Reducing Web Latency Using Reference Point Caching

Girish P. Chandranmenon
Bell Laboratories
girishc@dnrc.bell-labs.com

George Varghese
University of California, San Diego
varghese@cs.ucsd.edu

Abstract—
To reduce web access latencies, we propose a new paradigm for caching at the reference point of a document. If a document X is referred to from a document Y, information is cached at Y to reduce the latency of client accesses to X. We focus on two specific instances of this paradigm: caching IP addresses to avoid DNS lookups at clients, and caching information about documents to avoid setting up new connections. Avoiding DNS lookup saves over 4 seconds 10-12% of the time and avoiding connection setup saves 240ms on the average. These ideas enable new services such as search engines that return IP addresses to speed up search sessions, and caching at regional information servers that goes beyond the capabilities of today's proxy caching.

Keywords—**WWW, Latency, Precomputing**

I. Bandwidth vs Latency

The bandwidth of a transmission technology is the number of bits the technology can carry per second, while the latency is the time it takes to transfer one bit between the transmission endpoints. The improvements in network technology have been primarily in bandwidth rather than latency. In fact, improvements to gain higher bandwidth have, at times, increased latency.[1] As network technology becomes less dominated by bandwidth limitations, the number of round-trip times spent for protocol handshakes will become a dominant component in the overall transfer time. We use four observations to support this claim.

(1) Web traffic dominates the current Internet traffic, and most documents accessed are fairly small; for example the study of file access patterns conducted by SPEC [1] shows that 50% of accessed files are 5 Kbytes or less. (2) Even for large files, round trip delays will dominate if the bandwidth is high; for example, a 1Mbyte file transfer will require 36ms (latency at the speed of light) + 8.4ms (transfer time at 1 Gb/s) across the continental USA. (3) Real round trip times are much worse than speed of light calculations. Crovella and Carter [2] report that round trip latencies, as measured from a fixed host in their network in Boston University to 5262 random servers, have a median of 125ms and a mean of 241ms. Cheshire [3] further reports that latencies through current modems are very high (110ms and up, some even 300ms).[2] This is important because the majority of Internet access is from a PC through modem banks. (4) Studies of web caching [4], [5] report that web cache hit rates rarely, if ever, go beyond 70%.

Work done while at Washington University, St. Louis

[1] Examples include compression technology used in modems that gain bandwidth at the cost of waiting for more bytes from the application, and the use of pipelines in routers and end node adaptors.

[2] This is partly explained in [3] by the need to copy bytes to a serial port, and the temporary buffering imposed by modem compression algorithms.

Cheshire [3] argues that an access latency of around 100ms is necessary for applications to have an interactive feel. Even a target latency of around 200ms allows only 3 coast-to-coast round trip delays. Thus even in a perfect world, where bandwidth is cheap and the clients and servers are infinitely fast, users may still see large access latencies that limit their productivity.

We propose a general paradigm, *reference point caching*, for reducing RTTs by caching precomputed information about documents at points where the documents are referenced. Within this paradigm, we propose two specific mechanisms: reference point caching of *IP addresses* and *documents*. Caching of IP addresses reduces the latencies associated with DNS lookup — we show through measurements that avoiding DNS lookup saves 100-300ms on the average, and often on the order of seconds. Reference point caching of documents generalizes the scope of web caching to allow documents to be cached at any reference point instead of only on the path between the client and the server. It avoids connection setup and our measurements indicate that it can save 240ms on the average. Since a goal for interactive response is around 200 ms, these are significant savings.

The rest of the paper is organized as follows. In Section II we use measurements to quantify the major latency components of a web access. We use these measurements to motivate our proposed reference point caching paradigm in Section III. We then describe and evaluate two specific instantiations of reference point caching, namely caching IP addresses (Section IV) and caching documents (Section V). With each mechanism, we also discuss a few policies that can be used for the server and for the client.

II. Web Latency Components

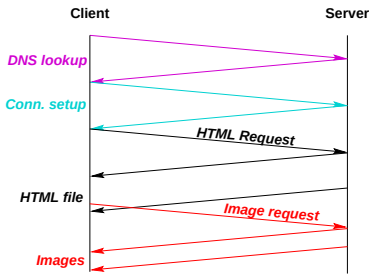


Fig. 1. Steps in a typical web access. Although in the figure it looks as if the DNS lookup is handled by the same server, it usually is not; potentially the DNS query can trickle down to the DNS server that is local to the web server's domain and such queries take much longer than a round trip.

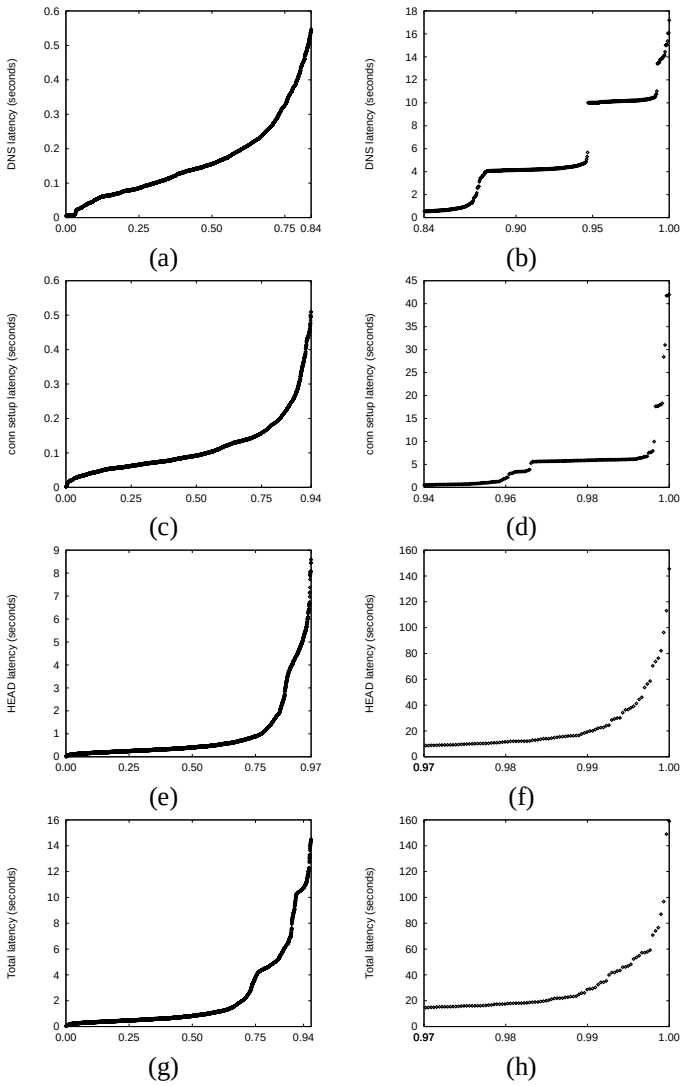


Fig. 2. Components of Latency

A Typical Web Access: Figure 1 illustrates the components of latency in a web access. A Web document is identified by a *Uniform Resource Locator (URL)*, which contains a machine name and the name of a file on that host (we use host and machine interchangeably). Given a URL, a browser such as Netscape or Internet Explorer, queries a Domain Name Service (DNS) server for the IP address of the host, establishes a connection to the host using its IP address, and sends a request for the file identified by the URL. The web server replies by sending the file to the client. The client browser receives the file and renders it on the screen. If the file contains images or inline data that are to be displayed with the web page, the client browser sends additional requests to retrieve them. The web page may contain *links* or references to other pages. Once the page is rendered on screen, if the user selects a link, the browser retrieves the new page using the same process.

The components of web access latency include the time taken for a DNS lookup, the time taken for TCP connection set up, the time to send the first request, the time to retrieve images etc. In order to quantify these components, we measured these latency components using a trace from Boston University[3] which contained logs of accesses from a CS Lab over several months. We extracted server names from this trace and made requests to server home pages using a HEAD request. A HEAD request, unlike a GET request, only returns response headers without including the body of the response. Using HEAD requests allowed us to focus on the latency component alone without the bulk data transfer component. We measured the DNS lookup time, the connection set up time, and the time between sending a request and receiving a response. These are reported in Figure 2.

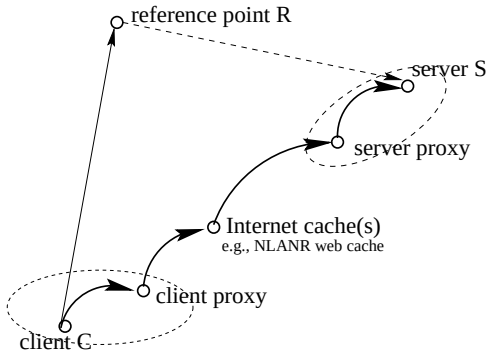


Fig. 3. Reference point caching allows caching of server S pages and S's IP's address at a reference point R whose page points to S. By contrast, standard HTTP caching is limited to client, Internet, and server proxies on the HTTP path between the client C and the server S.

In Figure 2, graphs (a) & (b) together show the time taken for DNS lookup for all the hosts. They are two parts of the same graph; in order to clearly illustrate the values at the extreme points, we have split the graph into two sections that use different scales. Similarly, (c) & (d) together show the time taken for connection set up, after the DNS lookup; (e) & (f) together show the time elapsed since the request was sent until the first byte of the response to arrive at the client, and (g) & (h) show the total time. The y-axis shows the time taken. The x-axis shows the cumulative fraction of the number of hosts. For example, 84% of the servers had a DNS lookup time of roughly 0.5 seconds or lower, 94% of the servers had a connection set up time of roughly 0.5 seconds or lower.[4]

From these graphs, we can see that the major latency component *today* is the time taken for the HEAD request. This indicates that server processing is the current bottleneck. However, there are several proposals to build low latency servers and the SPECweb96 benchmark results [6] cite several new servers that can saturate existing networks. When the server is no longer the bottleneck, DNS lookup times and connection set up delay will become the dominating components of access latency.

In this paper, we propose and evaluate new techniques to reduce DNS latencies and connection set up delays. Our solutions are based on a paradigm called *Reference Point Caching*, which we describe in detail in the following section.

TABLE I

Normal Proxy Caching vs Reference Point Caching	
Normal Proxy Caching	Reference Point Caching
Cache pages along the network path from the client to the server	Cache pages along the hyper link path to the page in the web graph
Clients have to fetch documents through the proxies	Clients can fetch documents directly from the origin server, thereby avoiding cache-dilution
No information is passed from the proxy to the client	Proxies pass information about the URLs they have cached by adding hints to the current page
Only one proxy at the client side or the server side	Can have many proxies at both sides

III. Reference Point Caching

Figure 3 illustrates reference point caching. Consider client C browsing through a page at R (called the reference point) which has a link to a page on server S. If C decides to browse the page at S, the standard mechanism for C is to first initiate a DNS query for the hostname S. If the DNS mapping is not available in the client DNS cache, the query is sent to the local DNS server in the client domain; if S is not cached in the local name server, the DNS query may be sent to the root server and then to the authoritative server for S in S's domain. Our measurements indicate that DNS query times can be very large, up to several seconds.

Once C gets the IP address for S, it tries to retrieve the page from S, by setting up a connection to S and sending the request for the page. This will cost at least two additional round trip times. If there are proxy caches involved along the path from C to S, the time for retrieval could be even larger in the worst case where the requested page is not found in any of the proxy caches along the path. C may first connect to a *client proxy* for the local domain (see Figure 3). If the client proxy does not have the page, it may attempt to obtain the page from Internet caches [5]. If the domain of S is served by a *server proxy*, the request will be directed to the server proxy. Finally, if the server proxy does not have the page cached, it will actually initiate a connection to S, retrieve the desired page from S, and then serve the request.

By contrast, in our new mechanism, the reference point R is allowed to have a cached copy of the page at S. If R has cached the page, it indicates that to C by annotating its link to S with a flag that indicates that the page is "locallyCached".[5] This flag is used by the client browser but is not displayed to the user. If the client browser decides to retrieve the cached page at R, the browser can do so using *the same connection it already has to* R. This not only avoids a connection set up delay but also makes it more likely that the congestion window of the TCP connection is high enough to sustain higher throughput.

The reference point is free to decide which of its link pages it wishes to cache. If R does not have a cached copy, R returns the IP address of S (and the DNS cache timer) with its home page. This time, C has to use the standard mechanisms. However, if the DNS cache timer is fresh, C can eliminate a potentially costly DNS query. Notice that in this case R is acting as a DNS cache for the hosts that serve its links. Table I summarizes the differences between normal proxy caching and reference point caching. These differences are also emphasized in the following subsections.

A. Potential for New Services

Reference point caching generalizes the scope of proxy caching at both the client and the server domains and extends DNS caching. At the client side, reference point caching allows us to have regional proxy servers for custom services such as travel, finance, etc. These proxies can selectively mirror pages from far away sites. The key difference is in their ability to select the links that they mirror. This has two benefits: the proxy can avoid caching huge documents, and it lightens the load at the proxy — when a document is not available at the proxy, the client fetches it directly from the origin server.

Reference point caching at the server can generalize the concept of server side proxies (HTTP accelerators). Typically, a server side proxy caches documents for a web server by taking over its identity, and then on, all web accesses must go through the proxy. Using reference point caching, a pool of web servers can cache documents for one another, and the ability to select the links that they want to cache gives them precise control over the load they want to support for other servers.

Reference point caching of IP addresses, where we store the IP address of the hostnames in the web pages, is primarily beneficial for search sessions, where the hostnames returned from a query are unlikely to be available locally in the client's domain. We discuss and evaluate this scheme in detail in the next section.

B. Design Issues:

In implementing our new mechanism, we have designed schemes that will allow us to stay compatible with existing web infrastructure. We use MIME headers and HTML tags so that those servers, proxies and clients that do not understand the modifications are unaffected.

Another issue is of maintaining consistency. For the cached documents, we depend on the current consistency mechanism that are in place, and for the cached IP addresses, we include the time to live field along with the address in order to preserve the DNS semantics, and we recommend periodically recompiling the pages so that they have the most consistent information.

As more and more web pages are being created by sophisticated editors, the extra step of compiling it before publishing can be incorporated into the web page design environment. Therefore, we believe the new mechanism will not cause a serious hurdle for convincing common web page designers to use it.

[3] This trace is available at the Internet Traffic Archive, http://ita.ee.lbl.gov/

[4] The analysis of DNS lookup in Section IV provides more detailed descriptions of how the lookup times are distributed etc.

[5] This can be implemeted using MIME headers or HTML tags in a backward compatible manner.

Lastly, a word about implementation: we implemented both caching IP addresses and caching documents using reference point caching. We used Apache 1.3a1 and libwww5.1b. Apache is the most popular web platform; libwww was the only available C implementation of the client side when we started the project. It is important to note that integration into existing software was easy (less than a few hundred lines of code) and very modular. For example, reference point caching of documents was easily added to libwww because libwww already had provision for supporting proxies; reference point caching can be considered to be adding a proxy *per URL*. We also built several tools for evaluating our ideas. These included a compiler that compiled web pages to add IP addresses and the headers required to support reference point caching, a grapher that recursively fetched web pages off a web site, and a prefetching server that could prefetch a selected set of web sites (such as a set of newspapers) at user requested periodic intervals. We will not discuss implementation in this paper due to lack of space; for more details of the implementation please refer to [7].

IV. Caching IP addresses

When a client contacts a server for the first time, it has to lookup the IP address of the server, since URLs provide only server names. This lookup can take hundreds of milliseconds if the address is not already cached locally. This is especially significant for the site names supplied by a search engine as the result of a query. These sites are unlikely to have been looked up before by the client.

In reference point caching of IP addresses, we recommend that a server, such as a search engine, include the IP addresses of all the hosts in the page it supplies. The server can preprocess static pages to include the IP addresses in the page, and while generating dynamic pages, it can include the IP addresses by looking up its local DNS. To avoid latency in generating these pages, the server should not include an address for a link if the address is not currently in its local DNS cache. To reduce the DNS traffic originating from search engines and to reduce latency, we recommend that the search engine should run a modified name server which prefetches IP addresses of frequently queried host names before they expire. This will ensure that a frequently used name translation will always be fresh in the search engine's DNS cache.

Since a search engine (or any page that caches addresses of its links) is effectively introducing a DNS cache, it must respect DNS caching semantics. Every DNS lookup returns the IP address of the name and a time to live value for the address. Our measurements show that the majority of IP addresses have a TTL of a day or higher. Even though address lifetimes are much higher in practice, in order to be correct, clients must respect the TTL supplied by DNS.

We attach the IP address to an anchor in the HTML page as a new tag — `DNS = "ipaddr expires-at"`. This enables those clients that understand this tag to use it, and others to ignore it. `ipaddr` is an ascii representation of the hex value of IP address. Thus, 128.252.169.2 will be 80fca902. `expires-at` is the expiration time of this IP address in seconds since the epoch (Jan 1, 1970).

When a browser wants to follow a URL to connect to a site, it can try to connect to the IP address stored with the URL if the address is still valid according to the DNS TTL. However, if the browser had already resolved the host name to a different IP address more recently, it should give priority to the address in its local cache over the address supplied with the link. Since we follow the DNS semantics strictly, a correct client implementation will not use a stale IP address any more than it does today with a DNS lookup.

In the following sections, we attempt to quantify the potential benefits obtained using reference point caching of addresses scheme, and the associated cost in terms of extra bandwidth needed.

A. Evaluation

First, we want to find out how long an average lookup takes, since that is how much a browser gains on every first access to a web site; and secondly, we want to find out what the typical life times (TTL values) of DNS cache entries are, since that is how often we have to compile a web page in order to maintain the accuracy of our hints.

We performed two measurements: one using 35712 unique host names collected from `www.yahoo.com`, by traversing the graph using a robot that followed only the links within yahoo.com, and the second using a set of server names from the BU trace. We looked up hostnames in the DNS using `dnsquery` (modified to report the DNS TTL value) from one site in the United States and another in Europe.[6]

The average DNS lookup time was 835ms in the Yahoo collection, which is what a client could save if Yahoo supplies IP addresses along with its query results. The results of the DNS query are shown in Figure 4(a). Figure 4(a) shows the time taken to get a response for the query, and (b) shows the TTL for the address record returned. In (a) the X axis shows the list of hosts in increasing order of lookup delay, in (b) it is the list of hosts in the increasing order of TTL. From the graph we can see that the majority of the delays are in the 100ms—1s range. There are many hosts that required around 4 seconds for their address lookup; there are hosts that took 10 or more seconds as well. Any delays above 100 msec should correspond to DNS queries that are not satisfied by the local DNS caches.

The experiment based on hostnames from Yahoo seems to indicate that DNS queries to sites followed in a search query are unlikely to be in the local DNS cache.

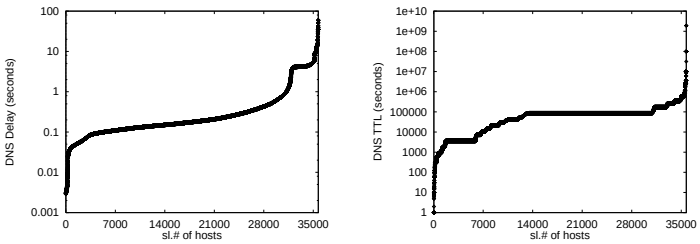
In the second measurement, we used a set of host names collected from a publically available[7] trace from Boston University [8].[8] The results for the BU trace were very similar to those from the yahoo database. Please refer to [7] for the exact results.

In both measurements, the majority of servers had a TTL of one day or higher which implies that the IP addresses stored in the compiled HTML files are likely to be valid for a day.

[6] The measurements from Europe was done only for the BU trace and not for the Yahoo names.

[7] This trace is available at the Internet Traffic Archive, http://ita.ee.lbl.gov/

[8] There are many other traces at the traffic archive, but most of them are server side traces. Server side traces are not useful for us, since we want to collect server names; server side traces only contain client side IP addresses. Among the client side traces available, only the Boston University trace has the server names. Other traces (e.g., Berkeley traces) have the IP addresses of both the server and the client encrypted for privacy.



(a) (b)

DNS Lookup Delay – Yahoo Collection			
Number of hosts with DNS lookup delay			
0-100ms	5426	800ms-1s	542
100-200ms	14933	1s-2s	589
200-300ms	4925	2s-4s	304
300-400ms	2246	4s-5s	2663
400-500ms	1397	5s-6s	375
500-600ms	778	6s-10s	244
600-700ms	556	10s-20s	258
700-800ms	413	20s-61s	63

Fig. 4. Result of DNS lookup for 35712 hosts collected from Yahoo. All lookups done from a site in the US. Note that the delay and the TTL axes are in logscale. The hosts axis is the index of hosts in the list of hosts sorted according to delay for (a) and according to TTL for (b).

Thus it should suffice to recompile once a day. In [4] the largest number of distinct requests seen per day was 3408 (at the NCSA server), and the average number of requests per day at the same server was 327073. Therefore, recompiling a document after it is first accessed for the benefit of future accesses would avoid compilation of pages that are infrequently read.

From the BU trace, we also attempted to compute the percentage of user clicks that result in a DNS query. Clearly, every user click may not result in an expensive DNS query, since the IP address may be cached by the local name server if some other client has accessed the page recently. A user click is followed by a fetch of one HTML page and fetches of many images within the page, all of which do not require additional DNS lookups. Besides, any client side log is likely to contain many accesses that are local to the domain. Caching these IP addresses in the pages is not necessary, since they do not result in any significant savings. We are interested in the first accesses to remote sites because it is likely that they require a full DNS lookup. Using the following method, we tried to extract the first accesses to remote sites because these are likely to incur an expensive DNS lookup: the original log (referred to as A) contained entries even for those requests that were satisfied from the local disk cache. We eliminated these, since these do not result in any HTML request at all, and analyzed the rest of the trace (referred to as B). We classified B further as 'without image requests' (called C); 'without local requests' (called D), and 'without images and local requests' (called E). We assume that, in each case, an IP address remains in the DNS cache for a day, after a lookup. These results are shown in Table II. The first column lists the total number of entries in the trace, the second column lists the number of entries that would have resulted in a DNS lookup assuming one-day life time for a result, and the third column lists its percentage.

Notice that for a standard trace, the percentage of overall clicks that result in a DNS lookup is very small (1.42%). However, if we consider only the clicks to remote hosts, adding IP addresses to web pages can help the client avoid expensive lookups

TABLE II

Analysis of DNS lookup in BU trace				
	Trace	total hosts	dnslookup required	%
(A)	original trace	1061901	15057	1.42 %
(B)	(A) – cached access	270042	14933	5.53%
(C)	(B) – image access	125178	13934	11.13%
(D)	(B) – local hosts	179150	14157	7.90%
(E)	(B) – local hosts – image access	73859	13188	17.86%

17.86% of the time. In other words, the local DNS cache was able to resolve queries for remote hosts only 82.14% of the time. This number could be even worse for sessions (e.g., a search session) that frequently visit remote sites and hence exhibit less locality of reference.

B. File Overheads

We measured the overhead of adding the DNS information to web pages using our University's web graph we collected using a tool we built. Since the each unique hostname adds 24 bytes of DNS information, the larger the number of unique links, the larger the overhead. The average size of an HTML document was 8642.5 bytes and the average number of URLs per page was 10.8. Therefore, the average overhead is 251.2 bytes or 3.0%. The page with highest overhead contained 1699 URLs; with an overall page size of 126889 bytes it resulted in an overhead of 32%. This maximum was for a very exceptional case of a page containing a large list of users. Also note that we have included the size of the HTML file alone in our calculations. Each HTML file in our sample had about 3 images with an average size of 11230.8 bytes. Therefore, the actual number of bytes fetched in retrieving a document is roughly 40 KB; the additional 251.2 bytes added by our address caching mechanism results in an overhead of only 0.6%. Also, we had measured this overhead using a smaller set of 36 pages we collected from Yahoo, the results are available in [7]. The average overhead there was 0.8%.

C. Policies

Caching IP addresses in the web pages themselves is just a mechanism. The benefits to clients and servers, however, depends greatly on the policies they implement. We list a few useful policies for each.

Server Policies: At the server side, the two important choices to make are: *i)* how to reduce DNS lookup overhead, and *ii)* when and how often to recompile pages and which ones.

We suggest that search engines run their own name servers, modified to prefetch frequently used IP addresses. An algorithm that prefetches the address for a host name H when its TTL runs down to half its original value, *if H has been used at least once since its last lookup*, should keep most of the frequently referenced IP addresses up to date. Note that the locality of reference on IP addresses will be more at the server, since any one of the clients accessing a page that contains such a hostname would force a refresh on the cached entry.

For other servers that want to use caching of IP addresses in statically generated pages, we suggest they recompile all pages at least once a day, and frequently used pages more often. We

suggest one day as the default period, because a majority of IP addresses have a DNS lifetime of 1 day.

Client Policies: Client policies decide which reference points to trust and when and how to use the cached value. We recommend that a list of trusted servers be configured into the browser, or discovered by authentication at the start of a session. For example, search engines or other well known sites may be easy to trust. It is also important to treat a cached IP address only as a hint. In other words, if the client fails to connect to the address listed in the web page, it should do a DNS lookup and try to connect to the IP address reported by the DNS. A client also has the choice of discarding cached addresses that are too old by its own policy. We discuss security concerns in the next subsection.

D. Interactions

Here, we discuss two interactions of IP address caching with other parts of the system: security and DNS load balancing.

Security: It might appear that we are introducing a new security risk by allowing IP addresses to be stored in the web pages. But, this risk exists today as well, *without the addition of our mechanisms*. That alone does not reduce its importance. There are initiatives already to secure DNS address translations using DNSsec [9]. Although we could use some of these mechanisms to make the caching more secure, we think a simpler scheme of letting users know (by showing a message on their screen) that the browser is following a cached IP address would suffice to alert them to the security risk. For most web accesses (unless they are giving out credit card numbers), people may not care to have the highest form of security. Also, if the use of cached addresses is limited to, trusted server domains and search engines, it is much easier for users to trust the referenced information. Finally, the browser can always retrieve pages that require data entry directly from the original URL by the standard mechanism.

DNS-based load balancing: Another interesting interaction is with DNS load balancing, where a server consists of multiple hosts, and DNS returns a set of host addresses; each new lookup returns a new ordering of these addresses to balance load. If we only cache one such address at a reference point, we defeat DNS load balancing. Our solution is to cache the multiple addresses subject to efficiency concerns (length of added information); we only have to transmit one of these (in say round robin order) when a request for a page comes in. Thus we may have to store multiple addresses but we only need to transmit one (important for transmissions over modem links where the length of the added information is an issue). Finally, if the number of addresses is too long, we do not need to cache any addresses at all.

In the case of search engines this is not a serious concern, since they should use a modified name server at their site which keeps track of all IP addresses, and returns one in the normal DNS fashion to the search engine while constructing the new page.

It is possible for some of the IP addresses that are returned by DNS to be inactive at any given point. So it is the client's responsibility to try all the addresses returned. In order to maintain correctness, if a client's use of a cached IP address fails to establish a connection, it should look up the IP address in its local DNS and try all the IP addresses returned.

Summary: DNS lookup is expensive. Since DNS lookup times often take several seconds and TTL values are typically a day or more, clients can benefit from a stored IP address for every hostname in a web page without violating DNS caching semantics, and with only a small amount of periodic compilation overhead at servers. The addition of IP addresses increases the file size by only 24 bytes for every unique host name in the page, and we have found that this overhead is less than 3.0% on the average. The modifications required to the client and the server are simple to implement, and the scheme can be deployed incrementally, since it is backward compatible. Typically, a browser using stored IP addresses can save 100-300ms, and sometimes 4–5 seconds in overall latency. While the ideas are described in the context of a web implementation, many of the implementation and evaluation ideas can be applied to other applications that do DNS lookups.

V. Caching Documents

As discussed in Section III, in reference point caching of documents, if a document X at server R refers to a document Y at server S, Y can be cached at R, thus allowing a client that accesses document X from R to retrieve document Y from the server R itself without making an additional connection to S. The only additional mechanism required in HTTP to implement this new form of caching is a way for the server to inform the client about the documents that it has cached. With this mechanism, reference point caching enables every web server to be a potential proxy cache.[9]

Generalizing the proxy mechanism to include all servers to be proxies has advantages: it avoids single point bottlenecks at a proxy, and each proxy can choose to cache and serve only the documents it wants. These two benefits, for example, create an opportunity to create regional caches for categories of web pages. The differences between reference point caching and conventional proxy caching are explained in Section III and Table I.

In addition, traditional proxies prefetch documents at run time based on the access patterns of the user. Typically, when a document X is fetched by a client, it is expected that some of the documents referred by X are likely to be referenced in the immediate future. The proxy fetches a subset of these documents, based on some heuristics, soon after the client has requested the document X. However, in reference point caching, the situation is more static; it is possible to prefetch documents periodically, based on the frequency of accesses of the referred documents collected at the origin server.

A. Incorporating into HTTP

Incorporating reference point caching to the current infrastructure boils down to informing the client about files that are cached at the current server, yet leaving to the client the decision of fetching from the cache or the origin server. We describe three mechanisms (see Table III).

[9] However, we only see this mechanism as being useful in more limited contexts such as regional information servers and server-side proxy pools where the security problems are easy to solve.

TABLE III

Different ways to incorporate reference point caching.	
Rewriting URLs to point to local copies of the document	Easier to work with older browsers, however, serious semantic conflicts could arise, such as users bookmarking the current URL and passing to friends. Not recommended.
Adding a flag to each URL inside the anchor field	Hard to do dynamic update as cache contents change. But has the least amount of byte overhead; highly recommended if the cache contents are static for the period recompilation.
Prepending potentially cacheable URL list as a header so that the server can process it when it is sent to the client	Good for more dynamic caches; significant byte overhead since the URL strings are replicated as a MIME header. A more sophisticated recompilation can reduce the byte overhead, with significant changes to HTML that are not backward compatible.

Rewriting URLs to point to local copies: One simple solution is for the server S to rewrite all the URLs in the files to point to the local copies. The advantage of this technique is that it is completely compatible with current browsers.

However, the disadvantages of rewriting URLs are many. First, since the browser no longer knows about the original URL, it causes semantic misperceptions. A user may bookmark this rewritten URL, or pass it to his/her friend as a reference to the document. This leads to problems since the rewritten URL is valid only when the document is cached and the original URL may be closer to the friend. Second, the rewriting of URLs does not allow a client to bypass the proxy, based on its own policy, and access the document from its origin server. We believe that a browser should know that it is following a cached copy. This gives the browser the option of informing the user, or automatically overriding the reference point cache based on user preferences and fetching the copy from the origin server instead.

Statically Attaching a Flag to an Anchor: Another method is to add a flag to each link reference (to each of `<A>`, `<LINK>`, etc.) embedded in the HTML text of the page. This is sufficient for the client to understand that the link has been cached at the server or not. In this scheme the preprocessor at the server has to statically compile in this information into the page. Since the set of cached documents is dynamic, a statically compiled in flag could convey stale information.

If the set of documents is not as dynamic, and the set changes only as frequently as the pages themselves are recompiled (for example once a day, for updating the IP addresses — see Section IV), this scheme is the most attractive, since it has the least byte overhead. All it needs is an extra bit and an integer time stamp (4 bytes) per unique URL.

Using a MIME header: This third mechanism is preferred for a more dynamic set of cached documents, despite its byte overhead. In this mechanism, in the preprocessing phase, we collect the names of all URLs that are not local to the current server (these are all potentially cacheable documents; all of these documents may not be in the cache at run time) in a MIME header at the start of the page. When a document is delivered by the server, this header is processed to reflect the availability of these documents in the cache.

The server should filter out those names that are not cached, and attach the expiry time of these documents to the header. Alternately, to reduce the number of bytes sent to the client, the server can create a bit map of these URLs — 1 indicates that the corresponding document is available in the cache, 0 indicates that it is not — and attach the earliest expiry time of those documents that it set the bit to 1. Even if a document is cached and its expiry time is too close to expiry, a server may choose to tell the client that it is not cached. A client upon receiving this document, can interpret the bit map using the implicit order of unique non-local URLs in the document.

A third way to process these headers (which we recommend), is to insert a place holder for the runtime bit where the URL is referenced in the page and save the offset of this place holder in the the MIME header. The server can use this offset to set this bit at run time, based on the presence of the URL in the cache, without parsing the entire file. One caveat of this scheme is that this kind of processing is possible only if the document itself is brought to user space, preventing the option of directly streaming the file from disk to the network for performance. In those circumstances statically compiling in the flag, or dynamically adjusting the MIME header seem to be the best strategy, depending on the degree of dynamism of the cache.

B. Evaluation

In this section, we evaluate the benefits of client and server side reference point caching. In order to save space, we have compressed the description of the details, and discussed the results.

Client Side Reference Point Caching: All the research in proxy caching has proved that client side caching provide very low access latencies, and reference point caching at the client side is no different. We have quantitatively verified that it does benefit to cache at the client side, and our experiements with a set of travel web pages from Yahoo, showed an average reduction of 72% in retrieval time. Our contribution is not in showing the performance benefits of client proxy caching but in generalizing the notion of client proxy caching to allow a different client proxy for each URL.

Server Side Reference Point Caching: The primary quantitative benefit we expect from reference point caching at the server is the reduction in latency due to avoiding new connections while browsing through a sequence of pages to reach a particular page. We evaluate this process using two experiments: one over the regular internet, and the second over a modem line, simulating a browsing session from home.

In the first experiment, we tried to trace one hyperlink path from a University's web page to the page of an individual home page at the University, repeating the experiment for 5 top university sites. We measured the DNS lookup time, connection set up time, and the web page transfer time, for each individual page along the path. We observed (more details are in [7]) that there were on the average 3 additional connections set up for every such hyper-path to a user's home page, and the connection set up time is roughly 80ms. Thus reference point caching can save potentially 240ms on the average for such browsing ses-

TABLE IV

Characterstics of the Test Pages					
HTML page	size of HTML	size of modified HTML	added bytes	# of imgs	total size of imgs
Univ.html	4536	5214	678	6	136501
Engg.html	4109	4511	402	3	85913
cs.html	2906	3191	285	3	72154
lab.html	3533	4546	1013	1	2436

Time taken to download using modem				
HTML page	Original Pages Over 4 conn (sec)		Compiled Pages Over 1 conn (sec)	
	Latency	Transfer	Latency	Transfer
Univ.html	1.35	71.75	1.39	68.11
Engg.html	1.39	39.41	0.48	41.32
cs.html	1.23	40.57	0.48	35.32
lab.html	1.35	2.10	0.64	2.06
Browsing time 28.8Kb/s modem	5.32 + 153.83 = 159.15		2.99 + 146.81 = 149.8	
Projected time 1Mb/s modem	5.32 + 4.43 = 9.75		2.99 + 4.23 = 7.22	

sions. This is in addition to the savings from avoiding the DNS lookup.

In the second experiment, we simulated browsing the web through a modem by transferring a chain of four web pages (each page is referenced in the previous one) shown in Table IV. We made two sets of measurements, one, where each of these pages were accessed using separate connections, thus simulating the case where the user is browsing the web using conventional web infrastructure; and two, where these pages (their compiled versions) were downloaded using the same connection, thus simulating the browsing using reference point caching. The results are shown in Table IV.

We note from the results that the access delay is still dominated by the transfer times. If we do reference point caching at the server for Univ.html, we do reduce indeed access latency considerably (2.99 versus 5.32 seconds). However, with an overall transfer time of around 150 seconds (the transfer time for the reference point version may be slightly smaller because of larger TCP windows), the improvement is only around 6%.

However, note that we have been using 28.8 kbps modems. Today, we can obtain 1Mbps cable modems, and the bandwidth available to homes is improving rapidly. Therefore, we project our measurements to 1 Mbps cable modems, and observe that the scaled transfer times are roughly 4.5 seconds. *Thus over a 1Mbps cable modem, the access latency improvement could be as high as 25%.*

C. Policies

The server decision to cache a referenced document could depend on the document size, its access frequency, and memory available to the server. The client decision to prefer a reference cache copy could depend on whether the reference point access is faster than accessing the original server.

Server side policies: Server side caching policies can be evaluated by modeling the problem using a graph annotated with access frequencies (once we obtain the right frequencies) and sizes of the pages. Each node in the graph represents a web page and is annotated with its size. The links are annotated with the percentage frequency of traversals. A server expands its graph

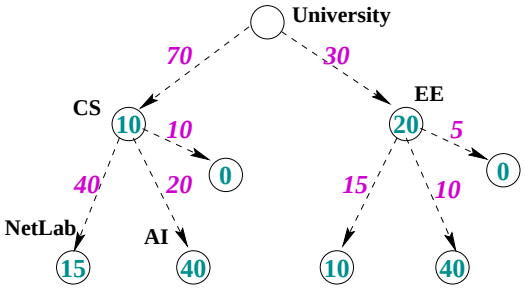


Fig. 5. Evaluating caching policies using a graph model. Edge weights represent frequency of access, and node weights represent the size of the documents. Zero sized documents represent accesses that terminate at that intermediate node.

starting at each local document. In Figure 5 we show an example graph from one such starting node.

Note that in reference point caching, if a node is cached all its ancestors should also be available at the server, either from the cache or locally at the server. If this is not the case, a client will never be informed that this node is cached and hence, it will never request for it. We could use any classical proxy caching policies such as LRU, LFU (Least Recently Used or Least Frequently Used) subject to this additional constraint. We also have explored a new policy that picks the nodes in the decreasing order of their density (access frequency/size). In order to apply any of these policies to reference point caching we used an algorithm that keeps track of the frontier of an expanding tree, similar to Dijkstra's shortest path algoirthm. On each iteration, it picks the vertex with the largest metric (density, frequency, recency) from the frontier, and expands the frontier to include the nodes reachable from the selected node. The details of this algorithm and our evaluations can be found in [7].

Client side policies: When a client accesses a page from a server R that implements reference point caching and a link L is marked as being available, it must decide whether to access L from R or from the original server. We should not unilaterally retrieve L from R; after all, S may be closer to the client than R! The factors that affect the choice between S and R are the latency from the client to the two servers and the two server loads.

Selecting between reference point and origin server to download the current page from is likely to incur very high overhead in terms of latency and statistics collection. So, we recommend a simple heuristic: retrieve the document from the reference point only if the reference point is in the same domain as the client or if it is in the same domain as the origin server. This also may alleviate some of the security concerns, since the client can trust its local servers, and the servers in the origin server's domain.

Summary: In this section, we proposed several ways to incorporate reference point caching into the current web framework. We also showed, using experiments and projected results, that reference point caching can reduce latency at the client side (as would any client proxy, but allowing more generality) and even at the server side (by reducing connection set up delay).

VI. Related Work

We have divided the related work into two sections: web performance and cache hierarchies. The related work on caching

TABLE V

Comparing Performance Enhacement Schemes					
	Scheme	# conn	# req.	# dns	disadvantage
1	HTTP/1.0 ([10]	n/doc	n/doc	1/server	too many connections
2	HTTP/1.1 ([11])	1/server	n/doc	1/server	too many requests
3	Prefetching	1/doc	n/doc	1/server	cache dilution
4	Client Proxies ([12])	1/doc	n/doc	1/server	single point bottleneck
5	Server Proxies ([13])	1/domain	n/doc	1/server	single point bottleneck
6	Caching IP addr at ref point	1/doc	n/doc	0	page modifications ; consistency problems
7	Caching docs at ref point	1/domain	n/doc	1/domain	page modifications

policies is omitted here due to lack of space.

Proposals for Web Performance Improvement: Table V gives an overview of several general schemes that have been proposed. The normal HTTP protocol (first row) makes n separate connections for each document (web page plus $n-1$ inline images), n separate requests per document, and one DNS lookup per server. Persistent HTTP [14] (incorporated into HTTP 1.1) reduces the number of connections to one per server (multiple documents at the same server can be accessed over the same connection). Client Prefetching (third row) does not reduce the number of connections, requests or DNS lookups. Client prefetching can, however, mask the access latency by obtaining the required documents *before* the user wants them. [15] reports an upper bound of 57% of reduction on access latency using prefetching. However, prefetching has the disadvantage of potentially prefetching useless documents, thereby wasting network bandwidth and diluting the (limited) client cache.

The next two rows evaluate server and client caching. Caching depends on locality patterns which may not hold as users explore new links; [16] found the locality of reference to be only 50% with high miss penalties for proxy cache misses. Also, [4] found that about a third of the files and bytes are accessed only once; this implies that caching at the server is tricky.

Client and server side proxies were discussed in detail in Section III. To recap, both kinds of proxies can be a bottleneck: they have to serve all documents requests through them, whereas our schemes have the option of being selective about what they want to serve.

A great deal of recent work (e.g., [5]) has focused on a hierarchy of web caches. If a web cache cannot serve a page, it searches for the neighbor that has the closest copy of the page using the Internet Cache Protocol. This is especially useful to cache pages reachable across long delay high bandwidth links (such as transatlantic or transpacefic links). While this is a good idea, it is unclear that it is sufficient. The caching hit rates shown in [5] show a cache hit rate of at most 60%. Thus it is worthwhile considering other caching mechanisms.

Summary cache ([17]) enables caches to share the documents: in this scheme, each cache stores the directory of its contents in all other caches. Reference point caching doesn't directly promote sharing of cached documents. However, the mechanisms used in summary caching for sharing directory information can also be used in informing the client about the availability of documents in the reference point cache of a server.

System of Caching Proxies: Harvest [13], Cachemesh [18], Cooperating Web Caches [19], and Adaptive Web Caching [20] are all proposed caching schemes that involve a hierarchy of caches that cooperate. However, they all follow the current model of proxy caching, that is each client goes thorugh exactly one proxy and the proxies fetch and deliver all documents. They all differ among themselves in the kinds of mechanisms they use for inter cache interactions.

Akamai[10] uses URL rewriting to redirect traffic to the caches. We already discussed the disadvantages of URL rewriting in earlier sections. Also, in typical cache hierarchies, the clients are not given an option to fetch the documents from the origin servers, whereas reference point caching will allow the clients to bypass the cache.

Some cache systems such as Microsoft's CARP, use hash functions on URLs[11] to distribute load among several caches. However, the client has no control over which cache is chosen for which document, nor does it have an option of fetching the document directly from the origin server. As far as we know, ours is the only scheme that allows multiple proxy caches based on individual URLs.

VII. Summary

In this paper we proposed a new caching paradigm called *reference point caching* whereby information about a document is cached at a point where the document is referenced. Our motivation is to reduce latency by avoiding unnecessary protocol steps. We proposed two specific instances of this scheme: caching IP addresses and caching documents themselves. We have evaluated both these ideas in detail, and quantified their usefulness by estimating the saved time. Avoiding DNS lookup saves 100-300ms on the average, and sometimes on the order of seconds. Avoiding connection setup can save 240ms on the average. Since a goal for interactive response is around 200 ms, these can be important savings especially when used together with fast servers.

All our techniques are based on three general principles: *precomputing*, *passing information between nodes* and *using hints*. Precomputing suggests doing every task as early as possible. The second principle suggests passing information between nodes in order to reduce overall processing; intuitively, the more components in a distributed system know about the other components, the more they can help one another in accomplishing a task. The third principle recommends using hints as they are easier to maintain and deploy incrementally. For

[10] http://www.akamai.com/

[11] http://www.aciri.org/floyd/hash.html

example, cached IP address translations of URL links are precomputed by the server and passed to the client on a request; finally, the client treats the IP address translation as a hint that can be verified, if needed, by a direct DNS lookup.

REFERENCES

[1] A. Carlton, "An explanation of the SPECweb96 Benchmark. SPEC white paper, 1996.," http://www.specbench.org/, November 1996.

[2] M. E. Crovella and R. L. Carter, "Dynamic server selection in the internet," in *In Proceedings of HPCS'95*, August 1995.

[3] S. Cheshire, "Latency and the quest for interactivity," *White paper commissioned by Volpe Welty Asset Management, L.L.C.,*, November 1996.

[4] M. Arlit and C. Williamson, "Web server workload characterization: The search for invariants," in *In proceedings of SIGMETRICS'96*, May 1996.

[5] D. Wessels, "Information resource caching faq," *http://ircache.nlanr.net/Cache/FAQ/ircache-faq.html*.

[6] Spec96, "Specweb96 benchmark home page," *http://www.specbench.org/osg/web/*, 1996.

[7] Girish Chandranmenon, "Reducing web latencies using precomputed hints," Tech. Rep. PhD Thesis. Technical report WUCS-99-18, Dept of Computer Science, Washington University in St. Louis, August 1999.

[8] C. R. Cunha and M. E. Crovella A. Bestavros, "Characteristics of www client based traces," *Boston University Technical report, BU-CS-95-010*, July 1995.

[9] "DNSSEC," *http://www.toad.com/ dnssec/*.

[10] T. Berners-Lee, R. T. Fielding, and Henrik Frystyk Nielsen, "Hypertext transfer protocol – http/1.0," *Informational RFC 1945*, May 1996.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T Bernes-Lee, "Hypertext transfer protocol – http/1.1," *RFC 2068*, January 1997.

[12] A. Luotonen and K Altis, "World wide web proxies," *Computer Networks and ISDN Systems*, vol. 27, no. 2, 1994.

[13] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *USENIX 1996 Annual Technical Conference*, January 1996, pp. 153–163.

[14] J. C. Mogul, "The case for persistent connection http," in *Proceedings of the ACM SIGCOMM '95 Symposium*, September 1995, pp. 299–313.

[15] T. Kroeger, D. Long, and J. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *USENIX Symp. on Internet Technologies and Systems, 1997*, December 1997.

[16] S. Williams, M. Abrams, C. R. Standridge, Ghaleb Abdulla, and Edward A. Fox, "Removal policies in network caches for world wide web documents," in *Proceedings of the ACM SIGCOMM '96 Symposium*, October 1996, pp. 293–305.

[17] Li Fan, Pei Cao, Jussara Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," in *Proceedings of the ACM SIGCOMM '98*, September 1998.

[18] Z Wang, "Cachemesh: A distributed cache system for world wide web," in *Web Cache Workshop*, 1997.

[19] R. Malpani, J. Lorch, and D. Berger, "Making world wide web caching servers cooperate," in *In 4th International World Wide Web Conference*, December 1995, pp. 107–117.

[20] S. Michel, K. Nguyen, A Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching: towards a new caching architecture," in *Computer Networks and ISDN systems*, November 1998.