Automatic Selection of Nearby Web Servers*

USC TR 98-688

John Heidemann Vikram Visweswaraiah

December 8, 1998

Abstract

Performance of global services such as the world-wide web can be improved by physically distributed replicas. Most replicated systems today request users to select manually a nearby replica (typically from a list of sites), yet users often are not willing or able to make an informed choice. This paper describes an approach to automatic replica selection which works without change to existing web clients and web and FTP servers. We describe the assumptions behind our approach and propose several metrics for estimation of network distance. We examine the feasibility behind this approach and compare the time required for replica-selection by each algorithm. Finally, we describe a small change to HTTP that would improve replica selection transparency.

1 Introduction

A strength of the Internet and the world-wide web is its international character. Web links are *location transparent*; users use the same kinds of names to access pages coming from a server in London as from one in New York. A user in Washington, D.C. will likely notice a performance difference between the pages from the two servers, though. Web links are not necessarily *performance transparent*, since frequently congested links such as trans-oceanic network connections can be a bottleneck. To address this problem many wide-area services are *replicated* (or *mirrored*) in different physical locations, allowing users to select a nearby replica to avoid heavily used network links. Replicated servers can balance load on individual network connections or servers and can reduce client–server propagation latency. Examples of replicated services are the Internet Movie Database with replicas in England and America, and the Comprehensive Perl Archive Network (CPAN) with about 20 replicas spread around the globe.

Two problems often occur with this kind of replication today. First, objects in the web have the server's hostname embedded in the URL. Although a hostname can point to multiple servers, selection between these servers is often random [12]. Random selection works well to balance server load when all replicas are co-located, but it does little to optimize performance when they are geographically distributed.

An alternative is to list all the replicas to users and ask them to pick one. This approach has a number of problems. Users are distracted by the extra step of selecting a server between presenting a name and getting a service. This step makes non-interactive use more difficult. Most importantly, a user is often not willing or able to make an informed decision. When presented with a list of 20 or 30 replicas, a hurried user will often simply select the first one. A careful user might select a replica in the same geographic area. But this guess might not be very good: replicas physically close can be very distant in network "geography". A replica served by the same Internet-service provider but hundreds of miles distant may be closer than one served by a different backbone provider in the same city.

Automatic replica selection is important to address these problems. A single URL specifies replicated data so that a user is not forced to select from a long list. When the URL is interpreted a replica near to the user is automatically selected. Automatic selection allows use of more sophisticated distance metrics often providing better performance than manual selection.

This paper examines the assumptions behind one approach to providing automatic replica selection for existing web browsers accessing existing web and FTP servers. This simple architecture accommodates sev-

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through FBI contract #J-FBI-95-185 entitled "Large Scale Active Middleware". The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Army, DARPA, or the U.S. Government.

The authors can be contacted at 4676 Admiralty Way, Marina del Rey, CA, 90292-6695, or by electronic mail to {johnh,visweswa}@isi.edu.

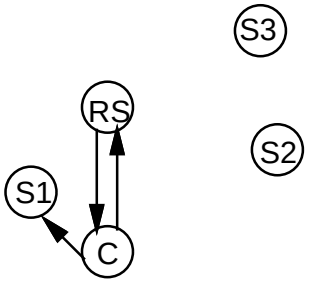

Figure 1: A client asks the replica selector for a nearby server.

eral different metrics of nearness. We describe and compare them. Prior work (discussed in Section 6) has compared the impact of different metrics on the file retrieval time, but for small retrievals the time required to perform replica selection can overwhelm overall retrieval time. We therefore evaluate the elapsed time of these algorithms in our common framework. Finally, we discuss a few simple modifications to existing systems which would allow our approach to provide better service.

2 Architecture

Three agents are involved in our approach to replica selection: a client, the replica selector, and a number of servers. Briefly, the client queries the replica selector which recommends a particular server, as shown in Figure 1. This section describes each component and its underlying assumptions.

The *client* in our system is an unmodified web browser. The user presents the URL for a replicated resource to the browser (by typing it in or clicking on a link). That URL queries the replica selector which then redirects the client to a nearby server to provide the data. We exploit the HTTP/1.0 temporary redirection message (return code 302) to perform this redirection [1]. All existing browsers we are aware of respond to this message by transparently resubmitting the request to the newly suggested URL.

Our basic architecture makes no assumptions about client location, but some replica selection algorithms are more stringent. We describe these in the next section.

The *servers* in our architecture are unmodified HTTP or FTP servers. An HTTP-knowledgeable client is required for the redirection mechanism, but the ultimate server can be any type supported by the client.

The servers replicate a common set of data. We assume a moderate number (2–50) of geographically distributed replicas, some of which will be closer to the client than others. With co-located replicas simpler solutions (for example, round-robin DNS records [12]) will probably provide a simpler solution than our system. Large numbers of replicas (20–100's) may require caching or hybrid approaches to replica selection to limit selection time; we discuss these in Sections 3.6 and 4.

We assume that all replicas are kept in loose consistency through an external mechanism. By loose consistency we mean that all replicas are sufficiently up-to-date or that data changes rarely enough that users do not care which the access. A number of existing replication mechanisms can keep web replicas consistent in this manner; examples include web and FTP mirroring which provide master/slave replication and peer-to-peer replicated filing systems such as Ficus [7] which provide wide-area, optimistic replication.

The *replica selector* does the actual work in our system. It handles HTTP-requests from the client and a returns redirect message after selecting a nearby replica. Several approaches to measure and approximate nearness are described in the next section. The selector can be implemented as a stand-alone server or with the common gateway interface under an existing web server. We use the later approach, although the straightforward adaptation to a stand-alone server would improve performance.

Logically there is a single replica selector per replicated service. To improve availability and distribute load we expect most services implement multiple (replicated) replica selectors with mechanisms such as round-robin DNS records [12]. Because the messages exchanged between the selector and the client and server are brief, the geographic location of selectors will not substantially affect performance and so the client's choice of selector is relatively unimportant.

Our architecture assumes that backwards compatibility is important enough that a solution must be provided without changes to either the clients or servers. This opportunity for incremental deployment distinguishes our work from other approaches such as Sonar [15] and most current proposals for URNs [18]. Although our system is backwards compatible with existing clients and servers, minor changes to the client can improve the user-model, performance, and accuracy of our system. We describe these changes in Section 4.

3 Algorithms

Replica selection is intended to improve web performance by directing clients to quickly responding servers. In general, a replica selector compares some value across different replicas and directs the client to the server with the minimum measure. This section examines why we believe propagation latency is a good approximation for nearness (the network component of response latency) and then considers several approaches to measure or approximate latency.

Hop-count, bandwidth, and propagation latency all might be compared as a measure of nearness between clients and servers. Our algorithms uniformly compare approximations of latency. Hop-count can be thought of as a simpler approximation of latency, thus latency seems preferable. Although Guyton and Schultz preferred weighted hop-counts to latency [8], the technical reasons for their choice seem to have been eclipsed by hardware improvements. We measure latency rather than bandwidth because latency can often be more easily measured and that low latencies are correlated with high bandwidths. Crovella and Carter found bandwidth a slightly better predictor of performance, but by a only by a small margin [5].

3.1 Random selection

A first approach makes no attempt to select a server intelligently. It simply chooses one at random. This approach is very fast, but better results should be possible. This algorithm is similar in effect to round-robin server selection with DNS records [12]. We consider it only for comparison to other approaches.

3.2 Domain-name-based approximation

Our next approach to very roughly approximate latency is to compare top-level domain names of the client and servers. A client is sent to a randomly chosen server with the same top-level domain name (TLD), to a server with a geographically adjacent TLD to the client, or to a random TLD.

The main advantage of this approach is that looking up domain names is very fast. It has several disadvantages, though. First, with 20–30 replicas there will be many TLDs that have no matching or adjacent servers. Second, generic TLDs (particularly .com) are international in scope and so have substantial variation in latencies. Third, several very large, flat domains (such as .com and .edu) often have many replicas which show substantial variation in latency. In such flat namespaces a name-based approximation may not be substantially better than random selection.

3.3 Geographic approximation

A third approach is to assume that geographic location approximates network latency. We can determine the geographic location of a site by mapping its hostname through a whois database [9], then mapping the telephone number or address in the database entry into latitude and longitude. Proposals have also been made to add geographic location to the DNS.

This approach addresses both the second and third problems of domain-name-based approximation. Hosts with the same TLD which are geographically distributed can be distinguished, distinguishing between hosts in an international TLD and in large, flat TLDs.

There are several problems with whois-based queries, though. Whois databases can be fairly slow (up to tens of seconds) and can require several queries to resolve a domain name to the administrative information. Whois records do not have a consistent format; heuristic methods are currently used to extract either the address or telephone number. Also, there are several whois databases covering different parts of the domain system. It's not always clear which database has authority over which portions of the namespace. Finally, geographic location doesn't necessarily equate with network locality.

Most of these problems would be addressed with DNS-based per-host physical location information. Unfortunately, maintenance of this information might be difficult and universal deployment will not occur in the immediate future.

3.4 Measurement of propagation latency

Rather than approximate latency one could directly measure latency from the client to each server. ICMP echo-request packets are one way to measure latency directly (as implemented in the "ping" utility). Unfortunately, the replica selector in our architecture does all selection measurements and the selector may be distant from both the client and the server.

We consider two approaches to avoid this problem. First, we assume that the replica selector is co-located with the client, perhaps by embedding it in the browser as a plug-in or through Java. Second, the selector can use IP-level source routing to direct its pings through the client to the server.

Several packets must be exchanged to get reproducible latency measurements, so a disadvantage of this approach is that the replica selection step may become noticeably slow to the user. Standard implementations of the Unix ping command measure latency with ICMP echo message, but they are not tuned to

take measurements accurately. Section 4.3 discusses our optimizations to make ping suitable for replica selection. Also, source-routed packets are often filtered out because of potential security issues. In Section 5.1 we describe how readily source routing is possible.

3.5 Routing information as an approximation

Routing protocols such as currently maintain network distance estimates. Unlike hostnames, routing is directly related to packet forwarding and so should be well correlated with network distance and latency. Routing information is also attractive because it is already maintained locally for packet forwarding.

Autonomous-system (AS) numbers provide a high-level association of hosts with "networks". Unfortunately, they suffer many of the problems associated with hostnames when applied to replica selection. A single AS-number can include a large variation of range of latencies and network distance.

BGP routing distances characterize network distance as hop-counts. Protocols to locate appropriate routers and extract this information still need to be developed if these metrics are to be used outside of routers.

We have not yet implemented routing-based replica selection algorithms.

3.6 Hybrid algorithms and server load

Combinations of these algorithms are possible. Hybrid approaches are particularly interesting when an time to evaluate a individual replicas is high. For example, one can quickly prune 100 replicas with a domain-name-based algorithm to 10–15 for input to the geographic-based algorithm, substantially improving run-time.

Although propagation delay predicts the network component of response latency, server load is also an important part of total response latency. Unfortunately, server load is difficult to obtain and changes rapidly. We have not yet experimented with combining server and network load measurements, but such an approach would support trading network utilization for server utilization, using remote but idle servers instead of near but busy ones.

4 Implementation Issues

This section describes several important implementation issues. Replica binding time has implications on the "user-interface" (what URLs are display and kept in bookmarks) of replication. We then consider implementation options if client or server changes were permitted. Finally we consider optimizations which improve algorithm performance.

4.1 Replica binding time

Replica selection translates a replica-independent name to a replica specific name. For example, `http://ref.isi.edu/RFC/rfc1945.txt` (the replica-independent name) might be mapped to `ftp://-ftp.isi.edu/in-notes/rfc1945.txt` for a user in Los Angeles where the site is located. Because replica selection is done with the HTTP redirect mechanism, this process consumes the replica-independent name. The user's browser will show the URL specific to the chosen replica.

Although this approach works well for one browsing session, if the user creates a bookmark to the post-selection URL, future uses of that URL will bypass replica selection and remain fixed on one replica. In some situations this behavior may be desired, but often re-selection would be preferred. A simple solution to this problem is possible if client-side changes are allowed. The replica selection can return a new field "Permanent-URL" with the redirection message. The permanent URL would vector through the replica selector, while the session URL would refer to the specific replica and be used only for one browsing session. The client browser would manage these two URLs in parallel. Continuing the above example, the replica selector would return a session URL of `ftp://ftp.isi.edu/in-notes/rfc1945.txt` and a permanent URL of `http://ref.isi.edu/RFC/-rfc1945.txt`. This approach is analogous to how some Unix shells maintain logical and physical pathnames to the current working directory when a user traverses a symbolic link.

4.2 Client or server changes

If one is allowed to change the client or server, the replica selector can be co-located at one or the other. Embedding the replica selector in the client is difficult because of the variety of clients, but browser plug-ins and downloaded Java code may make this approach feasible.[1] Adding the selector to the server is attractive since there are few servers to change (relative to the number of clients), but web server software is not always under the administrative control of the person replicated the data.

If the replica selector can be co-located with the

[1] Current Java security models limit network connections and protocols; these may need to be relaxed for some selection algorithms.

client or server, then availability of IP-level source routing (described in Section 5.1) is no longer an issue. It may also be possible to overlap data retrieval with replica selection, avoiding replica selection run-time such as communication and the overheads of a CGI-based selector (described in Section 5.2).

4.3 Implementation optimizations

Optimizations to latency measurement and caching of slow-to-retrieve data are both important to good replica selection operation.

Latency-based replica selection algorithms measure replica distance with ICMP echo messages. The Unix ping program is a typical implementation of this protocol. Two optimizations to ping are needed to make it useful for replica selection. First, traditional ping implementations send echo requests periodically, perhaps once per second. Since we average several pings to estimate round trip, these pauses can lead to substantial delay. Instead we issue additional echos as replies return. Second, since we may be considering distance to several replicas we take RTT measurements in parallel.

In addition to latency measurement optimizations, determining the geographic location of hosts has substantial cost. We determine location through whois queries on the domain name. Since whois queries take a substantial amount of time and location information changes very slowly, caching it at the replica selector can save substantial time. Caching latency measurements can also be helpful, but must be done with consideration to route stability [6].

5 Algorithm Evaluation

Section 3 described several approaches that can be used to select replicas. These algorithms have different levels of network and run-time overhead and can generate different quality results. This section evaluates these algorithms considering replica selection run-time. Measurement of propagation latency requires IP source routing, a service which is often disabled because of security concerns. We describe how frequently source routing is possible in today's Internet.

5.1 Source routing availability

Because we assume that the client and the replica selector are not co-located we measure latency with source-routed ICMP echo messages. Unfortunately, source routing is not universally supported across the Internet. Source routing at gateways is optional [2] and is often disabled because of security concerns.

Since the effectiveness of direct latency measurement is dependent on support for source routing we have measured how frequently source routing is available. To quantify source routing availability we attempted source-routed pings with three hosts: from a single client at ISI, a replica selector (also at ISI), and several different FTP servers. The list of FTP servers was the union of sites storing the Perl and Linux archives in Fall 1996 (a total of 100 sites from around the world). We attempted source-routed pings both from replica selector through the server to the client and from replica selector through the client to the server.

degree of source routing support	**hosts**
full	25
unidirectional	14
none	48
host unavailable	13
total	**100**

Table 1: Evaluation of support for source routing in the Internet. Experiment taken on 25 October 1996.

Table 1 summarizes of one day's measurement. Source routing worked in either direction (through client to server or through server to client) for about a quarter of the hosts; it failed for completely for about half. Since the client is at ISI and ISI supports source routing through our regional network provider (Los Nettos) to two backbone providers, we conclude that source routing is disabled at about 50% of servers we measured.

Because source routing can be prevented at any hop in the routing path and routes change over time [6] we made observations for 10 consecutive days. Table 2 summarizes these results. Full source routing was supported by 27 hosts (about 30%) at some time and never available at the other 64 hosts. Of hosts where source routing is possible at some time, it was disabled (presumably at some intermediate segment of the route) for 5 (18%) of these hosts occasionally (10–40% of the time).

From these measurements we conclude that source routing is suitable for experimental use in replica selection algorithms, but its availability is too limited for production use at this time.

5.2 Algorithm run time

Replica selection adds a certain overhead to web page retrieval. If the amount of data received is small, this overhead can substantially reduce or eliminate the

source routing availability	hosts
always supported	22
available about 90% of time	3
available about 60% of time	2
never supported	64
total available hosts	**91**

Table 2: Availability of full source routing over a 10 day period. Experiment taken from October 16–25, 1996.

algorithm	primary cost
random selection	C-to-S RTT
domain-name based	C-to-S RTT
geographic	time to do whois lookup
latency	RS-to-S-to-C ping time
routing information	time to query router

Table 3: Primary component of elapsed-time cost for each algorithm.

benefits of selecting a nearby replica. We therefore next examine the performance of the algorithms we consider, both in the abstract and as measured from our implementation.

Table 3 lists the primary components of the costs of each algorithm in terms of the client (C), server (S), and replica selector (RS). Some have minimal overhead (C-to-S RTT), while latency measurements require pings (which take several round trips) and geographic measurements require a whois lookup.

To evaluate the actual performance of these algorithms, we ran a series of tests with the the replica selection engine runs as a CGI program on a web server and services requests from a co-located client. The request, for all algorithms is for a file of size 10KB, chosen arbitrarily from the CPAN archives. A measurement program at the client records the time taken by the selection engine to return a nearby replica and the time taken for final retrieval.

Top level domain name matching and random selection both require no external information and so are fairly quick at about 2s (Table 4). When compared to the retrieval time, these figures are not high; the average ratio of redirection time to retrieval time are well under 1 for both algorithms. Much of this cost is due to CGI startup and could be eliminated by merging the replica selection algorithm with a web server.

Selection by geographical distance or by latency measurements are both substantially more expensive than a 10KB retrieval. Latency measurement is expensive because of the several RTT measurements required to estimate latency. Whois servers used for geographic distance are not optimized for speed and the occasional need for multiple queries leads to high variability in geographic location. While these algorithms are too expensive to be used for short retrievals, caching measurements (shown in the last line of Table 4) substantially improves performance.

algorithm	response time
random selection	1.87s (0.53)
domain-name based	2.04 (0.56)
geographic	8.97 (11.70)
latency (direct)	4.91 (1.07)
latency (source routed)	14.15 (9.34)
caching	2.86 (0.49)

Table 4: Replica selection times for each algorithm. Values are the mean of 200 runs; standard deviations are given in parenthesis.

In practice, Figure 4 suggests that source-routed latency measurements are much slower than direct measurements. This difference is much larger than would be expected due to additional propagation delay or network overhead. Instead, this difference arises because of lack of support for source routing on the Internet. Latency measurements are based on several round-trip packet exchanges. When packets are silently lost (as when source routed packets are rejected by a router) our latency measurement program takes several seconds to time out.

Based on these measurements we believe replica selection requires either caching or very simple algorithms (such as domain-name based) to be appropriate for short web documents.

6 Related Work

A number of proposals have been made examining replica selection in the Internet, both in general and for web applications. This section briefly reviews this work.

Guyton and Schwartz examine the cost of locating servers through several schemes [8] including forms of broadcast and triangularization [10]. They compare hop count and round-trip packet latency, choosing weighted hop-count because of poor clock resolution and RTT variability. They assume an active client and, in some cases, additional servers (beacons) deployed throughout the network. Their work focuses mainly on network bandwidth costs of replica selection rather

than end-user elapsed times as ours does.

Crovella and Carter have compared the effectiveness of hops count and latency distance metrics, concluding that latency is a better predictor of performance [5]. They also explored approaches to measure bandwidth, annotating links based on expected transfer times [3]. They find that replica selection based on bandwidth measurements performs substantially better than random selection, while bandwidth and best-of-5-RTT measurements perform comparably.

The CPAN Multiplex Dispatcher automatically matches file requests to a nearby replica of a Perl FTP archive [4]. It uses client and server TLDs as its nearness metric. Our work explores additional algorithms and algorithm performance. Commercial applications of similar approaches exist (for example, IBM's WomPlex and systems under development at other companies) but are difficult to compare because of lack of publicly available implementation details.

The primary goal of Uniform Resource Names (URNs) is to provide a "globally unique, persistent identifier" [18]. URN protocols are still being discussed, but most of the implementation proposals for URNs employ some form of replication to provide high availability. URNs are a complementary approach to our mechanisms and may eliminate the binding-time issues we observe (see Section 4.1); URN mechanisms could employ replica selection algorithms such as we describe to improve performance.

Like URNs, PURLs (Persistent URLs) provide a persistent identifiers [17]. PURLs are implemented with the same redirection mechanism we employ. Currently PURLs do not employ replication.

Another URN-like system is the Resource Cataloging and Distribution System (RCDS) [15]. RCDS provides a multi-step resolution system where URNs are converted to Location Independent File Names (LIFNs) which are mapped to nearby standard web or FTP servers with the help of a Sonar service. Sonar servers are assumed to be co-located with the clients and determine and cache round-trip time to the servers as an estimate of nearness.

Proxy caches are similar to web replication in that they make copies of data at distributed servers to improve performance [13]. A primary difference between caching and replication is that caching is typically on-demand (with lazy evaluation), client-based, and best effort. Replication is often pre-emptive, server-based, and pre-meditated. Replication can thus reduce start-up costs (since the data is pre-replicated) and can be purchased and deployed as desired by the information provider (since it is server-based and pre-meditated). Caching can allow easier deployment and can adapt better to changing use, however.

A number of replication schemes have been proposed or deployed for web use. Caching file-systems such as AFS are attractive [11] and have been used to for replicated, co-located servers [12]. Wide-area, peer replicated file systems such as Ficus support independent update at different sites [7, 16]. In addition, a number of application-level approaches to replication or mirroring exist (for example, Lee McLoughlin's program "mirror" [14]).

7 Future Work

Several directions exist for extensions of our work. Our current framework for replica selection makes it easy to experiment with new distance measures; we would like to explore additional metrics.

Our current architecture takes as an assumption that neither the client nor the server can change. Opportunities now exist to extend incrementally the client with plug-ins and Java. Client modification can offer more accurate distance estimation, simpler and more robust algorithms (for example, source routing could be avoided).

Another opportunity for client-side improvement is separate permanent and session-specific URLs as described in Section 4.1.

We would like to compare the different algorithms in actual use over the long term to estimate their effects on perceived performance.

8 Conclusions

This paper described an approach to replica selection suitable for use with existing web clients and web and FTP servers. The architecture supports a range of replica selection algorithms. We presented algorithms based on domain names, geographic, and two kinds of latency measurements. We compared the run-time performance of these algorithms to determine how they affect response latency. We also examined feasibility of source routing in the current Internet and explored the option of client-side software changes. As individual web services grow to serve millions of users we expect that automated replica selection mechanisms using approaches similar to those outlined in this paper will become necessary.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol—HTTP/1.0. RFC 1945, Inter-

net Request For Comments, May 1995.

[2] R. Braden. Requirements for Internet hosts—communication layers. RFC 1122, Internet Request For Comments, October 1989.

[3] Robert L. Carter and Mark E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report 96-007, Boston University, March 1996.

[4] Tom Christiansen. CPAN multiplex dispatcher. At http://www.perl.com/CPAN, 1996.

[5] Mark E. Crovella and Robert L. Carter. Dynamic server selection in the internet. In *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*. IEEE, August 1995.

[6] Ramesh Govindan and Anoop Reddy. Analysis of internet inter-domain topology and route stability. In *Proceedings of the IEEE Infocom*, pages 851–858, Kobe, Japan, April 1997. IEEE.

[7] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71, Anaheim, CA, June 1990. USENIX.

[8] James D. Guyton and Michael F. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of the ACM SIGCOMM*, pages 288–298, Cambridge, Massachusetts, August 1995. ACM.

[9] K. Harrenstien, M. Stahl, and E. Feinler. NICNAME/WHOIS. RFC 954, Internet Request For Comments, October 1985.

[10] Steven Michael Hotz. *Routing Information Organization to Support Scalable Interdomain Routing with Heterogeneous Path Requirements*. PhD thesis, USC, 1996.

[11] John Howard, Michael Kazar, Sherri Menees, David Nichols, Mahadev Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[12] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In *Proceedings of the First International World Wide Web Conference*, pages 155–164, May 1994.

[13] Ari Luotonen and Kevin Altis. World-Wide Web proxies. In *Proceedings of the Fourth International World Wide Web Conference*, December 1994.

[14] Lee McLoughlin. Mirror. A perl program to duplicate FTP-able directory trees, available from http://www.perl.com/CPAN/scripts/ftpstuff/mirror-2.8.tar.gz, 1996.

[15] Keith Moore, Shirley Browne, Stan Green, and Reed Wade. Resource cataloging and distribution service (RCDS). Technical Report 97-346, University of Tennessee, January 1997.

[16] T. W. Page, R. G. Guy, J. S. Heidemann, D. Ratner, P. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software—Practice and Experience*, 28(2):155–180, February 1998.

[17] Keith Shafer, Stuart Weibel, Erik Jul, and Jon Fausey. Introduction to persistent uniform resource locators. http://purl.oclc.org/OCLC/PURL/INET96, 1995.

[18] Karen Sollins and Larry Masinter. Functional requirements for uniform resource names. RFC 1737, Internet Request For Comments, December 1994.