The Case for Geographical Push-Caching

James S. Gwertzman

Division of Applied Sciences
Harvard University
Cambridge, MA 02138

Margo Seltzer

Division of Applied Sciences
Harvard University
Cambridge, MA 02138

Abstract

Most wide-area caching schemes are client initiated. *Decisions on when and where to cache information are made without the benefit of the server's global knowledge of the situation. We believe that the server should play a role in making these caching decisions, and we propose* geographical push-caching *as a way of bringing the server back into the loop. The World Wide Web is an excellent example of a wide-area system that will benefit from geographical push-caching, and we present an architecture that allows a Web server to autonomously replicate HTML pages.*

1 Introduction

The World-Wide Web [1] operates for the most part as a cache-less distributed system. When two neighboring clients retrieve a document from the same server, the document is sent twice. This is inefficient, especially considering the ease with which Web browsers allow users to transfer large multimedia documents.

To combat this problem, some Web browsers have begun to add local client caches. These prevent one client from transferring the same document twice. Some networks are also beginning to add Web proxies [8, 9] that prevent two clients on the same campus network from transferring the same document twice.

The problem with both these schemes is that they are myopic. A client cache does not help a neighboring computer, and a campus proxy does not help a neighboring campus. Furthermore, these caches are usually limited in size. Disk might be cheap, but as the size of multimedia files increases it will be impossible to cache everything. As a result these caches will only be able to store the most popular items even though there is still some demand for other, less popular items.

The solution is for clients to share each other's cache space. The degree to which a file is replicated should be proportional to that file's global popularity, and clients should retrieve files from the the nearest cache to minimize network traffic. The server can satisfy both goals by deciding when and where to cache files. Furthermore, when the server decides where to cache a file, it can make this decision using its knowledge of network topology and the file's access history for even greater network bandwidth savings.

2 Motivation

Preliminary analysis of Web server access logs show that a few files on each server are responsible for most traffic from that server. Servers can therefore save a great deal of bandwidth and decrease their load by making access to those files more efficient. The slight load increase due to replicating and distributing files will be offset by this savings, and the client in turn will see increased performance.

Analysis also shows that the access pattern for each file is not uniform. File requests are often clustered geographically which implies that judicious selection of cache sites will provide excellent bandwidth and latency savings. Furthermore, we have also found strong correlations between geographical distance and Internet metrics like latency and network hops over short distances. This implies that geographical distance, which is easy to figure out using the `nets.unl.now` file at `nic.merit.edu` [10], can be used to predict network topology, which can be hard to figure out.

3 Architecture

There are two components to our proposed Web system: a modified HTTP server and a replication service. The modified server is responsible for tracking geographical access information for its files and for accepting and offering cached replicas of other files. This can be done by modifying a proxy server, such as the CERN proxy server [8], to accept files for replication using a modified POST request.

The replication service keeps track of modified HTTP servers that are willing to serve replicated files, the amount of available free space on each server, and each server's average load. The replication service

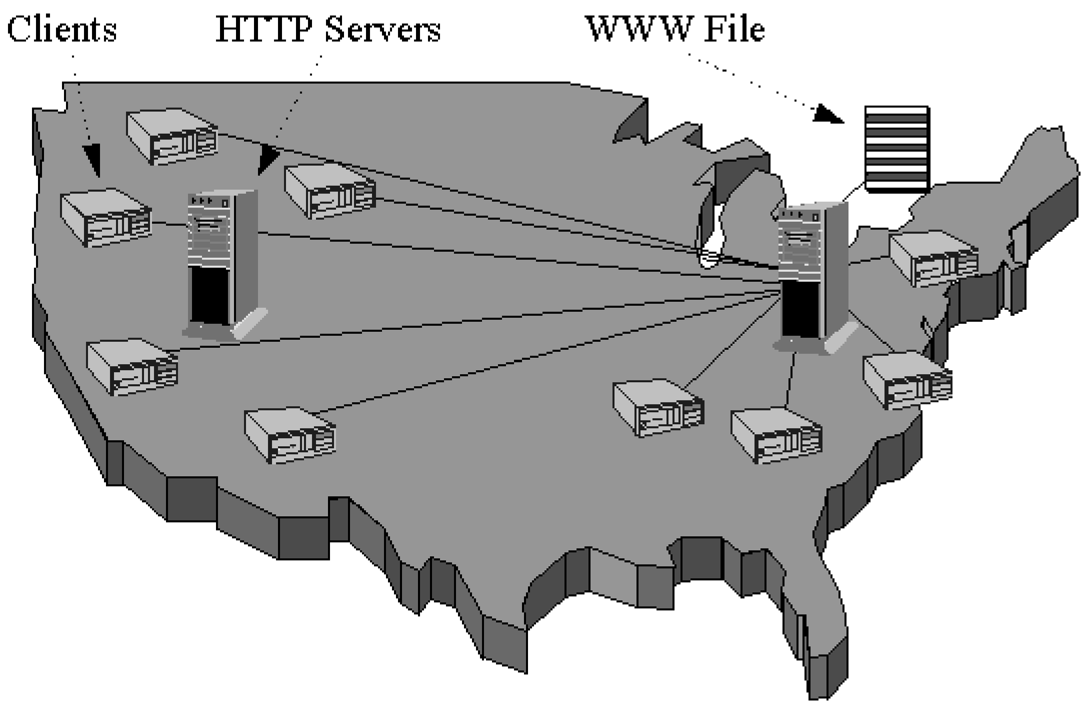

Figure 1: Before file replication takes place: several clients accessing a World Wide Web file on the east coast.

works with the modified server to decide where a given file should be cached.

We must minimize the amount of state that each Web server stores for its files, or else we will face scalability problems. We therefore track geographical access information in a coarse manner. We are currently using states and countries since these can easily be obtained from network addresses.

When the demand for a file exceeds a replication threshold, the server replicates it. We are using trace-driven simulation to determine reasonable values for the replication threshold, and we expect it to be dynamic.

The replication service decides where to replicate the file given its access history. The goal is to pick a server to cache the file that will minimize the amount of bandwidth and/or expected latency used in the future. We predict this by using the file's access history.

Figure 1 and figure 2 illustrate the replication process in action. Several clients from across the United States are accessing a file on an east coast server. The east coast server replicates the file such that network bandwidth is minimized, and the file ends up on a west coast server.

The replication service maintains a list of all HTTP servers willing to replicate files; it must choose one of them to cache the file. We are considering several algorithms to determine the optimal cache location. If the service knows the Internet's topology it can solve the problem by finding a good solution to the corresponding graph partitioning problem.

If only coarse grained information is available about the Internet, such as average latency between servers (available from `traceroute`), a better solution is to iterate over a representative sample of the available servers, calculating bandwidth savings for each. The service would then replicate the file on the server that would have reduced network bandwidth the most.

Once the primary server gives a file to another server for caching, the primary server can forget about the other server. The primary server's load will drop as clients begin to access the file from the new server. Should the primary server's load climb high enough that it must replicate the file again, the primary server will choose a different server to cache it on since the access patterns will have changed. Likewise, if the new server's load climbs high enough such that it must replicate the file, it will be cached in yet another place,

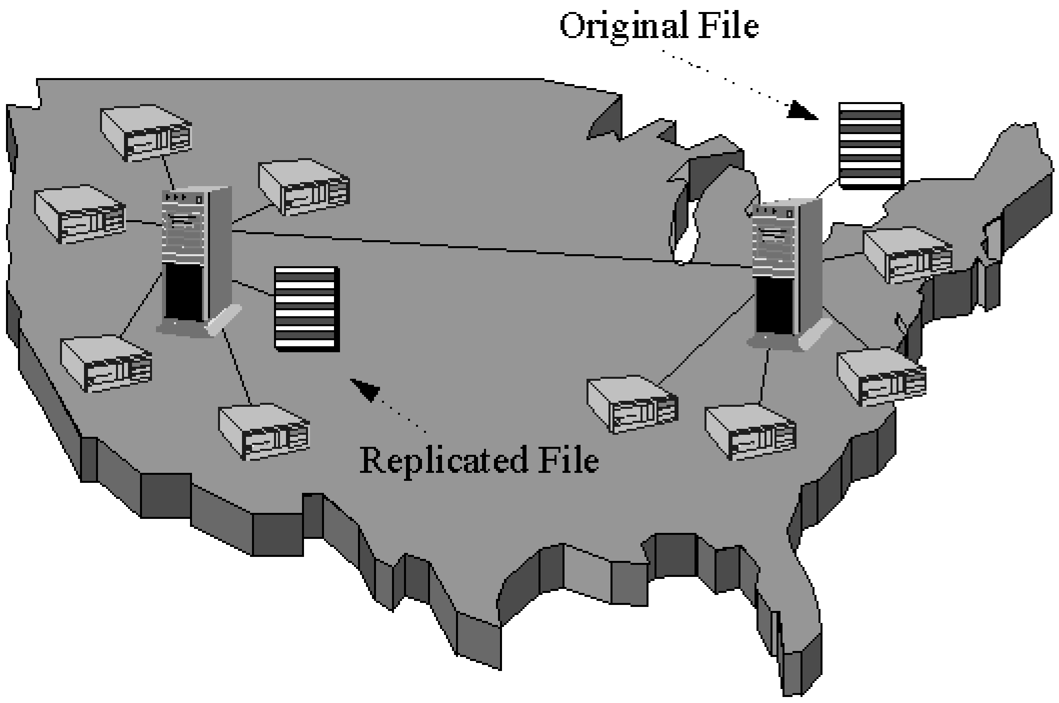

Figure 2: After file replication has taken place: the file has been replicated onto a west coast server so as to minimize network bandwidth.

because the access patterns for the new server will be very different than those for the old server.

There are two issues that must still be addressed for this scheme: file consistency and resource discovery. A server may determine that its copy of a file is out of date by using the *get-if-modified-since* HTTP request. This is an efficient way to both check consistency and to request the new file in the event it has been modified, but it is too expensive to use every time a file is requested.

Since weak-consistency should be acceptable for the Web, we are using a scheme developed for the Alex [5] file system. With the exception of dynamic pages which can not be cached we expect the Web to obey the same principle as FTP: the older a file is, the less likely it is to be modified. Therefore, the older the file that an HTTP server is caching, the less frequently the HTTP server must poll to check if its copy is still up-to-date. This is very efficient compared to checking for every request, and the client will be able to force a poll if it is essential to use the latest file.

As for resource location, there are several groups working on this problem. Until this problem is solved we are using a technique proposed by Blaze [3], which we call the "1-800 technique". Clients "call" the primary server to ask for the "server nearest you." This is not elegant, but it works because latency is currently more critical than bandwidth. The expense of querying a distant server once is amortized over the many local requests that are thereby made possible. Eventually there will need to be a cleaner solution so that all dependence on the original server can be removed. Otherwise we will face scalability and reliability problems.

4 Other Applications

Throughout this paper we have referred to HTTP servers and files. This was for the purposes of clarity, as well as to provide a focus for our research. We expect our results to be applicable to any wide-area distributed system, however; not just the World Wide Web. One application for geographical push-caching that we have in mind is to replicate not only data files but also services themselves.

A good example would be Archie [6], whose load problems are notorious. If Archie were to be written in a machine-independent network-service scripting language (e.g. Tcl [11]), its code could be replicated and cached just like a Web file. This might also be the answer to how to cache dynamic pages, such as those generated by cgi-bin scripts that are used to create Web pages on the fly.

5 Related Work

There is little work on caching in large-scale distributed systems outside of distributed file systems, since only in the past few years has the attention of the distributed systems community turned toward globally distributed systems such as the World-Wide Web and FTP. Several groups are working on similar problems, but most are focused on client-side caching. Only one group that we know of, at Boston University, is also working on server-initiated caching.[2]

One such client-side system is included in The Harvest system [4]. Their *object caching subsystem* provides a hierarchically organized means for efficiently retrieving Internet objects such as FTP and HTML files. The group at the University of Colorado at Boulder has also examined the resource location problem [7] and the consistency problem [13].

Blaze [3] has addressed caching in a large-scale system. His research focused on distributed file systems, but can be applied to FTP or the Web. Finally, the Alex system [5] was designed to provide a means of caching FTP files. Of these three systems, Blaze's design comes closest to our own since it supports replication when demand becomes too high, and because it lets clients use any nearby cache. It does not, however, provide the server with control over where replicas are placed.

6 Conclusion

We do not believe that geographical push-caching should replace client-initiated caching. These two techniques address the same problem on different time-scales and are therefore complimentary. Client-initiated caching responds quickly to local changes in a file's popularity, but can not alleviate a global rise in demand. Likewise, server-initiated caching can not cope very well with sudden, localized jumps in popularity, but is best suited to handling long-term file request trends.

We close with this reminder of why server-initiated caching is necessary, taken from the Web home page of the WebLouvre [12].

> Note: Starting end of October 1994, we are currently experiencing severe network problems on our 256 Kb school Internet connection. Please be understanding! I am still looking for a site willing to mirror the WebLouvre exhibit (30 Mb in all), preferably in the USA.

References

[1] T. Berners-Lee, R. Cailliau, J-F. Groff, and B. Pollermann. World-wide web: The information universe. *Electronic Networking Research, Applications and Policy*, 2(1):52–58, 1992.

[2] Azer Bestavros. Demand-based document dissemination for the world-wide web. Technical Report 95-003, Boston University, 1995.

[3] Matthew A. Blaze. Caching in large-scale distributed file systems. Technical Report TR-397-92, Princeton University, January 1993.

[4] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Mich ael F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, University of Colorado, Boulder, 1994.

[5] Vincent Cate. Alex - A global filesystem. In *USENIX File Systems Workshop Proceedings*, pages 1–12, Ann Arbor, MI, May 21 - 22 1992. USENIX.

[6] Alan Emtage and Peter Deutsch. Archie - an electronic directory service for the internet. In *Proceedings of the USENIX Winter Conference*. USENIX, January 1992.

[7] James D. Guyton and Michael F. Schwartz. Locating nearby copies of replicated internet servers. Technical Report CU-CS-762-95, University of Colorado at Boulder, 1995.

[8] Ari Luotonen and Kevin Altis. World-wide web proxies. In *Computer Networks and ISDN systems*. First International Conference on the World-Wide Web, Elsevier Science BV, 1994. available from 'http://www.cern.ch/ PapersWWW94/ luotonen.ps'.

[9] Mosaic-x@ncsa.uiuc.edu. Using proxy gateways. World-Wide Web. available from 'http: //www.ncsa.uiuc.edu/SDG/Software/Mosaic/ Docs/proxy-gateways.html'.

[10] nic.merit.edu. ftp://nic.merit.edu/nsfnet/ announced.networks/nets.unl.now.

[11] John K. Ousterhout. Tcl: An embeddable command language. In *USENIX Conference Proceedings*, pages 133–146, Washington, D.C., January 22-26 1990. USENIX.

[12] Nicolas Pioch. Le weblouvre. World-Wide Web. http: //mistral.enst.fr/ pioch/louvre/louvre.shtml.

[13] Kurt Jeffery Worrell. *Invalidation in Large Scale Network Object Caches*. PhD thesis, University of Colorado–Boulder, 1994.