CGI latency is not hidden. However, with this heuristic, we can pre-fetch CGI pages and cache them for short periods of time in anticipation of an access. A CGI page is deleted upon the timeout, or after it is read once.

7. *Thresholds*: Since many documents are only accessed once from a parent document (e.g. a news item from a topic), we impose a minimum threshold edge frequency (in addition to the edge weight) in order to pre-fetch the node. Thus, we retain both edge weights and frequency of accesses along the edge in order to prevent unnecessary download of documents. Another threshold we impose is the size of documents. Large documents are not pre-fetched in our current implementation. However, we anticipate that with the inclusion of size as a weight parameter, this heuristic will be subsumed in the weighted edge heuristic.

4 Implementation

A detailed description of the implementation can be found in the full paper.

5 Performance Measurements

We have been using this sytem on a daily basis for a few months now, and our access patterns are well captured in our usage profiles (described in the full paper). While our access patterns change fairly rapidly, the profile manager is typically able to track these changes and adapt itself quickly. Due to space constraints, we only present the summary of performance results in this extended abstract. Detailed performance results are available in the full version of the paper.

Performance of the system was evaluated with a browser simulator called surf. Surf reads a text file containing a list of URLs to access (obtained from a trace of user accesses). It then issues HTTP requests for each of the URLs and fetches the dependents if necessary. Surf simulates user reading time by sleeping an amount of time proportional to the HTML file size. For every 500 bytes in a HTML file, surf sleeps for 1 second.

Table 1: Cache Manager Performance

Number of requests	320
Number of prefetches	298
Number of hits	281
Hit Ratio	87.8125%
Accuracy of prefetches	94.2953%

Table 2: Simulated Browsing Times

Test	Surf w/o cache	Surf with cache	Surf with system
Total surfing time	448.266 s.	431.625 s.	329.937 s.
Total network waiting time	135.254 s.	118.613 s.	16.94 s.
% Time spent waiting	30.1727%.	27.4806%.	5.13431%.
% Time spent reading	69.8273%.	72.5194%.	94.8657%.
Total bytes transferred	1217774 bytes.	1001565 bytes.	1049646 bytes.
Number of requests	320	320	320
Perceived bandwidth (Total bytes transferred/Total waiting time)	9003.61 bytes/sec.	8443.97 bytes/sec.	61962.6 bytes/sec.
Perceived delay per request (Total waiting time/Number of requests)	0.422669 secs/request.	0.370666 secs/request.	0.0529375 secs/request.

Results from a small number (approx 300) of requests, corresponding to 14 URLs, showed that typically, with a learned profile, a cache hit ratio of 87% could be obtained. Also, the accuracy of pre-fetches was about 90%. See table 1.

Results from the same test also showed that network waiting time was significantly reduced. In the test, the simulated web session (with a normal proxy system) took a total of 432 seconds with 119 seconds of network waiting time. With the introduction of profiling, pre-fetching and HTTP request filtering, the same web session took a total of 330 seconds with 17 seconds of network waiting time. Note that in both the tests, the amount of reading time remained constant at (432-119) = (330-17) = 313 seconds. See table 2.

In general, we have observed that the profiles are learned for the first time over a week. Learning changes in user patterns online is dependent on the time window of measurement as well as the constants in the weighting policy - in our case, once these parameters were tuned, the profiles were learned quickly as access patterns changed. The cache hit ratio for a learned profile was typically around 85%, and the percantage of pre-fetches that are actually used was typically around 90%. The reduction in waiting time ranges from a factor of 3 at night to a factor 7 during the day. Note, that all these numbers are specific to our access patterns and for our learned profiles. We expect that these numbers may change for other usage patterns - in particular, the tuning of the constants in determining user profiles played a key role in improving the efficiency of pre-fetching. However, while we believe that significant work needs to be done in order to automatically tune the system to match user access patterns, we do believe that our system can provide perceptible improvements in the experience of web browsing.

References

[1] Peakjet. http://www.peak-media.com/PeakJet/PeakJet.html.

[2] Squid internet object cache. http://squid.nlanr.net/Squid/.

[3] WWWW collector - the prefetching proxy server for www. http://shika.aist-nara.ac.jp/products/wcol/wcol.html.

[4] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Caching and replication in the world-wide web or a european perspective on www-connectivity. *3rd CaberNet Plenay Workshop*, April 1997.

[5] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0. RFC 1945, MIT/LCS and UC Irvine, May 1996.

[6] Azer Bestavros. Using speculation to reduce server load and service time on the www. *Proceedings of CIKM'95: The 4th ACM International Conference on Information and Knowledge Management*, Nov 1995.

[7] Vaduvur Bharghavan and Vijay Gupta. A Framework for Application Adaptation in Mobile Computing Environments. *Proceedings of IEEE Compsac'97*, November 1997.

[8] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of www client-based traces. *Technical Report TR-95-010, Boston University, CS Dept, Boston, MA 02215*, Apr 1995.

[9] Carlos R. Cunha and Carlos F. B. Jaccoud. Determining www user's next access and its application to pre-fetching. *Proceedings of ISCC'97: The Second IEEE Symposium on Computers and Communications*, Jul 1997. (Extended version).

[10] Adam Dingle and Tomas Partl. Web cache coherence. *Fifth International World Wide Web Conference*, May 1996.

[11] Firefly. *http://www.firefly.com*.

[12] A. Fox, S. Gribble, E. Brewer, and E. Amir. *Adapting to Network and Client Variation via On-Demand Dynamic Distillation*, **Proceedings of ASPLOS-VII**, Boston, Massachussetts, October 1996.

[13] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. *Using collaborative filtering to weave an information tapestry*, **Communications of the ACM**, Volume 35, Number 12, December 1992.

[14] James Gwertzman and Margo Seltzer. An analysis of geographical push-caching. http://www.eecs.harvard.edu/vino/web/server.cache/icdcs.ps.

[15] James Gwertzman and Margo Seltzer. The case for geographical push-caching. *Proceedings of the Fifth Annual Workshop on Hot Operating Systems*, May 1995.

[16] Barron C. Housel and David B. Lindquist. Webexpress: A system for optimizing web browsing in a wireless enviornment. *MobiCom'96 Demonstration Session*, Nov 1996. http://www.networking.ibm.com/art/artwewp.htm.

[17] A.-M. Kermarrec, M. van Steen, and A.S Tanenbaum. A flexible alternative to web caching. *3rd CaberNet Plenay Workshop*, April 1997.

[18] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems, Vol. 10, No. 1*, Feb 1992.

[19] Evangelos P. Markatos and Catherine E. Chronaki. A top-10 approach to prefetching on the web. Aug 1996.

[20] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving world wide web latency. *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Jul 1994.

[21] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, Jul 1996.

[22] C.D. Tait, H. Lei, S. Acharya, and H. Chang. *Intelligent File Hoarding for Mobile Computers*, **Proceedings of the ACM Conference on Mobile Computing and Networking**, Berkeley, California, November 1995.

[23] Joe Touch. Lowlat. http://www.isi.edu/lowlat/.

[24] Stuart Wachsberg, Thomas Kunz, and Johnny Wong. Fast world-wide web browsing over low-bandwidth links. http://ccnga.uwaterloo.ca/ sbwachsb/paper.html.

[25] B. Zenel and D. Duchamp. *Intelligent Communication Filtering for Limited Bandwidth Environments*, **Proceedings of the fifth Workshop on Hot Topics in Operating Systems**, Rosario, Washington, May 1995.