Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers

Yair Amir, Alec Peterson, and David Shaw

Department of Computer Science
Johns Hopkins University
{yairamir, chuckie, dshaw}@cs.jhu.edu

Abstract. The explosion of the web has led to a situation where a majority of the traffic on the Internet is web related. Today, practically all of the popular web sites are served from single locations. This necessitates frequent long distance network transfers of data (potentially repeatedly) which results in a high response time for users, and is wasteful of the available network bandwidth. Moreover, it commonly creates a single point of failure between the web site and its Internet provider. This paper presents a new approach to web replication, where each of the replicas resides in a different part of the network, and the browser is automatically and transparently directed to the "best" server. Implementing this architecture for popular web sites will result in a better response-time and a higher availability of these sites. Equally important, this architecture will potentially cut down a significant fraction of the traffic on the Internet, freeing bandwidth for other uses.

1. Introduction

The explosion of the web has led to a situation where a majority of the traffic on the Internet is web related. In fact, in the beginning of 1995, web traffic became the single largest load on the Internet [1]. Today, practically all of the popular web sites are served from single locations. This necessitates frequent long distance network transfers of data (potentially repeatedly) which results in a high response time for users, and is wasteful of the available network bandwidth. Moreover, it commonly creates a single point of failure between the web site and its Internet service provider.

We present a new approach that uses web replication where each of the replicas resides in a different part of the network. The key contribution of this paper is how to have the client – the web browser – automatically and transparently contact the *best* replica, taking into account:

- Network topology: which replica is "closest" to the client, network-wise.

- Server availability: which servers are currently active.
- Server load: which server is currently able to return the most rapid response.

Most of the existing web replication architectures involve a cluster of servers that reside at the same site. These architectures improve performance by sharing the load between the different replicas, and improve availability by having more than one server. However, they cannot address the performance and availability problems embedded in the network.

Our architecture, in contrast, incorporates wide-area replication and automatic server selection to adequately address the above issues. Moreover, the architecture can be used in conjunction with both caching and cluster replication. The architecture includes three alternative methods to automatically direct the user's browser to the best replica.

- *The HTTP redirect method*: This method is implemented using web server-side programming at the application level.
- *The DNS round trip times method*: This method is implemented at the Domain Name Service (DNS) level, using the standard properties of DNS.
- *The shared IP address method*: This method is implemented at the network routing level, using the standard Internet routing.

Implementing this architecture for popular web sites will result in a better response-time and higher availability for these sites. Equally important, this architecture will potentially cut down a significant fraction of the traffic on the Internet, freeing bandwidth for other uses.

The reminder of the paper is organized as follows: The next section describes current web caching and replication methods. We then describe the alternative methods that automatically direct the web browser to the best replica. The next section describes our experience with the different methods. We then analyze and compare the methods, and draw our conclusions.

Due to lack of space, we will not detail a replication protocol to keep the consistency of the web server replicas. Several primary-backup [2], active replication [3], or lazy replication [4] techniques may be adequate.

2. Current web caching and replication methods

The two most common methods used to alleviate slow web response problems today are *caching* and *replication* (sometimes referred to as mirroring). In caching, we still have one main web server. On demand, pieces of the data are cached closer to the client. Subsequent requests for the same data are served from the cache rather than necessitating an access to the main web server.

Practically, web caching uses a special proxy on either the client or server side. This proxy acts as the go-between between the web browser and web server. Client side caching is when a client makes all requests through the proxy. The proxy then makes the actual request and caches the response for other clients (who are presumably at the same site as the original client). Many browsers (including Netscape's and Microsoft's) support this capability internally and perform client side caching on a per-user basis. This per-user caching is independent of any other caching done at the server or client side. Some caches used on a per-site basis include the Squid [5], Harvest [6], and Apache [7] caches.

In server-side caching, there are one or more caching servers that are front ends to the main server. When a document is requested, the caching server attempts to serve the document from its own cache, or from that of another caching server, before resorting to retrieving the document from the main web server. Frequently accessed documents will therefore be quickly cached, and thus the load for serving them will not fall onto the main web server. These caching methods do not conflict, and any or all of them may be used together as needed.

An intrinsic limitation for caching arises where the client is sending information to the server and asking for more than a static page (e.g. CGI scripts, or other server-side programming). Since the server's response to the request is dependent on the parameters sent along with the request, the response cannot be cached.

Another caching limitation is in the freshness of data. Given a particular document, it may be found in any (or all) of numerous caches between the client and the server. This creates a consistency problem as there is no to way guarantee the validity of any particular cache. No cache can guarantee complete freshness of data without the immediate expiration of cache data and the subsequent refresh from the master server. This clearly defeats the purpose of a cache, as every request will need to be served (however indirectly) by the master server.

In HTTP 1.0 [8] the cache management abilities were poor. This has been rectified in HTTP 1.1 [9], and authors now have the ability to exercise considerable control over the caching (or non-caching) of their documents, but this is not universally supported yet.

Web *replication* duplicates the web server, including its data and any ancillary abilities. A user can access any of the replicas, as any of them will provide a valid response. Replication addresses some of the limitations of caching as certain types of responses cannot be cached. These include responses to requests that require client-server interaction such as CGI scripts, database lookups, and server-generated pages.

Moreover, replication correctly handles situations such as advertising, where an accurate count of requests needs to be kept. For these situations, caching must be disabled: since the caches may not be under the web server's control, there is no way to count the number of actual hits (for more information, see [10]).

There is a problem that arises in both caching and replication: There is no way for the client to determine which is the "best" server to connect to. This paper examines a few possible methods that address this problem. These methods are applicable to both replicated and cached servers, as the benefits for connecting a client to the "best" replica are similar (though not identical) to those for connecting the client to the best cache.

3. Automatically locating the best replica

All of these methods involve the concept of the “best” server. This is a relative term, dependent on where the client is located on the network, and which of the collection of replicated servers is most able to answer the clients’ request. Weighting is also a factor – in the extreme case, server load may be more significant than network topology, as it is probably a better decision for the client to be sent to a more distant server than to one that is extremely overloaded, but closer.

Once the best server is determined, there are several possible ways to get the client there.

The HTTP redirect method

This is clearly the simplest method, but is the most limited as well.

The HTTP protocol since version 1.0 [8] has supported an “HTTP redirect”, where the client seeking a particular resource is told to go elsewhere for it. In combination with an intelligent server, this capability can be used to send the client to the proper place. Once this transfer of control takes place, the client will continue to communicate with the second server for the rest of the session.

As the HTTP redirect method works without any knowledge of the network topology and the location of the client within it, the HTTP redirect method is only really useful for connecting a client to the best server *overall*, rather than the best server for that particular client. This is significant, as it only addresses part of the problem. For example, even given a very fast and unloaded server, and a slower and heavily loaded server, the faster server may well not be the best choice for the client, if that client happens to be on the same network as the slower server.

There is also a major disadvantage in the HTTP redirect method in that there has to be a “central” server to accept the HTTP query and pass it off onto another server. This model has a single point of failure – a failure at this central server can immobilize the entire system. This problem can be partially addressed by using multiple central servers, but this leads to other problems.

The final problem with this method is that the user can inadvertently or otherwise defeat it. A user that notes down (“bookmarks”) the address of a web site after the redirection has taken place may get the wrong address noted down – that of one of the “children” rather than the central redirection server. Each further access would then go to that child directly, bypassing and defeating any optimization that is applied at the central server.

All is not negative for this method, and in fact there is one optimization that is possible here, but not with other methods – one where the actual content of the request is taken into account before the best server is determined. It can be determined that certain servers are better at serving some sorts of requests (for example, a server optimized for running CGI scripts, and another server optimized for serving static text). While both servers may be identical replicas data-wise, they still differ in their underlying capabilities.

The DNS round trip times method

This method utilizes the domain name system (DNS) [11], [12] to help determine which server is closest to the client. To explain this, a brief review of how a name-to-IP address lookup is done may be helpful:

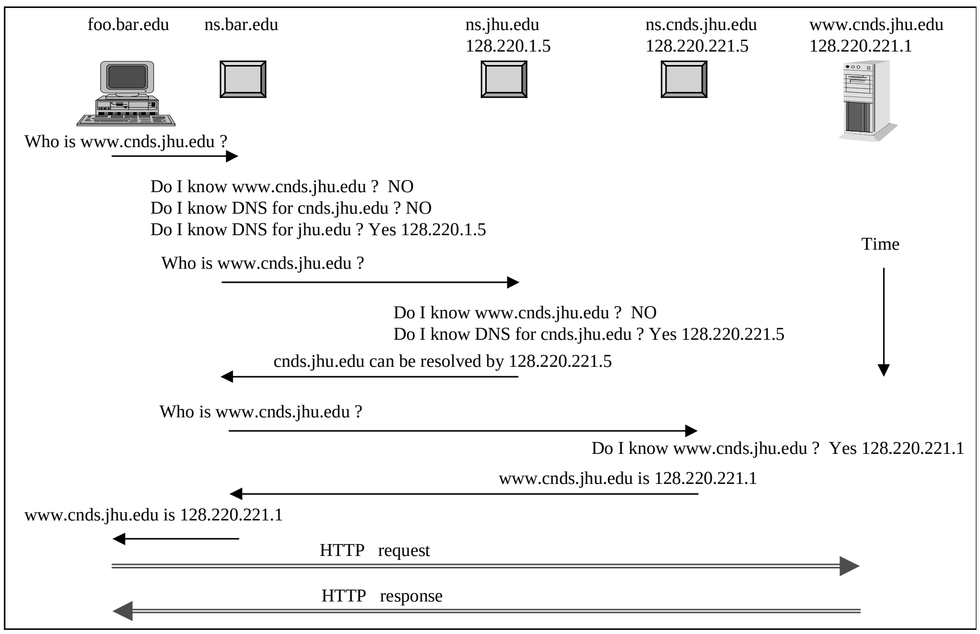

Fig. 1. Domain Name Service Resolution

Every server on the Internet has a fully qualified domain name (FQDN). FQDN are usually read from left to right, and each segment is rooted in the segment after it. <u>www.cnds.jhu.edu</u> is a machine known as "<u>www</u>" located in the <u>C</u>enter for <u>N</u>etworking and <u>D</u>istributed <u>S</u>ystems, which is part of the <u>J</u>ohns <u>H</u>opkins <u>U</u>niversity, which is an <u>edu</u>cational institution in the US. There is an assumed dot (".") at the end of each FQDN, which stands for the root domain at the top of the domain name tree. Barring any information cached on a local name server, a query for www.cnds.jhu.edu is sent to one of the root (".") name servers. This server (which due to the enormous size of the entirety of the domain name system cannot know every address), returns a list of name servers for "edu.", and the process is repeated, with one of the "edu." servers returning a list of name servers for "jhu.edu.", and one of the “jhu.edu.” servers returning a list of servers for “cnds.jhu.edu.”, one of which then returns the requested IP address(es) for “www.cnds.jhu.edu”. Caching is used to reduce the difficulty of the problem. If at any stage, a server has the address requested (or the address of a name server authoritative for that domain) cached, it will return it directly, short-cutting the above process.

This method takes advantage of the fact that each local server, when querying a remote server, keeps track of the round trip time (RTT) of packets to that server. This is done for optimization reasons, as over time the local server will try and favor those servers that respond faster. Given otherwise similar servers, this usually indicates a server that is closer network-wise.

This optimization can be leveraged for the problem being discussed by changing one concept in the DNS – normally, all servers for a particular domain (called a “zone” in DNS terms) carry the same data, but this is *not* a requirement. From the perspective of a querying server all that is significant is whether the remote server is in the chain of authority for the domain we are querying.

Thus, if we have many name servers, each serving an *authoritative, but different* IP address for www.cnds.jhu.edu, the web server the client is directed to is dependent on which name server the client’s local name server happens to ask. Since, as already established, the client’s local name server tends to favor those remote name servers that are closest to it, an excellent system for adding the last missing piece of information – that of “where is the client relative to the server” emerges.

Thus, by dynamically updating [13] each particular name server with the address of the web server that we want to direct users in that particular area of the network to, users from that area will tend to go there.

This method handles web server failures very well. In the case of a web server failure, the name server for that web server can be instructed to not give out the IP address of that server until that web server returns to service. Indeed, simply stopping all name service on that machine will accomplish the same thing, as a name server failure is handled automatically by the name server system – in the case of a non-responsive name server, other servers for that zone are queried in RTT-order until an answer is received.

A major advantage of this method is that it does not entail *any* modification of the network infrastructure to function. It uses readily available building blocks (standard unmodified name and web servers), and is completely platform-independent.

It can be pointed out that using the DNS optimization gives a good idea how close the chosen name server is to the client’s name server, but that may say nothing about where the web server is located relative to the client itself. It is assumed that when the system is set up, the name server and the web server will be placed in close network proximity. In addition, the local name server the client is querying is expected to be in close network proximity to the client, as is the usual setup on the Internet today.

The RTT method the DNS uses to determine the best name server for a particular zone is not necessarily optimal, but over time is a reasonable approximation. The time it takes to establish this reasonable approximation is the chief problem with this method. In order for the local name server to query the fastest responding name server, it needs to try them all. Thus, until all servers for the remote zone are queried at least once, the local server may query a server that is very far away network-wise, and thus get the IP address of a web server that is very far away from the client. The local server is unlikely to repeat its error, as this distant name server will likely have a larger RTT value than the other servers in the list will, but for this one connection the harm has already been done.

Due to the necessity for the client's local name server to cycle through and establish timings for all of the name servers for the zone in question, the web server must receive a fairly large quantity of hits from a relatively concentrated area (generally one institution such as an ISP or a university) for this method to work. Thus this method is fairly unlikely to work in the short term. It *will*, however, work in the long term.

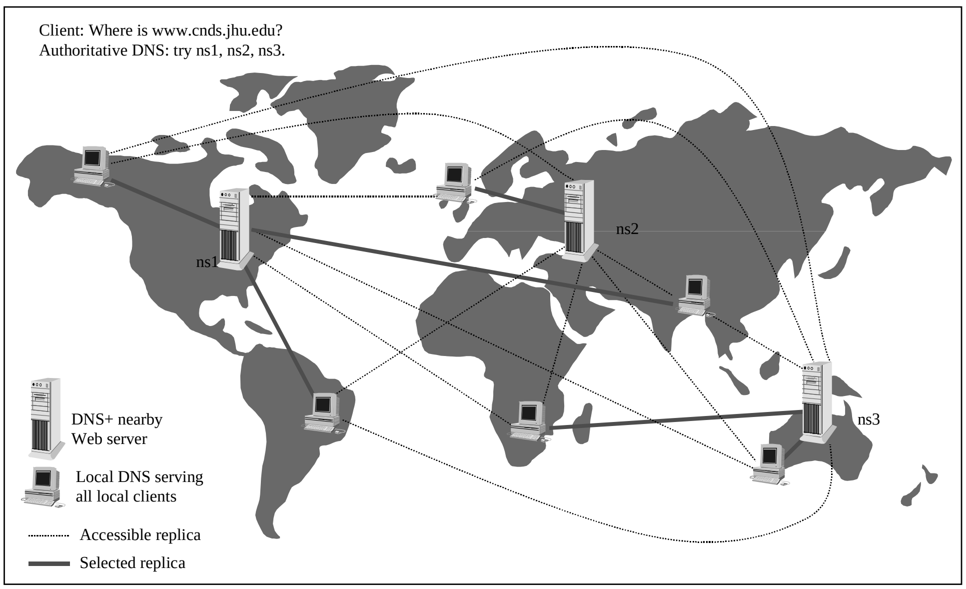

Fig. 2. DNS Round Trip Times method in action

The DNS Round Trip Times method is illustrated in figure 2. The server www.cnds.jhu.edu is replicated in three places, and has an accompanying name server in each place. Clients that want to query this server ask their local name server to resolve the name. The local name server will (eventually) try all three of the possible name servers that are authoritative for www.cnds.jhu.edu. Eventually, the local name server learns which remote name server is closest and concentrates on the web server that is pointed to by this name server.

To help the process along a bit, another concept from DNS can be used. Each name-to-number mapping can have a variable time-to-live (TTL) value. This number specifies a time, after which the address returned should be regarded as invalid, and not used again. By setting a lower than usual TTL on the address record of the web server, we can force the name to be re-requested with greater frequency. Of course, this can be a disadvantage as well, as each request causes extra traffic and occasionally a perceptible delay to the client, as the name is re-resolved.

Using this system, one connection (defined as one or more connections to the web server within a certain TTL) has a *1/n* chance of getting the “correct” server, where *n* is defined as the number of name servers serving that zone. Therefore, given the characteristics of the zone, one can estimate the optimal number of name servers for that zone. For example, a very popular site that gets continual connections from around the world (e.g., Yahoo) could have many more name servers than a site that gets a slower stream of connections. The larger number of connections that a busy site will receive can more quickly get past the disadvantage of being forced to try all of the name servers for a zone before optimizing on the best one. The TTL value may also be “tuned” to give the best results in the particular environment it is used in. Note that this time period is timed relative to the local name server, and not the client or the web server — if several clients that share a local name server all request a document from the same web server within the time period, that still counts as one “connection”.

The shared IP address method

Typically, any given machine on the Internet has a unique IP address. While this is necessary for most things to work properly on the network, it is possible to have multiple machines share an IP address. If this configuration is implemented properly on a network that spreads over a wide geographic area, some very intelligent distribution of network load becomes possible.

If an autonomous network (such as Sprint, MCI, or any other Internet Service Provider) has data centers in San Francisco and Washington DC, for example, the network operator could put servers in each of those locations that have the same IP address. Since all networks nowadays practice what is known as “closest exit routing” this kind of approach can work very well. Closest exit routing is based on the premise that the destination network will know how to find a specific host better than the host, so it should be handed to the destination network as quickly as possible. With external routing protocols, many addresses are aggregated into one announcement to help keep the size of the global routing table down. The destination network that makes those aggregated announcements has the required knowledge to find the specific address that a packet is destined for.

The protocols that carry this more specific information are capable of dealing with multiple instances of the same IP address. The vast majority of these protocols use Dijkstra's Shortest Path First algorithm, which can deal with duplicate addresses. This is done by simply importing the closest instance of the IP address into the router forwarding table. When the destination network gets the packet from the source network, it will send the packet to the closest instance of the shared IP address from its own point of view.

In effect, this creates the illusion that one logical machine is connected in several places of the network via an extremely fast connection. The true situation, of course, is that behind each connection there is a different replica.

Once the servers with the shared IP address are placed throughout a network, one must still implement the higher-level services (HTTP, FTP, etc) on them. An intuitive way to approach the problem is to assign the web server the shared IP address, so that the requests will be sent directly to the "closest" web server. This naïve approach may work in a completely stable network environment, but if the network topology changes during a TCP/IP session, then the client connection with the "closest" web server will immediately break.

We solve this problem by assigning the shared IP address to the DNS servers rather to the web servers we wish to replicate. Each of the DNS servers is configured with the unique IP address of the web server that is "closest" to it. Since DNS requests are stateless, we will not run into the problem stated above, because all stateful communications are done with a unique IP address. As in the DNS Round Trip Times method above, DNS dynamic update can be used to ensure the proper addresses are given to the clients, taking into account the load on the servers and their current availability.

4. Experience and Analysis

We ran a number of tests to confirm the feasibility of all three approaches.

- The HTTP redirect method can easily be implemented using the CGI functionality built into all modern web servers. Some servers can even do this internally without using an external process at all. As previously discussed, this method has no intrinsic ability to incorporate network distance to select the most suitable server. It has, however, the ability to base its decision on the actual request made by the client.

- To verify the DNS Round Trip Times method, we set up seven authoritative name servers around the world. There is one each in Austria, Israel, and Australia, and four in the United States: one in Maryland (at Johns Hopkins), and three in Colorado. Each of these DNS servers responds with a different IP address for the same name of the undergraduate web server on the Department of Computer Science, The Johns Hopkins University, which is a fairly busy server. On the server at Hopkins, we used a special web server configuration using multiple IP addresses, so we could log the address of the client as well as which web server replica it used. In this way we were able to identify the DNS server that was contacted by the user's name server.

- We have found that over the period of several days, clients sharing a remote name server did indeed tend to converge on the name server closest to them. The results presented in Table 1 show, for example, that over a short period of time (approximately 15 days, including the convergence period) 53% of all hits from Australia (as selected by hostnames that end in ".au") went to the IP address given out by the name server in Australia. If the local name server effect (see below) is discounted, we estimate that this figure may rise as high as 80%. Note that this time period includes the convergence period, for which this method is inaccurate. The effectiveness of this method will rise even further over time after the convergence is completed. Therefore, we expect longer tests to show even better behavior.
- This is not an ideal test, as some clients do not have reverse DNS, nor are all Australian clients in a ".au" domain, but it is illustrative of the concept. For a similar reason, it is very difficult to determine the location of clients in the ".com" and ".edu" (and other three-letter) domains.
- We have found that the system convergence time goes up linearly with the number of replicas. However, the law of diminishing returns dictates that the benefit derived from each additional replica decreases. Since the system is not effective until it converges, we feel that using a smaller number of replicas will provide the best overall performance.
- A small change in the way DNS works can remedy the above problem: If the DNS will check the round trip time for all of the remote name servers when it first resolves the name, this method will converge after the first access. This does not seem to be a major change in DNS, and does not seem to create too much extraneous load on the network, but it gives a huge benefit to the current DNS round trip time heuristics.
- One limitation that we discovered is that the name server that delegates to the multiple authoritative name servers should not also be used as a local name server for users. If this is done, that name server would soon learn which of the replicas is the best for its own users, and will return that answer to any remote name server. This distorting effect will last until the TTL of the web server's address expires. This effect explains the skew of the results presented in Table 1 towards the Baltimore server, because the jhu.edu primary name server also serves local clients on the Johns Hopkins campus.

- To test the Shared IP method, we used the Erols Internet network. Erols is an Internet Service Provider servicing all major markets on the East Coast of the US, from Boston MA to Norfolk VA, with about 340,000 users. The Erols network contains about eighty routers running the OSPF routing protocol.
  - We have assigned the same IP address to machines on two different routers: one in New York City, and one in Vienna, Virginia. We have observed that accesses for this shared IP address from New Jersey arrived at the New York City machine. Accesses from Washington DC, in contrast, arrived at the Vienna machine, as expected.
  - We have observed that the network converges to the correct address within about forty seconds after a change to the network, such as one of these machines crashing, occurs. In that case, all of the traffic is directed to the remaining machine after the network converges.
  - It is interesting to note that accessing this IP address from Baltimore, which is, coincidentally, exactly half way between New York's and Vienna's routers according to the network metric, we managed to observe some of the traffic going to the New York machine and some going to the Vienna machine. This confirmed the fact that there will be a problem if the shared IP address is assigned to the web server itself. However, this has no adverse effect when the shared IP address is assigned to the DNS server, thanks to the stateless single-packet nature of most DNS.

Table 1. DNS Round Trip Times Method Results

Location	Accesses from Australia	Accesses from Israel	Accesses from Austria
Baltimore	2192	635	174
Austria	279	233	**88**
Colorado 1	72	24	5
Colorado 2	56	18	2
Colorado 3	78	4	13
Israel	325	**841**	132
Australia	**3439**	236	42

To analyze the relative merits and drawbacks of the three methods, we have identified five important criteria that allow us to compare the different methods:

- *Long-term effectiveness*: How effective and accurate is the method? What factors can be considered?
- *Convergence speed*: How fast can the method converge to the best replica?
- *Setup complexity*: What is required from the web site manager in order to implement such a solution?

- *Extensibility to other services*: How general is this method?
- *Special requirements*: Are there any special requirements?

Table 2 compares the different methods according to the above criteria.

Table 2. Comparing the different location methods.

Property	HTTP redirect	DNS round trip times	Shared IP address
Long- term effectiveness	Not able to consider network distance internally. Can incorporate server load and current availability. The only method that can consider request content in the decision.	Uses round trip times to approximate network distance. Can consider server load and current availability.	Accurately incorporates network distance. Can incorporate server load and current availability.
Convergence speed	Method is in effect only after the first access. Fully effective after that.	Convergence is linear with the number of replicas. Convergence is done per source DNS. All of the remote name servers must be accessed at least once before convergence is achieved. Usually, it is slow (hours to days).	Method is always in effect. Convergence is immediate when there is a network change (about 30 seconds).
Setup complexity	Programming CGI scripts that may re-direct to another replica.	Set up a number of secondary DNS servers that are close to the web server replicas	Set up a shared IP address for the primary DNS servers.

Extensibility to other services	Limited only to the web and HTTP.	The same method works for any name based service (ftp, e-mail, etc.)	The same method works for any name based service (ftp, e-mail, etc.)
Special requirements	None	None	All DNS (and probably all web replicas) have to be on the **same** autonomous network. *i.e.* have the same Internet provider (such as MCI, Sprint, BBN, Erols).

5. Conclusions

We have presented a new approach that uses web replication where each of the replicas resides in a different part of the network. We showed three different methods to seamlessly direct the client to the best replica taking into account the network topology and the server availability and load.

In cases where it is possible to implement the shared IP address method for DNS, we recommend using this method because of the fast convergence time. However, there can be cases where this is not possible, such as when the replicas span different continents and there is no access provider that can route a shared IP address solution (cheaply enough). In these cases, we recommend using the DNS round trip times method with a small number of widely separated replicas.

Preliminary tests show that using either method, our approach may yield significant benefit, both for the client (fast response time, higher availability), and for the network (lower overall traffic).

References

1. NSFNET Backbone Traffic Distribution by Service report. ftp://ftp.merit.edu/nsfnet/statistics/1995/nsf-9503.ports.gz
2. Trigdell, A., Mackerras, P.: The Rsync Algorithm. Technical Report TR-CS-96-05, The Australian National University. Available at ftp://samba.anu.edu.au/pub/rsync/

3. Amir, Y.: Replication Using Group Communication over a Partitioned Network, Ph.D. Thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel (1995). Available at http://www.cs.jhu.edu/~yairamir/
4. Ladin, R., Liskov, B., Shrira, L., Ghemawat, S.: Providing Availability Using Lazy Replication ACM Transactions on Computer Systems, 10(4), pages 360-391.
5. Squid Internet Object Cache. http://squid.nlanr.net/Squid/
6. Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., Worrell, K. J.: A Hierarchical Internet Object Cache. Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, (1995).
7. The Apache HTTP Server Project. http://www.apache.org/
8. Berners-Lee, T., Fielding, R., Frystyk, H.: RFC-1945: Hypertext Transfer Protocol – HTTP/1.0. (1996).
9. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: RFC-2068: Hypertext Transfer Protocol – HTTP/1.1. (1997).
10. On Interpreting Access Statistics. http://www.cranfield.ac.uk/docs/stats/
11. Mockapetris, P.: RFC-1034: Domain Names – Concepts and Facilities. (1987).
12. Mockapetris, P.: RFC-1035: Domain Names – Implementation and Specification. (1987).
13. Vixie, P. (ed), Thompson, S., Rekhter, Y., Bound, J.: RFC-2136: Dynamic Updates in the Domain Name System (DNS UPDATE). (1997).