A Hierarchical Internet Object Cache

Anawat Chankhunthod
Peter B. Danzig
Chuck Neerdaels
Michael F. Schwartz
Kurt J. Worrell

CU-CS-766-95

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **MAR 1995**	2. REPORT TYPE	3. DATES COVERED **00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE **A Hierarchical Internet Object Cache**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Computer Science,University of Colorado,Boulder,Co,80309**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **16**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

A Hierarchical Internet Object Cache

Anawat Chankhunthod
Peter B. Danzig
Chuck Neerdaels
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
+1 213 740 4780
{chankhun,chuckn,danzig}@usc.edu

Michael F. Schwartz
Kurt J. Worrell
Computer Science Department
University of Colorado - Boulder
Boulder, Colorado 80309-0430
+1 303 492 3902
{schwartz,kworrell}@cs.colorado.edu

March 1995

Technical Report 95-611
Department of Computer Science
University of Southern California
and
Technical Report CU-CS-766-95
Department of Computer Science
University of Colorado - Boulder

Abstract

This paper discusses the design and performance of a proxy-cache designed to make Internet information systems scale better. A hierarchical arrangement of caches mirroring the topology of a wide-area internetwork can help distribute load away from server hot spots raised by globally popular information objects, reduce access latency, and protect the network from erroneous clients. We present performance figures indicating that the cache significantly outperforms other popular Internet cache implementations at highly concurrent load and that indicate the effect of hierarchy on cache access latency. We also summarize the results of experiments comparing TTL-based consistency with an approach that fans out invalidations through the cache hierarchy. Finally, we present experiences derived from fitting the cache into the increasingly complex and operational world of Internet information systems, including issues related to security, transparency to cache-unaware clients, and the role of file systems in support of ubiquitous wide-area information systems.

1 Introduction

Perhaps because software developers perceive network bandwidth and connectivity as free commodities, Internet information services like FTP, Gopher, and WWW were designed without caching support in their core protocols. The consequence of this misperception now haunts popular WWW and FTP servers. For example, NCSA, the home of Mosaic, moved to a multi-node cluster of servers to meet demand. NASA's Jet Propulsion Laboratory wide-area network links were saturated by the demand for Shoemaker-Levy 9 comet images in July 1994, and Starwave corporation runs a five-node SPARC-center 1000 just to keep up with demand for college basketball scores. Beyond distributing load away from server "hot spots", caching can also save bandwidth, reduce latency, and protect the network from clients that erroneously loop and generate repeated requests [8].

This paper describes the design and performance of the Harvest [5] cache, which we designed to make Internet information services scale. Harvest caches can be deployed hierarchically for scalability's sake, and can service a highly concurrent stream of requests. The Harvest cache has been in use for one year by a growing collection of users across the Internet. It can also be paired with existing CERN and NCSA HTTP servers (httpd) to permit a site's HTTP server to scale to two orders of magnitude higher request rates.

Internet object caching differs from traditional applications of caches (such as hardware and file system caches) in several respects. First, Internet information systems exhibit much more locality than Muntz and Honeyman reported for file systems [17]. File systems typically exhibit little sharing, because users mostly work in their own private part of the name space and only occasionally reference shared directories. Much file system sharing is eliminated by replicating read-only executables and libraries on individual workstations for performance reasons. In contrast to distributed file systems, FTP, Gopher, and WWW were designed to facilitate read-only sharing of autonomously owned and managed objects. Hence, in contrast to file sharing studies, in 1992 over half of NSFNET FTP traffic was due to read-only sharing of objects [9]. Similar statistics have been recorded for WWW traffic [7].

Second, the cost of a cache miss is much lower for Internet information systems than it is for traditional caching applications. Since a page fault can take 10^5 times longer to service than hitting RAM, the RAM hit rate must be 99.99% to keep the average access speed at twice the cost of a RAM hit. In contrast, the typical miss-to-hit cost ratio for Internet information systems is 10:1, and hence a 50% hit ratio will suffice to keep average costs at twice the hit cost.

Finally, Internet object caching serves more than latency reduction. As noted above, a perhaps more important motivation is distributing load away from server hot spots.

Several issues arise when building an Internet object cache. First, cache policy choices are made more difficult because of the prevalence of information systems that provide neither a standard means of setting object Time-To-Live (TTL) values, nor a standard for specifying objects as non-cacheable. [1] Second, because it is used in a wide-area network environment (in which link capacity and congestion vary greatly), cache topology is important. Third, because the cache is used in an administratively decentralized environment, security and privacy are important. Fourth, the widespread use of location-dependent names (in the form of Uniform Resource Locators, or URLs) makes it difficult to distinguish duplicated or aliased objects. Finally, the large number of implementations of both clients and servers leads to errors that worsen cache behavior.

2 Design

This section describes our design to make the Harvest cache fast, efficient, portable, and transparent.

2.1 Cache Hierarchy

To reduce wide-area network bandwidth demand and to reduce the load on HTTP servers around the Web, caches resolve misses through other caches higher in a cache hierarchy, as illustrated in Figure 1. For example, several of the authors are running caches on their home workstations, configured as children of caches running in laboratories at their universities. Each cache in the hierarchy independently decides whether to fetch the reference from the object's home site or from the cache or caches above it in the hierarchy. The cache resolution algorithm also distinguishes *parent* from *sibling* caches. A parent cache is a cache higher up the hierarchy; a

[1] For example, it is popular to create WWW pages that modify their content each time they are retrieved, by returning the date or access count. Such objects should not be cached.

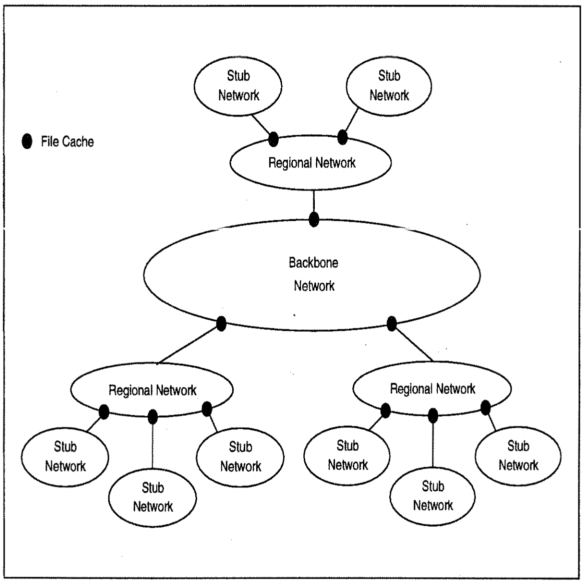


Figure 1: Hierarchical Cache Arrangement.

sibling cache is one at the same level in the hierarchy, provided to distribute cache server load. When a cache receives a request for a URL that misses, it performs a remote procedure call to all of its siblings and parents, looking to see if the URL hits any sibling or parent. It also tricks the referenced URL's home site into implementing the resolution protocol by sending a UDP "Hit" packet to the UDP echo port of the object's home machine.

A cache resolves a reference through the first sibling, parent, or home site to return a UDP "Hit" packet or through the first parent to return a UDP "Miss" message if all caches miss and the home's UDP "Hit" packet fails to arrive within two seconds. However, the cache will not wait for a home machine to time out; it will begin transmitting as soon as all of the parent and sibling caches have responded. The resolution protocol's goal is for a cache to resolve an object through the source (cache or home) that can provide it most efficiently.

Hierarchies as deep as three caches add little noticeable access latency, even when the object's home site fails to run a UDP echo server (e.g., machines behind firewalls frequently disable UDP echo). The only case where the cache adds noticeable latency is when one its parents has failed *and* the object's home is not running a UDP echo server. In this cases, references to this object are delayed by two seconds, the parent-to-child cache timeout [2]. Also, as the hierarchy deepens, the root caches become responsible for more and more clients. For this reason, we recommend that the hierarchy terminate at the first place in the regional or backbone network where bandwidth is plentiful.

2.2 Cache Access Protocols

The cache supports three access protocols: *encapsulating*, *connectionless*, and *proxy-http*. The *encapsulating* protocol encapsulates cache-to-cache data exchanges to permit end-to-end error detection via checksums and, eventually, digital signatures. This protocol exchanges an object's remaining TTL. The cache uses the UDP-based *connectionless* protocol to implement the parent-child resolution protocol. This protocol also permits caches to exchange small objects without establishing a TCP connection, for efficiency. While the *encapsulating* and *connectionless* protocols both support end-to-end reliability, the *proxy-http* protocol is the protocol supported by most Web browsers. In that arrangement, clients request objects via one of the standard information access protocols (FTP, Gopher, or HTTP) from a cache process. The term "proxy" arose because the mechanism was primarily designed to allow clients to interact with the WWW from behind a firewall gateway.

2.3 Cacheable Objects

The wide variety of Internet information systems leads to a number of cases where objects should not be cached. In the absence of a standard for specifying TTLs in objects themselves, the Harvest cache chooses not to cache a number of types of objects. For example, objects that are password protected are not cached. Rather, the cache acts as an application gateway and discards the retrieved object as soon as it has been delivered. The cache treats other URLs similarly, if the URL implies that the object is not cacheable. See the Harvest User's Manual [12] for details about cacheable objects. Large objects comprise one class of non-cacheable objects. It is possible to limit the size of the largest cacheable object, so that a few large FTP objects do not purge ten thousand smaller objects from the cache.

2.4 Unique Object Naming

A URL does *not* name an object uniquely; the URL *plus* the MIME [3] header issued with the request

[2] Reaping dead parents has not yet been implemented.

[3] MIME stands for "Multipurpose Internet Mail Extensions". It was originally developed for multimedia mail sys-

uniquely identify an object. For example, a WWW server may return a text version of a postscript object if the client's browser is not able to view postscript. We believe that this capability is not used widely, and currently the cache does not insist that the request MIME headers match when a request hits the cache. However, the cache does record the MIME header used to fetch each object.

2.5 Negative Caching

To reduce the costs of repeated failures (e.g., from erroneously looping clients), we implemented two forms of negative caching. First, when a DNS lookup failure occurs, we cache the negative result for five minutes. Second, when an object retrieval failure occurs, we cache the negative result for a parameterized period of time.

2.6 Cache-Awareness

When we started writing the cache, we anticipated *cache-aware* clients that would decide between resolving an object indirectly through a parent cache or directly from the object's home. Towards this end, we created a version of Mosaic that could resolve objects through multiple caches, as illustrated in Figure 2. Within a few months, we reconsidered and dropped this idea as the number of new Web clients blossomed (cello, lynx, netscape, tkwww, etc.).

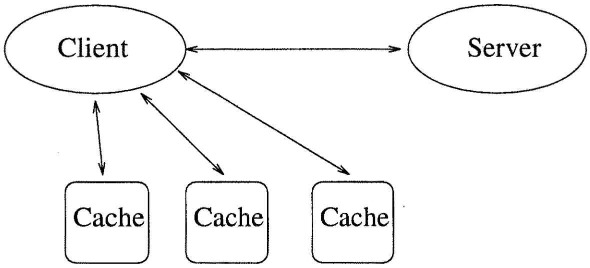


Figure 2: Cache-aware client

While no Web client is completely cache-aware, most support access through IP firewalls, as illustrated in Figure 3. Clients send all their requests to their *proxy-server*, and the proxy-server decides how best to resolve it.

There are advantages and disadvantages of each approach to cache-awareness. Cache-unaware clients have the clear advantage that there is no need to modify clients, they work using the proxy mechanism that users already understand, and they are needed by sites that wish to allow access to outside Web services from behind a firewall gateway. On the other hand, cache-aware clients make it possible to balance load closer to the client, avoid the single point of failure caused by proxy caching, and (as noted in Section 2.2) allow for a more efficient and capable protocol.

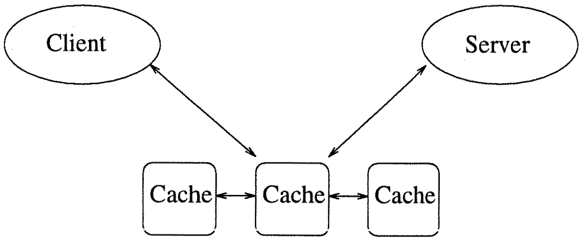


Figure 3: Proxy-caching client

2.7 Security, Privacy, and Proxy-Caching

What is the effect of proxy-caching on Web security and privacy? WWW browsers support various authorization mechanisms, all encoded in MIME headers exchanged between browser and server. The *basic* authorization mechanism involves clear-text exchange of passwords. For protection from eavesdropping, the *Public Key* authorization mechanism is available. Here, the server announces its own public key in clear-text, but the rest of the exchange is encrypted for privacy. This mechanism is vulnerable to IP-spoofing, where a phony server can masquerade as the desired server, but the mechanism is otherwise invulnerable to eavesdroppers. Finally, for those who want both privacy and authentication, a *PGP* based mechanism is available, where public key exchange is done externally.

For example, a *basic* authentication exchange follows the following dialog:

```
Client: GET <URL>
Server: HTTP:1.0 401 Unauthorized --
                authentication failed
Client: GET <URL> Authorization:
                <7-bit-encoded name:password>
Server: One of:
          Reply (authorized and authenticated)
          Unauthorized 401 (not authorized)
          Forbidden 403 (not authenticated)
          Not Found 404
```

tems [4], but was later adopted by HTTP for passing typing and other meta data between clients and servers.

Note that the basic and public key schemes offer roughly the same degree of security as Internet rlogin. Their authentication relies on client IP addresses, which can be spoofed, and they assume that intruders do not masquerade as real servers. Their authorization relies on user names and passwords, which can be snooped.

When a server passes a `401 Unauthorized` message to a cache, the cache forwards it back to the client and purges the URL from the cache. The client browser, using the desired security model, prompts for a username and password, and reissues the GET URL with the authentication and authorization encoded in the request MIME header. The cache detects the authorization-related MIME header, treats it as any other kind of non-cacheable object, returns the retrieved document to the client, but otherwise purges all records of the object. Note that under the clear-text *basic* authorization model, anyone, including the cache, could snoop the authorization data. Hence, the cache does not weaken this already weak model. Under the *Public Key* or *PGP* based models, neither the cache nor other eavesdroppers can interpret the authentication data.

Proxy-caching defeats IP address-based authentication, since the requests appear to come from the cache's IP address, rather than the client's. However, since IP addresses can be spoofed, we consider this liability an asset of sorts. Proxy-caching does not prevent servers from encrypting or applying digital signature to their documents, although encryption disables caching.

As a final issue, unless Web objects are digitally signed, an unscrupulous system administrator could insert invalid data into his proxy-cache. You have to trust the people who run your caches, just as you must trust the people who run your DNS servers, packet switches, and route servers.

2.8 Threading

For efficiency and portability across UNIX-like platforms, the cache implements its own non-blocking disk and network I/O abstractions directly atop a BSD *select* loop. The cache avoids forking except for misses to FTP URLs. [4] It implements its own Domain Naming System (DNS) cache and, when the DNS cache misses, performs non-blocking DNS lookups. As referenced bytes pour into the cache, these bytes are simultaneously forwarded to all sites that referenced the same object and are written to disk, using non-blocking I/O. The only way the cache will stall is if it takes a virtual memory page fault—and the cache avoids page faults by managing the size of its VM image. The cache employs non-preemptive, run-to-completion scheduling internally, so it has no need for file or data structure locking. However, to its clients, it appears multi-threaded.

2.9 Memory Management

The cache keeps all meta-data [5] about cached objects in virtual memory and it also tries to keep exceptionally hot objects loaded in virtual memory. However, when the quantity of VM dedicated to hot object storage exceeds a parameterized high water mark, the cache discards hot objects by LRU until VM usage hits the low water mark. Note that these objects still reside on disk; just their VM image is reclaimed. The hot-object VM cache is particularly useful when the cache is deployed as an *httpd-accelerator* (discussed in Section 3.1).

The cache is write-through rather than write-back. Even objects in the hot-object VM cache appear on disk. We considered memory-mapping the files that represent objects, but could not apply this technique because it would lead to page-faults. Instead, objects are brought into cache via non-blocking I/O, despite the extra copies.

Objects in the cache are referenced via a hash table keyed on URL. Cacheable objects remain cached until their cache-assigned TTL expires, they are evicted by the cache replacement policy, or the user manually evicts them by clicking the browser's "reload" button.

The cache keeps the URL and per-object data structures in virtual memory but stores the object itself on disk. We made this decision on the grounds that memory should buy performance in a server-bottlenecked system: the meta-data for 50,000 objects will consume 10MB of real memory. If a site cannot afford the memory, then it should use a cache optimized for disk space rather than performance.

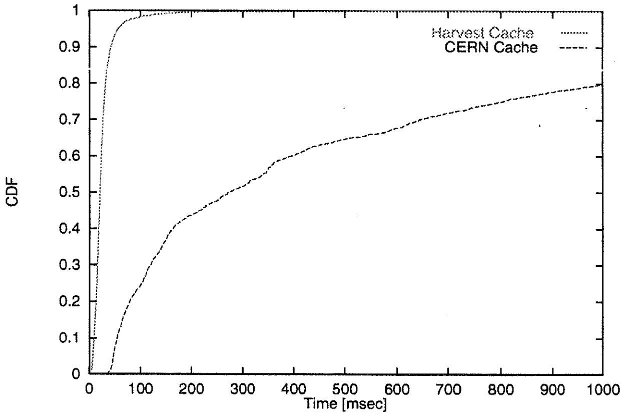
[4] We retrieve FTP URLs via an external process because the complexity of the protocol makes it difficult to fit into our select loop state machine.

[5] The meta-data consist of the URL, TTL, MIME header, file name where the object is stored on the local disk, and a number of other pieces of state information needed to manage the cache.

3 Performance

We now compare the performance of the Harvest cache against the popular CERN proxy-http cache [15], report the performance gain of deploying the Harvest cache as an *httpd-accelerator*, break down the savings as per the design decisions laid out in Section 2, and evaluate the latency degradations of using hierarchical caching.

In addition to its popularity, the CERN cache is a good candidate for performance comparisons because it is implemented as efficiently as any of the other Internet object caches, using standard UNIX programming techniques.



Measure	Harvest	CERN
Median	20 ms	280 ms
Average	27 ms	840 ms

Figure 4: Cumulative distribution of cache response times for hits, ten concurrent clients.

3.1 Harvest vs. CERN Cache

To make our evaluation less dependent of a particular hit rate, we evaluate cache performance separately on hits and on misses for a given list of URLs. Sites that know their hit rate can use these measurements to evaluate the gain themselves. Figures 4 and 5 show the cumulative distributions of response times for hits and misses respectively. We computed the cumulative response time of references to cache hits by faulting 2,000 objects of various sizes and types into the cache. Then, with ten client programs concurrently referencing these same objects in random order, we measured the response time for each reference. These figures show that the Harvest cache is an order of magnitude faster than the CERN cache on hits and on average about twice as fast on misses. These measurements were taken on a SPARC 20/61 with 128MB of memory. The reasons for the CERN cache overhead will be discussed in Section 3.2.

For misses there is less difference between the Harvest and CERN caches because response time is dominated by remote retrieval costs. However, note the bump at the upper right corner of Figure 5. This bump comes about because approximately 3% of the objects we attempted to retrieve timed out (causing a response time of 75 seconds)—either due to unreachable remote DNS servers or unreachable remote object servers. While both the Harvest and the CERN caches will experience this long timeout the first time an object retrieval is requested, the Harvest cache's negative DNS and object caching mechanisms will avoid repeated timeouts issued within 5 minutes of the failed request. This can be important for caches high up in a hierarchy because long timeouts will tie up file descriptors and other limited system resources needed to serve the many concurrent clients.

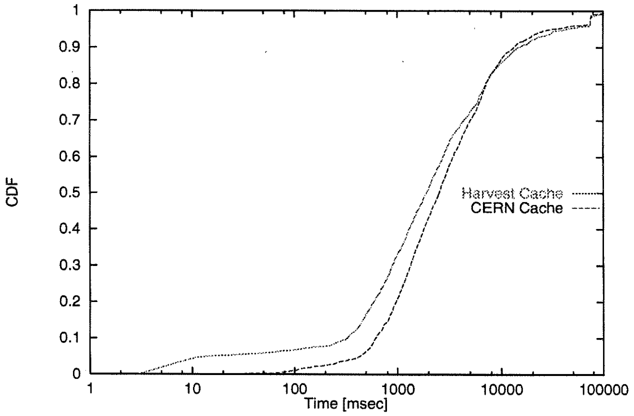


Figure 5: Cumulative distribution of cache response times for misses, ten concurrent clients, 2,000 URLs.

The order of magnitude performance improvement on hits suggests the idea of using the Harvest cache as an httpd *accelerator*. In this configuration, the cache fields all requests to an httpd server, forwarding the requests that miss to the real httpd. If a good fraction of the references are to cacheable objects, an order of magnitude performance improvement can be realized.

Figure 6 demonstrates the performance of a Harvest cache configured as an httpd-accelerator. In this experiment, we faulted several thousand objects into a Harvest cache and then measured the response time of the Harvest cache versus the native NCSA httpd.

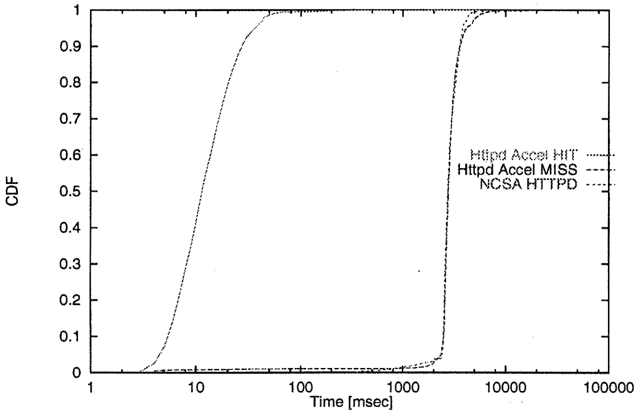


Figure 6: Response time of an httpd-accelerator versus a native httpd for a workload of all hits.

3.2 Decomposing Cache Performance

We now decompose how the Harvest cache's various design elements contribute to its ability, under concurrent access, to serve 200 small-objects per second in contrast to CERN's 3 small-objects per second. Our goal is to explain the roughly 260 ms difference in median and roughly 800 ms difference in average response time for the "all hits" experiment summarized by Figure 4.

The factor of three difference between CERN's median and average response time, apparent in CERN's long response time tail, occurs because under concurrent access, the CERN cache is operating right at the knee of its performance curve. Much of the response time above the median value corresponds to queueing delay for OS resources (e.g., CPU). Hence, below, we explain the 260 ms difference CERN's and Harvest's median response times (see Table 1).

Establishing and later tearing down the TCP connection between client and cache contributes a large part of the Harvest cache response time. Recall that TCP's three-way handshakes add 1 or 2 round trip transmission times to the beginning of a connection and add 1 or 2 round trip times to the end. Since the Harvest cache can serve 200 small-objects per second (5 ms per object) but the median response time as measured by cache clients is 20 ms, this means that 15 ms of the round-trip time is attributable to TCP connection management. This 15 ms is shared by both CERN and the Harvest cache.

We measured the savings of implementing our own threading by measuring the cost to `fork()` a UNIX process that opens a single file (`/bin/ls .`). We measured the savings from caching DNS lookups as the time to perform `gethostbyname()` DNS lookups of names pre-faulted into a DNS server on the local network. We computed the savings of keeping object meta-data in VM by counting the file system accesses of the CERN cache for retrieving meta-data from the UNIX file system. We computed the savings from caching hot objects in VM by measuring the file system accesses of the CERN cache to retrieve hot objects, excluding hits from the OS buffer pool.

We first measured the number of file-system operations by driving cold-caches with a workload of 2,000 different objects. We then measured the number of file-system operations needed to retrieve these same 2,000 objects from the warm-caches. The first, all-miss, workload measures the costs of writing objects through to disk; the all-hit workload measures the costs of accessing meta-data and objects. Because SunOS instruments NFS better than it instruments directly-connected file systems, we ran this experiment on an NFS-mounted file systems. We found that the CERN cache averages 15 more file system operations per object for meta-data manipulations and 15 more file system operations per object for reading object data. Of course, we cannot convert operation counts to elapsed times because they depend on the size, state and write-back policy of the OS buffer pool and in-core inode table. As a simple approximation, Table 1 assumes that disk operations average 15 ms and that half of the file system operations result in disk operations or 7.5 ms average cost per file system operation.

3.3 Cache Hierarchy vs. Latency

Hierarchy's benefits, reduced network bandwidth consumption, reduced access latency, and improved

Factor	Savings [msec.]
RAM Meta Data	112
Hot Object RAM Cache	112
Threading	36
DNS Lookup Cache	3
Total	264

Table 1: Breakdown of Performance Improvements

resiliency, come at a price. Caches higher in the hierarchy must field the misses of their descendents below them. If the equilibrium hit rate of a leaf cache is 50%, this means that half of all leaf references get resolved through a second level cache rather than directly from the object's source. If the reference hits the higher level cache, so much the better, as long as the second and third level caches do not become a performance bottleneck. If the higher level caches become overloaded, then they could actually increase access latency, rather than reduce it.

On our Sparc 20s, the Harvest cache can respond to over 250 pings/second, deliver 200 small objects/second, and deliver 4 Mbits/second to clients. At today's regional network speeds of 1Mbit/second, it is clear that Harvest caches, in any configuration, are not a performance bottleneck.

Figure 7 shows the response time distribution of faulting an object through zero, one and two levels of hierarchical caching. This figure is computed with ten concurrent clients, each referencing different objects against cold caches.

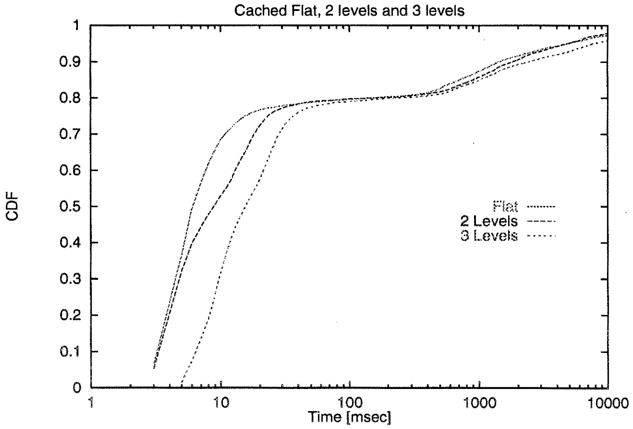


Figure 7: Effect of cache hierarchy on cache response time.

Recall that the Harvest cache keeps its metadata in memory, not on disk, and reserves a fraction of virtual memory to cache hot, small objects. Figure 8 shows the size of the cache server as a function of the number of cached objects, with the hot object cache size set to 20 MB. We see that the VM image quickly grows to the VM limit, 20MB, and then grows linearly at 750 bytes per object: 125 bytes of meta data and 500 bytes of stored MIME header. Discarding the MIME header is possible, if memory is tight, at some loss of transparency.

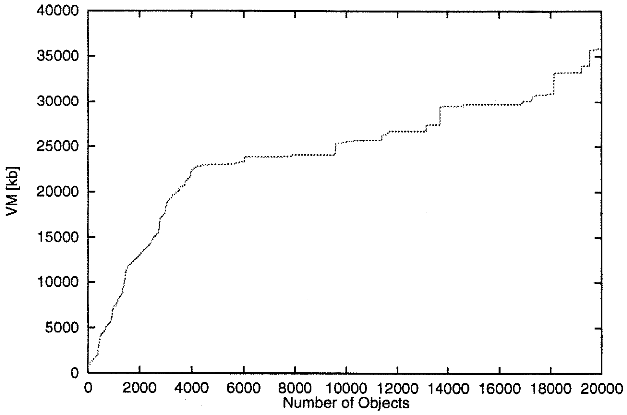


Figure 8: Size of meta-data plus hot objects as function of number of cached objects.

4 Cache Consistency

The Harvest cache, patterned after the Internet's Domain Naming System, employs TTL-based cache consistency. However, just like the Domain Naming system, the Harvest cache can return stale data. Unfortunately, HTTP, Gopher, and FTP provide neither a means for owners to specify TTLs, nor a protocol to pass TTLs from servers to caches and clients [6]. Hence, when the Harvest cache fetches an object from the object's home, it is forced to assign a default TTL. When the TTL expires, the cache discards the object. When a cache fetches an object from a parent or neighbor, it inherits the parent's remaining TTL.

Here we present measurements of WWW object lifetimes that indicate that no default TTL is short enough to avoid frequent access to stale data. Hence, we believe that a standard must emerge that lets object owners encode object TTLs in the object's MIME header. The standard might let the HTTP server assign a TTL as a function of the

[6] Netscape Communications Corp. is promoting active documents, which is the needed standard.

modification time, should the owner neglect to assign one.

Other consistency models are possible, such as a hybrid between a hierarchical variant of AFS's callback invalidation-based [13] scheme and a TTL-based mechanism. We discuss hierarchical invalidation below [7].

Object Lifetimes

Internet object lifetimes vary widely. To illustrate this, we periodically sampled the modification times of 4,600 HTTP objects distributed across 2,000 Internet sites during a three month period. We found that the mean lifetime of all objects was 44 days, HTML text objects and images were 75 and 107 days respectively, and objects of unknown types averaged 27 days. Over 28% of the objects were updated at least every 10 days, and 1% of the objects were dynamically updated. While WWW objects may become less volatile over time, the lifetime variance suggests that a single TTL applied to all objects will not be optimal. Figure 9 plots the average lifetime and range in lifetime, sorted by increasing average lifetime. Note that even within individual objects the lifetime of the revisions for a particular object are often variable. This means that not only will it work poorly to use a single TTL for a particular class of objects, but that even TTLs specified by object owners may be wrong.

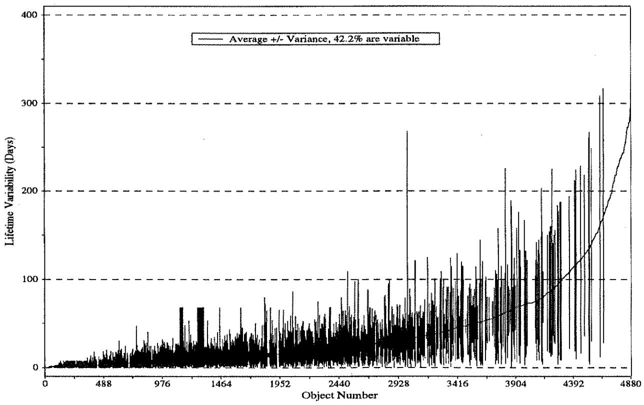


Figure 9: Average lifetime and range for a set of WWW objects.

In the absence of provider sites specifying TTLs for their objects, our observations lead us to suggest that caches could manage object TTLs via binary exponential backoff. When an object TTL expires, the object need not be flushed from the cache. Instead, the cache could refetch the object, compare it to the cached version, and then double the previous TTL if the object had not changed. [8] This does not help eliminate stale objects (and in some cases can lead to long lapses of inconsistent data, when an object is suddenly updated after its TTL has gotten large), but does provide dynamic TTLs without support from the object servers.

Hierarchical Invalidation

In large distributed environments such as the Internet, systems designers have typically been willing to trade some degree of cache consistency to achieve the important benefits of reduced server hot spots, network traffic, and retrieval latency. Here we investigate the possibility of achieving better consistency through invalidation, using the cache hierarchy to fan out invalidation messages. In particular, we determine the point where the bandwidth consumed by extra data transfers (for the TTL scheme) is the same as the bandwidth consumed by the extra invalidation messages and data transfers (for the invalidation scheme). At this point it can be determined what percentage of the TTL-scheme references are stale. When Internet users can no longer stand for that percentage of stale data references then it makes sense (bandwidth- and staleness-wise) to consider switching to an invalidation scheme.

To perform this analysis, we constructed a simulation and loaded it with a network topology intended to model a hierarchy of caches that might eventually exist in the Internet. We placed a top-level cache at each U.S. regional network and at each country outside the United States. We then placed lower-level caches according to the host count in each domain: domains with more than 200 hosts were given an internal caching hierarchy, domains with 50-199 hosts were given a single cache, and domains with fewer than 50 hosts directly referenced their parent's domain's caches. We then placed the caches into a hierarchy using a subset of Internet routes (collected using traceroute), so that domains were connected to top-level caches in proportion to the number of routes that actually connect each regional network.

[7] Reference [20] discusses hierarchal invalidation in detail.

[8] In the case of HTTP the fetch can be avoided by using the "if-modified-since" feature. Also, the last-modified time could be retrieved using the HTTP "HEAD" command.

We found that TTL-based consistency saved network bandwidth over hierarchical invalidation for TTLs in excess of five days, but that at this TTL, twenty percent of references are stale.

Note that achieving a well-working hierarchical invalidation scheme will not be easy. First, hierarchical invalidation requires support from all data providers and caches. Second, invalidation of widely shared objects will cause bursts of synchronization traffic. Finally, hierarchical invalidation cannot prevent stale references, and would require considerable complexity to deal with machine failures.

At present we do not believe hierarchical invalidation can practically replace TTL based consistency in a wide-area distributed environment. However, part of our reluctance to recommend hierarchical invalidation stems from the current informal nature of Internet information. While most the data available on the Internet today cause no problems even if stale copies are retrieved, as the Internet evolves to support more mission critical needs, it may make sense to try to overcome the hurdles of implementing a hybrid hierarchical invalidation mechanism for the applications that demand data coherence.

5 Experience

5.1 Transparency

Of our goals for speed, efficiency, portability and transparency, true transparency was the most difficult to achieve. Web clients expect caches and firewall gateways to translate FTP and gopher documents into HTML and transfer them to the cache via HTTP, rather than simply forwarding referenced objects. This causes several problems. First, in HTTP transfers, a MIME header specifying an object's size should appear before the object. However, most FTP and gopher servers cannot tell an object's size without actually transferring the object. This raises the following problem: should the cache read the entire object before it begins forwarding data so that it can get the MIME header right, or should it start forwarding data as soon as possible, possibly dropping the size-specifying MIME header? If the cache reads the entire object before forwarding it, then the cache may inject latency in the retrieval or, worse yet, the client may time out, terminate the transfer and lead the user to believe that the URL is unavailable. We decided not to support the size specification to avoid the timeout problem.

A related problem arises when an object exceeds the configured maximum cacheable object size. As implemented, our cache currently truncates and appends an error message to it. While this is an implementation rather than design limitation, it can arise on occasion.

Web clients, when requesting a URL, transmit a MIME header that details the viewer's capabilities. These MIME headers differ between Mosaic and Netscape as well as from user to user. Variable MIME headers impact performance and transparency. As it happens, the Mosaic MIME headers average about a kilobyte and are frequently fragmented into two or more IP packets. Netscape MIME headers are much shorter and often fit in a single IP packet. These seemingly inconsequential details have major impact that force us to trade transparency for performance.

First, if a user references an object first with Netscape and then re-references it with Mosaic, the MIME headers differ and officially, the cache should treat these as separate objects. Likewise, it is likely that two Mosaic users will, when naming the same URL, generate different MIME headers. This also means that even if the URL is a hit in a parent or sibling cache, correctness dictates that the requested MIME headers be compared. Essentially, correctness dictates that the cache hit rate be zero because any difference in any optional field of the MIME header (such as the user-agent) means that the cached objects are different because a URL does not name an object; rather, a URL plus its MIME header does. Hence, for correctness, the cache must save the URL, the object, *and* the MIME header. Testing complete MIME headers makes the parent-sibling UDP ping protocol expensive and almost wasteful. For these reasons, at present we do not compare MIME headers.

Second, some HTTP servers (possibly handcrafted servers implementing only a portion of the HTTP protocol) close their connection to the client before reading the client's entire MIME header. Their underlying operating system evokes a TCP-Reset control message that leads the cache to believe that the request failed. The longer the client's MIME header, the higher the probability that this occurs. This means that Mosiac MIME headers cause this problem more frequently than Netscape MIME headers. Perhaps for this reason, when it receives a TCP-Reset, Mosaic transparently re-issues the request with a short, Netscape-length MIME header. This leaves us with an unmaskable transparency failure since the cache cannot propagate

TCP-Resets to its clients. Instead, the cache returns a warning message that the requested object may be truncated, due to a "non-conforming" HTTP server.

Third, current HTTP servers do not mark objects with a TTL, which would assist cache consistency. With absence of help from the HTTP servers, the cache applies a set of rules to determine if the requested URL is likely a dynamically evaluated (and hence uncacheable) object. Some news services replace their objects many times in a single day, but their object's URLs do not imply that the object is not cacheable. When the user hits the client's "reload" button on Mosaic and Netscape, the client issues a request for the URL and adds a "don't-return-from-cache" MIME header that forces the cache to (hierarchically) fault in a fresh copy of an item. The use of the "reload" button is the most intrusive aspect of the cache to users.

Fourth, both Mosaic and Netscape contain a small mistake in their proxy-gopher implementations. For several months, we periodically re-reported the bug to Netscape Communications Corp., NCSA, and Spyglass, Inc., but none of these organizations chose to fix the bug. Eventually we modified the cache to avoid the client's bugs, forcing the cache to translate the gopher and FTP protocols into properly formatted HTML.

Note that the Harvest cache's encapsulating protocol (see Section 2.2) supports some of the features that the proxy-http protocol sacrifices in the name of transparency. In the future, we may change cache-to-cache exchanges to use the encapsulating protocol.

5.2 Open Systems vs. File Systems

The problems we faced in implementing the Harvest object cache were solved a decade ago in the operating systems community, in the realm of distributed file systems. So the question naturally arises, "Why not just use a file system and dump all of this Web silliness?" For example, Transarc proposes AFS as a replacement for HTTP [19].

AFS clearly provides better caching, replication, management, and security properties than the current Web does. Yet, it never reached the point of exponential growth that characterizes blossoming parts of the Internet infrastructure, as has been the case with TCP/IP, DNS, FTP, Gopher, WWW, and many other protocols and services. Why would the Internet community prefer to rediscover and reimplement all of the technologies that the operating systems community long ago solved?

Part of the answer is certainly that engineers like to reinvent the wheel, and that they are naturally lazy and build the simplest possible system to satisfy their immediate goals. But deeper than that, we believe the answer is that the Internet protocols and services that become widespread have two characterizing qualities: simplicity of installation/use, and openness. As a complex, proprietary piece of software, AFS fails both tests.

But we see a more basic, structural issue: We believe that file systems are the wrong abstraction for ubiquitous information systems. They bundle together a collection of features (consistency, caching, etc.) in a way that is overkill for some applications, and the only way to modify the feature set is either to change the code in the operating system, or to provide mechanisms that allow applications selective control over the features that are offered (e.g., using `ioctls` and kernel build-time options). The Internet community has chosen a more loosely coupled way to select features: a la carte construction from component technologies. Rather than using AFS for the global information service, Internet users chose from a wealth of session protocols (Gopher, HTTP, etc.), presentation-layer services (Kerberos, PGP, Lempel-Ziv compression, etc.), and separate cache and replication services. At present this has lead to some poor choices (e.g., running the Web without caching support), but economics will push the Internet into a better technical configuration in the not-too-distant future. Moreover, in a rapidly changing, competitive multi-vendor environment it is more realistic to combine features from component technologies than to wrap a "complete" set in an operating system.

6 Related Efforts

There has been a great deal of research into caching. We restrict our discussion here to wide area network caching efforts.

One of the earliest efforts to support caching in a wide area network environment was the Domain Naming System [16]. While not a general file or object cache, the DNS supports caching of name lookup results from server to server and also from client to server [9], using timeouts for cache consistency. The hierarchical caching structure of the

[9] The most common *resolver* client library implementation (BIND) does not provide client caching.

DNS was the basis for part of the design of the Harvest cache.

Sheltzer et al. investigated the use of name caching to reduce delays for remote operations [18]. The Harvest cache also caches name lookups.

AFS provides a wide-area file system environment, supporting whole file caching [13]. Unlike the Harvest cache, AFS handles cache consistency using a server callback scheme that exhibits scaling problems in an environment where objects can be globally popular. The Harvest cache implementation we currently make available uses timeouts for cache consistency, but we also experimented with a hierarchical invalidation scheme (see Section 4). Also, Harvest implements a more general caching interface, allowing objects to be cached using a variety of access protocols (FTP, Gopher, and HTTP), while AFS only caches using the single AFS access protocol.

Gwertzman and Seltzer investigated a mechanism called *geographical push caching* [11], in which the server choses to replicate documents as a function of observed traffic patterns. That approach has the advantage that the choice of what to cache and where to place copies can be made using the server's global knowledge of reference behavior. In contrast, Bestavros et al. [10] explored the idea of letting clients make the choice about what to cache, based on application-level knowledge such as user profiles and locally configured descriptions of organizational boundaries. Their choice was motivated by their finding that cache performance could be improved by biasing the cache replacement policy in favor of more heavily shared local documents. Bestavros also explored a mechanism for distributing popular documents based on server knowledge [3].

There have also been a number of simulation studies of caching in large environments. Using trace-driven simulations Alonso and Blaze showed that server load could be reduced by 60-90% [1, 2]. Muntz and Honeyman showed that a caching hierarchy does not help for typical UNIX workloads [17]. A few years ago, we demonstrated that FTP access patterns exhibit significant sharing and calculated that as early as 1992, 30-50% of NSFNET traffic was caused by repeated access to read-only FTP objects [9].

Finally, there have been several network object cache implementations, including the CERN cache [15], Lagoon [6], and the Netscape client cache. Netscape uses a 5 MB cache at each client, which can improve client performance, but a single user might not have a high enough hit rate to affect network traffic substantially. Both the CERN cache and Lagoon effort improve client performance by allowing alternate access points for heavily popular objects. Compared to a client cache, this has the additional benefit of distributing traffic, but the approach (forking server) lacks required scalability. Harvest is unique among these systems in its support for a caching hierarchy, and in its high performance implementation. Its hierarchical approach distributes and reduces traffic, and the non-blocking/non-forking architecture provides greater scalability. It can be used to increase server performance, client performance, or both.

7 Summary

Internet information systems have evolved so rapidly that they postponed performance and scalability for the sake of functionality and easy deployment. However, they cannot continue to meet exponentially growing demand without new infrastructure. Towards this end, we designed the Harvest hierarchical object cache.

This paper presents measurements that show that the Harvest cache achieves better than an order of magnitude performance improvement over other proxy caches. It also demonstrates that HTTP is *not* an inherently slow protocol, but rather that many popular implementations have ignored the sage advice to make the common case fast [14].

Hierarchical caching distributes load away from server hot spots raised by globally popular information objects, reduces access latency, and protects the network from erroneous clients. High performance is particularly important for higher levels in the cache hierarchy, which may experience heavy service request rates.

The Internet's autonomy and scale present difficult challenges to the way we design and build system software. Once software is released, both its merits and its bugs are with us forever. For this reason, the real-world complexities of the Internet make one face difficult design decisions. The maze of protocols, independent software implementations, and well-known bugs that comprise the Internet's upper layers, frequently force tradeoffs between design cleanliness and operational transparency. This paper discusses many of the tradeoffs forced upon us.

Software and Measurement Data

The Harvest cache runs under several operating systems, including SunOS, Solaris, DEC OSF-1, HP/UX, SGI, Linux, IBM AIX, and Apple AUX. Binary and source distributions of the cache are available from http://excalibur.usc.edu. The test code and the list of URLs employed in the performance evaluation presented here are available from http://excalibur.usc.edu/experiments. The reader can get information about the overall Harvest system from http://harvest.cs.colorado.edu/.

Acknowledgements

This work was supported in part by the Advanced Research Projects Agency under contract number DABT63-93-C-0052. Danzig was also supported in part by the Air Force Office of Scientific Research under Award Number F49620-93-1-0082, and by a grant from Hughes Aircraft Company under NASA EOSDIS project subcontract number ECS-00009, and by National Science Foundation Institutional Infrastructure Grant Number CDA-921632. Schwartz was also supported in part by the National Science Foundation under grant numbers NCR-9105372 and NCR-9204853, an equipment grant from Sun Microsystems' Collaborative Research Program, and from the University of Colorado's Office of the Vice Chancellor for Academic Affairs.

The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

We thank John Noll for writin the initial cache prototype. We thank Darren Hardy and Duane Wessels for all of the work they have done on integrating the cache into the overall Harvest system.

References

[1] Rafael Alonso and Matthew Blaze. Long-term caching strategies for very large distributed file systems. *Proceedings of the USENIX Summer Conference*, pages 3–16, June 1991.

[2] Rafael Alonso and Matthew Blaze. Dynamic hierarchical caching for large-scale distributed file systems. *Proceedings of the Twelvth International Conference on Distributed Computing Systems*, June 1992.

[3] Azer Bestavros. *Demand-Based Document Dissemination for the World-Wide Web.* Computer Science Department, Boston University, February 1995. Available from ftp://cs-ftp.bu.edu/techreports/95-003-web-server-dissemination.ps.Z.

[4] Nathaniel Borenstein and Ned Freed. RFC 1521: MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies, September 1993.

[5] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest information discovery and access system. *Proceedings of the Second International World Wide Web Conference*, pages 763–771, October 1994.

[6] Paul M. E. De Bra and Reiner D. J. Post. *Information Retrieval in the World-Wide Web: Making client-based searching feasible.* Available from http://www.win.tue.nl/win/cs/is-/reinpost/www94/www94.html.

[7] Hans-Werner Braun and Kimberly Claffy. Web traffic characterization: an assessment of the impact of caching documents from NCSA's web server. In *Second International World Wide Web Conference*, October 1994.

[8] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An analysis of wide-area name server traffic: A study of the Domain Name System. *ACM SIGCOMM 92 Conference*, pages 281–292, August 1992.

[9] Peter B. Danzig, Michael F. Schwartz, and Richard S. Hall. A case for caching file objects in internetworks. *ACM SIGCOMM 93 Conference*, pages 239–248, September 1993.

[10] Bestavros et al. Application-level document caching in the Internet. *To appear, Workshop on Services in Distributed and Networked Environments, Summer 1995. Available from ftp://cs-ftp.bu.edu/techreports/95-002-web-client-caching.ps.Z*, January 1995.

[11] James Gwertzman and Margo Seltzer. The case for geographical push-caching. *To appear, HotOS Conference*, 1994. Available as ftp://das-ftp.harvard.edu/techreports-/tr-34-94.ps.gz.

[12] Darren R. Hardy and Michael F. Schwartz. Harvest user's manual. Technical report, Department of Computer Science, University of Colorado, Boulder, Colorado,

February 1995. Version 1.1. Available from http://harvest.cs.colorado.edu/harvest/doc.html.

[13] John Howard, Michael Kazar, Sherri Menees, David Nichols, M. 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.

[14] Bulter Lampson. Hints for computer system design. *Operating Systems Review*, 17(5):33–48, Oct 10-13, 1983.

[15] Ari Luotonen, Henrik Frystyk, and Tim Berners-Lee. CERN HTTPD public domain full-featured hypertext/proxy server with caching, 1994. Available from http://info.cern.ch/hypertext/WWW/Daemon/Status.html.

[16] Paul Mockapetris. RFC 1035: Domain names - implementation and specification. Technical report, University of Southern California Information Sciences Institute, November 1987.

[17] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems - or - your cache ain't nuthin' but trash. *Proceedings of the USENIX Winter Conference*, pages 305–313, January 1992.

[18] A. B. Sheltzer, R. Lindell, and Gerald J. Popek. Name service locality and cache design in a distributed operating system. *Proceedings of the Sixth International Conference on Distributed Computing Systems*, pages 515–522, May 1986.

[19] Mirjana Spasojevic, Mic Bowman, and Alfred Spector. *Information Sharing Over a Wide-Area File System.* Transarc Corporation, July 1994. Available from ftp://grand.central.org/darpa/arpa2/papers/usenix95.ps.

[20] Kurt Jeffery Worrell. *Invalidation in Large Scale Network Object Caches.* Department of Computer Science, University of Colorado, Boulder, Colorado, December 1994. M.S. Thesis, available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/WorrellThesis.ps.Z.