A Distributed Caching Web Proxy Framework

Ovais A. Khan, Raza Ali
National University of Computer and Emerging Sciences
ST-4, Sector 17/D, Shah Latif Town, Karachi 75030, Pakistan
{ovais.khan, ali.raza} @ nu.edu.pk

Abstract: The growth of HTTP traffic and experience with use of proxy caches has demonstrated the need for increasing the scalability and availability of proxy cache software. To increase scalability, proxy caching software needs to be written in a modular, decentralized fashion. To make it more available, the service needs to be transparent to the client. There are two approaches available for proxy server implementation namely, centralized and distributed. In a centralized cache server, the main bottleneck is a central resource. On the other hand, in a distributed cache, it is the communication between the servers. We have selected the distributed approach to provide efficient request redirection that result in better load distribution and transparency of service. Our system is a distributed caching web proxy framework. It can accommodate any proxy server that supports extension to its basic operation such as open source proxy servers. Our emphasis is on transparency, availability, scalability and load distribution. In addition, we intend our system to work for high load environments such as university departments or large companies. Results will show clear performance difference in terms of scalability and availability between the two approaches.

INTRODUCTION

> "One of the overall design goals is to create a computing system which is capable of meeting almost all of the requirements of a large computer utility. Such systems must run continuously and reliably 7 days a week, 24 hours a day... and must be capable of meeting wide service demands.
>
> "Because the system must ultimately be comprehensive and able to adapt to unknown future requirements, its framework must be general, and capable of evolving over time." [1]

Our emphasis from design point of view is on scalability, availability and transparency factors. Additional advantages that we have achieved during our work are in terms of load distribution and cost effectiveness.

- By *scalability*, we mean that when the load offered to the service increases, an incremental and linear increase in hardware can maintain the same per-user level of service.
- By *availability*, we mean that the service as a whole must be available 24x7, despite transient partial hardware or software failures.
- By *transparency*, we mean that the way the request is serviced is transparent to the user as are its failures.
- By *load distribution*, we mean that the resources of our distributed system are utilized properly.
- By *cost effectiveness*, we mean that the service must be economical to administer and expand, even though it potentially comprises many workstation nodes.

The growth of the Internet and the World Wide Web have enabled increasing numbers of users to access vast amounts of information stored at geographically distributed sites. Due to the non-uniformity of information access, however, popular web pages create "hot spots" of network and server load, and thereby significantly increase the latency for information access.

Large-scale distributed caches appear to provide an opportunity to combat this latency [9] because they allow users to benefit from data fetched by other users, and their distributed architectures allow clients to access nearby copies of data in the common case. Current web cache systems define a hierarchy of data caches and follow a horizontal and vertical mechanism of retreiving a requested object [8]. It is mentioned in [5] that these hierarchies of data caches often achieve modest hit rates, can yield poor response times on a cache hit [10, 11], and can slow down cache misses.

In this paper, we propose a distributed caching web proxy framework, a combination of various existing techniques for web proxy and scalable network services. The framework harnesses the unused power of client machines and shifts the overhead of request distribution to client side with the help of a client proxy. The servers maintain a group arrangement among which a leader or coordinator is selected. The coordinator is responsible for maintaining a list of available servers and their load statistics. The client proxies communicate with the coordinator for various information. Server failures including that of the coordinator is handled using a popular election algorithm.

This framework may evolve as a general framework for scalable network services.

RELATED WORK

Existing cooperative proxy systems can be organized in hierarchical and distributed manners [6]. The hierarchical approach is based on the Internet Caching Protocol (ICP) [25] with a fixed hierarchy. A page not in the local cache of a proxy server is first requested from neighboring proxies on the same hierarchy level. Root proxy in the hierarchy will be queried if requests are not resolved locally and they continue to climb the hierarchy until the request objects are found. This often leads to a bottleneck situation at the main root server. The distributed approach is usually based on a hashing algorithm like the Cache Array Routing Protocol (CARP) [9].

Hash Routing

Hash Routing are deterministic hash-based approaches for mapping an object to a unique sibling cache [21, 22, 23]. This technique works at Application Level and distributes the URL space among the sibling caches, creating a single logical cache spread over many caches. Because an object is cached at only one caching node, hash routing can achieve a higher hit rate. On the other hand, it has few drawbacks. One is its large network traffic between caching nodes. Because a client cannot retrieve objects directly from the cache of its local caching node in most cases, a large number of object transmissions between caching nodes occur in the hash routing system. The other drawback is its lack of fault tolerance. Because an object is only cached at one node, all clients can suffer from errors or cache misses even when just a single caching node in the system suffers failure.

Load balancing at the client side using proxy servers was explored by Wu and Yu [12]. They emphasized on tuning the commonly used hashing algorithm for load distribution. Researchers from MIT, on the other hand, have proposed a new hashing algorithm called consistent hashing to improve caching performance when resources are added from time to time [13].

Super Proxy Script [11] is a client side application level protocol which is a minor variant of Hash Routing. It uses the Proxy Auto Configuration (PAC) script with timeout / failover to provide a robust, scalable and deterministic proxy caches. When PAC is used, a client receives a list of proxy servers. If the local proxy is in failure, the client can bypass it and forward its requests to one of other caching proxies according to the proxy list. The major point of failure is the machine carrying the PAC script. Apart from that, most web browsers need to be restarted to have any change in the configuration take effect.

Round Robin DNS

In this scheme, when DNS server receives name resolution queries, it returns the IP address of one of the replicated servers in a round-robin fashion (RR-DNS). A primary example of this scheme is the NCSA web site [14]. There, however, are several drawbacks to DNS-based solutions. A fundamental problem is caching of IP addresses at client machines and local DNS servers which makes the load less than perfectly balanced among the servers. Another problem is that DNS servers know nothing about server status or network topology, and then selected server may be a distant, overloaded, or even unavailable server.

Alternate DNS

A simple variation of RR-DNS is Alternate Domain Name Server (ADNS) [20]. It acts as a normal DNS server but implements laod balancing on current load on servers. ADNS finds the IP address and port number that the client should use to retrieve web pages from the internet. The system is robust enough to recover from failure of the client, the web proxies, and ADNS. The proxies automatically restart the ADNS when it fails.

Socket-level Redirection Mechanism

Socket-level redirection mechanism is a transparent mechanism to access replicated servers. Its main idea is to redirect a client request to a replicated server in the socket function of the client [15]. This mechanism requires the modification of the socket library of the client and do not include the overhead of packet rewriting at all. But, making modifications to the socket library of the client is not a simple task.

Magic Router

NOW project at Berkley has developed the MagicRouter [16], which is a packet-filter-based approach [17] to distributing network packets in a cluster. The Magic Router acts as a switchboard that distributes requests for Service to the individual nodes in a cluster. It requires packets from a client be forwarded (or rewritten) by the MagicRouter to the individual server chosen to service the request. Also it requires the packet from the server to be rewritten by the MagicRouter on its way back.

TCP Router

An architecture slightly different from MagicRouter is TCP Router [18], in which a 'TCP Router' acts as a front end that forwards requests for the service to the individual back-end servers of the cluster. TCP Router eliminates the need to rewrite the packet going from the server to the client and secondly, it assigns connection to the servers based on the state of these servers.

Distributed Packet Rewriting

Distributed Packet Rewriting (DPR) [19] is a distributed approach in which all hosts of the distributed system participate in connection routing. Definitely, this approach provides better fault tolerance and scalability. But the packet level modification and session management take too much of the effort. Scalability and load distribution are major problems.

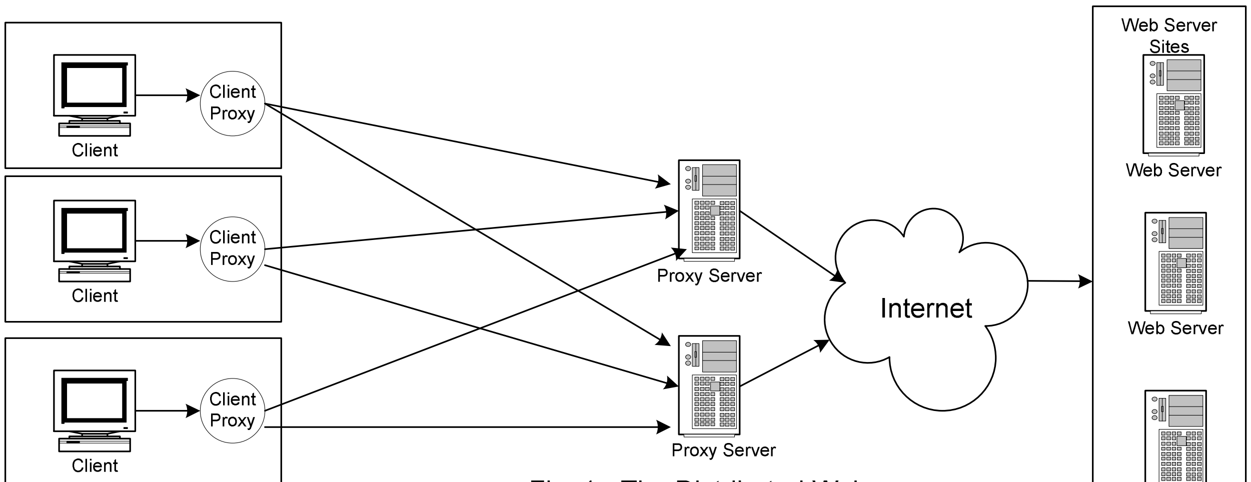


Fig. 1 - The Distributed Web Proxy Framework

PROPOSED FRAMEWORK

Overview

The framework which we have implemented breaks the proxy service into two pieces of software the *client proxy* and the *proxy server*. The client proxy is a fully functional but small scale and non-caching web proxy server and runs on the client machine. The client web browser is configured to client proxy, so all requests reach the client proxy first and it is responsible for getting them serviced. See Figure 1.

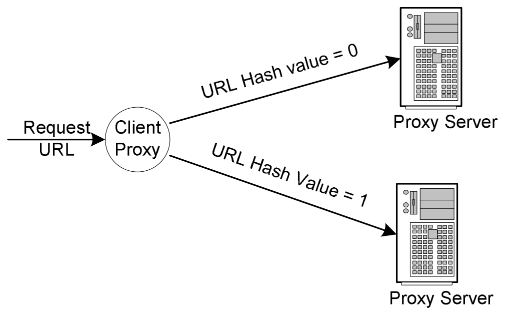
The client uses a discovery procedure to find out all available proxy servers and uses hashing and a feedback mechanism to transparently service all requests from different proxy servers. The following sections provide the details of all these procedures.

Client proxy

The client proxy software is a fully functional HTTP 1.0 proxy server. Caching is not implemented in the client as the purpose of this proxy server is to distribute the requests and isolate the client software from the infrastructure and load changes taking place at the actual proxy servers. We have currently used Java to write this proxy server as it makes it independent of the client platform.

The isolation provided by the client proxy effectively solves the transparency problem mentioned earlier. The client software now needs to know about only one proxy server which is locally present. Client proxy runs a Discovery procedure on-demand, when a certain percentage of the requests fail or after the expiry of a certain time interval. This way the client proxy can adapt to the dynamic state of the proxy servers.

Upon receiving a web request the client proxy uses the Hashing [11] technique to map this request onto one of the proxy servers. The hash based distribution helps localize the request service and not only increases the cache hit-rate but also reduces the need to keep duplicate copies of the same document on different proxy serves.



Request distribution using hashing

Another advantage that this scheme has achieved is the distribution of workload among the client machines and proxy servers. The client proxy has made the client machine essentially a part of our web proxy service. Instead if we concentrated all the distribution and service functionality into the proxy servers either this would have introduced a central resource such as DNS or router or the working resource demands and distribution overhead on the proxy servers would have greatly increased. Client machines often have a lot of unused computing power available at their disposal and such cooperative schemes can make things a lot less complicated at both client and server ends.

Proxy Server

The proxy servers are logically organized into two types, coordinator or monitor proxy servers and non-coordinator proxy servers. Currently we are using only one proxy server per group. The selection for the coordinator is done with the help of the Election algorithm [21].

The selected coordinator is responsible for maintaining a list of active proxy servers which is provided to the client proxy during the discovery process. Also it keeps the load statistics including the number of current requests and average service time for all the proxy servers. This information is provided to the client proxy during the periodic discovery cycle so that requests remain evenly distributed and no proxy server is overloaded during a hot-spot.

Electing the Coordinator

This election algorithm is based on the classical Bully Algorithm [21] except that the priority of a given node depends on its IP address. The lower the IP address, the higher the priority will be and vice versa.

Election Algorithm

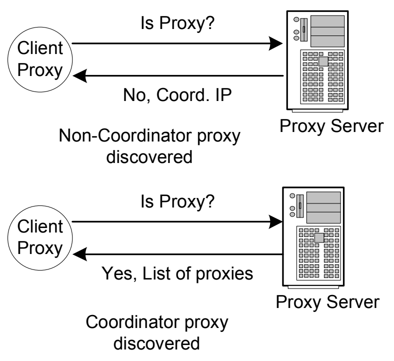
The implementation of the Election algorithm is straight forward. The communication takes place through TCP and UDP port 54321. In order to check which of the servers are up, we do a multicast on this port and the servers which reply are up.

When a new node joins the network, it sends a multicast message to a given IP range on the network. If the coordinator is up, it will reply. If no reply is received, the election procedure is started. The election procedure will start when the election candidate will broadcast the election message. Then it will look for all higher priority nodes which are up. If a reply is received, it stops and waits for the other node to start election and send it the result. Once no reply is received, it will notify all other members about its coordinator status.

Discovery algorithm

The discovery process is very simple in nature. The client proxy is provided in its configuration a range of IP addresses where at least one proxy server either coordinator or non-coordinator is present. The client proxy initiates the discovery process by scanning the IP range from start address and searches to the end address. For each address, it sends out a message “Is Proxy” in simple string format to that machine on a pre-defined port using a TCP connection. If the connection fails or garbage is returned in response then this machine is skipped. If the machine responds with a “No” then this is a non-coordinator proxy server and following this message would be the IP address of the coordinator machine. If the reply is a “Yes” then we have discovered the coordinator of the group and following this message would be the list of IP addresses of all the proxy servers in the group.

This procedure provides a direct shortcut to the coordinator proxy and avoids unnecessary communication. The client after discovering the coordinator remembers its address for later discovery cycles and builds a table of proxy server addresses over which it will map the requests. In case the coordinator proxy goes down, the process will start over to discover the new coordinator of the group.



Similarly the client proxy can send the message “Load Stats” to the coordinator proxy to get the current statistics and adapt its distribution pattern to avoid overloading any proxy server.

Load Distribution Mechanism

Distribution of load is one of the core issues when providing graceful degradation. As mentioned with respect to the client proxy that this objective is achieved through the hash-based routing technique. But a downside is that when a machine goes down all mapped requests will fail. The group would reorganize itself and the client proxies would discover this in the next discovery cycle. The client proxy would then adjust it’s modulo function to map all requests to the available machines.

The change in modulo can cause a surge of download because of cache losses. To counter this we propose a two stage modulo, lets say the old and the new. The modulo is calculated on the old one first to get the right mapping, for as many proxy servers as possible, before the lost server in the proxy list.

Proxy scaling is similarly controlled. When a new machine comes up and becomes a member of the group its IP is added to the end of the list by the coordinator. After the next discovery cycle this machine is discovered and the client proxies set both their old and new modulo functions to the group’s new size. This again will cause a disturbance in the pattern of cache access but since the proxy group is scaling it will be able to handle the load. Also due to this pattern change the new member will receive a lot of requests and would quickly become an active servicing member.

One problem the hash based routing can run into is what may be called hot-spots. Hot-Spots can occur when the user request suddenly get focused on a set of websites, lets say newspaper sites after 9/11. Due to this it may happen that some servers receive very few requests while others are getting overloaded. Solution to this problem is provided through the load statistics management at the coordinator. During each discovery cycle the client proxies not only ask for the list of servers but also their current load status. If the server mapped onto a request is currently serving a higher number of requests than a pre-defined number then this request is sent randomly to one of the least loaded servers. The randomness is kept to counter a sudden shift of load towards the less loaded servers.

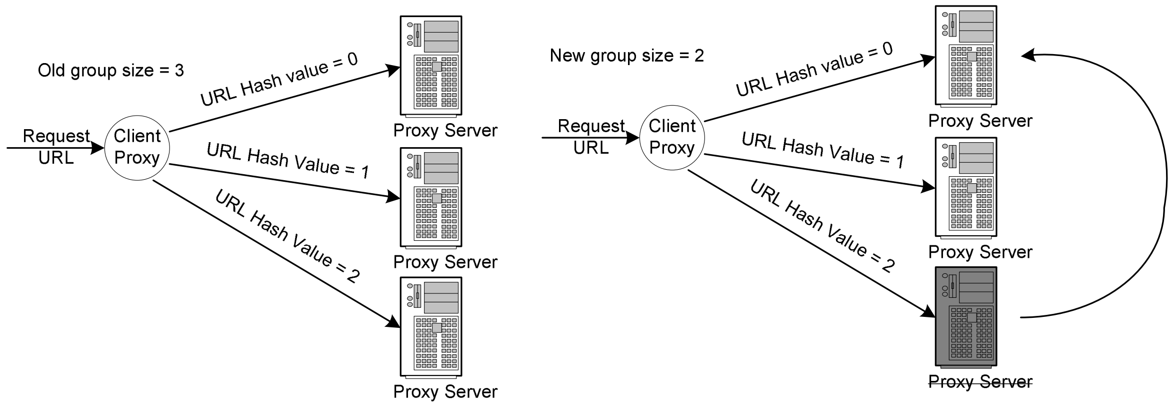
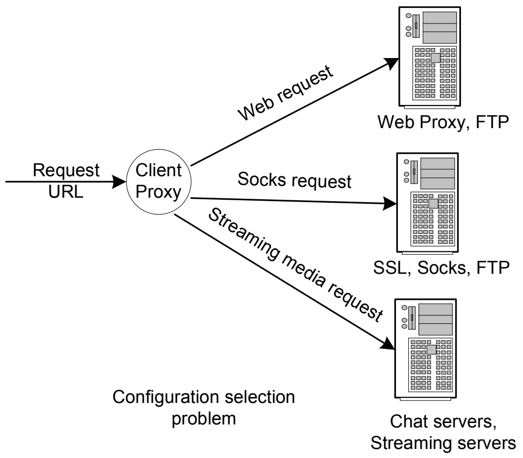


Fig. 2 - Hash routing and two stage hashing

Configuration Selection Problem

We have not implemented this feature but it is useful to have it in the framework and which we might implement in future. The configuration selection problem deals with mapping the requests of a user belonging to a particular user group onto a set of proxy servers with a specified configuration. For example two such groups in a university environment may be faculty and students. Both may have different requirements hence their requests sent to different machines.



Another implication of the configuration selection is that all machines do not have to run all kinds of services. For example some machines are running proxy servers dedicated to normal web requests, while others may support SSL and another group of proxy servers may allow FTP or gopher. This allows the proxy servers to run smaller, faster and more purpose oriented software while the job of sending the request to the correct machine is shifted to client end.

This kind of separate machine configuration would allow system administrators to easily maintain machines for serving different user group reducing the complexity of managing all configurations at one place and avoiding potential security risks.

Administration

Administration is a serious concern for systems of many nodes. We started from [27], which describes how a unified monitoring/reporting framework with data visualization support was an effective tool for simplifying cluster administration. Our framework has the ability to incorporate such a monitoring and administrative module, as the framework already has the ability to locate the active nodes. The only need is to define the format of the configuration and log files and implementation of a quorum based algorithm [28] for keeping the configuration consistent.

Cache Sharing

There has been a lot of research on the topic of proxy caching and cache sharing. Four basic design principles for large-scale caches are: (1) minimize the number of hops to locate and access data, (2) do not slow down misses, (3) share data among many caches, and (4) cache data close to clients. Although these principles may seem obvious in retrospect, we find that current cache architectures routinely violate them at a significant performance cost [5].

Web caching systems tend to be composed of multiple, distributed caches to improve system scalability, availability, or to leverage physical locality. In terms of scalability and availability, the existence of multiple, distributed caches permits a system to deal with a high degree of concurrent client requests as well as survive the failure of some caches during normal operation. In terms of physical locality, assuming that bandwidth is constant, simply having caches closer in proximity to certain groups of users may be an effective way to reduce average network latencies, since there is often a correlation between the location of a user and the objects requested [22].

There are five well-known protocols [23] for inter-cache communication: ICP, cache digests, CRP, CARP, and WCCP. ICP evolved from the Harvest project and was explored in more detail within Squid. It has the longest history and is the most mature. ICP uses simple queries to locate the best possible location of the requested object. One issue is of desirable limits to the depth of the cache hierarchy [24]. Another scalability concern was the number of ICP messages that could be generated as the number of cache peers increased [13]. CRP protocol uses multicast to query cache meshes. To optimize the path of meshes to query, adaptive caching uses CRP to determine the likely direction of origin for the content.

Cache digests is another technique which is used for this purpose, such as those implemented by Squid [13] and the Summary Cache [26]. Cisco's WCCP and Microsoft's CARP [9] method are two of the proprietary protocols.

We have used hashing on the client side to locate the probable server containing the object directly. If the object is not located, proxy server has been configured to issue ICP multicast query. These two techniques have been used in order to increase the scalability and availability while still maintaining high hit rates.

When a client issues a request for an object and all the servers are up, the client proxy hashes and sends the requests to the server responsible for fetching this object. If a particular proxy is down, the requests bound for this particular proxy will be distributed to other proxies. Definitely, during the time the proxy is down, other proxies will cache its objects. After the proxy recovers, ICP comes into the picture to increase the hits for those objects serviced by other proxies during its down time. Thus combining the two schemes produces better hit rates even when some portion of the network is down.

PROPOSED EXPERIMENTAL SETUP

Introduction

In order to test the proposed framework if it provides the objectives of an ideal proxy environment, we plan to test the reference implementation first on its ability to load balance and cluster in reaction to experienced hot-spot situations.

We also need to test the ability of the framework to adapt to various changes in the underlying infrastructure. The gained results will show how such distributed proxy system adapts to the request pattern in such a way that load balancing and data clustering will emerge. The performance of any proxy architectures are highly dependent on the pattern of the requests, but our proposed framework is able to handle local hot spots to a good degree.

Architecture

The infrastructure proposed for later simulations is similar to an institutional proxy environment. There are n clients and m proxy servers servicing their requests as shown in Figure 3. The final requested destinations will be from amongst the k web servers that we have setup on the Local Area Network. We assume that in this context, going to any proxy within the array of proxies, even by doing a maximum number of hops, is always faster than requesting the data from the origin server. A local hit will always be assumed faster than a remote request. We further assume full knowledge and full connectivity within the proxy layer.

In place of a normal web browser, we have used Scalable URL Reference Generator (SURGE) to generate references which match empirical measurements of various web traffic pattern.

Request Pattern

The request distribution of SURGE has been shown to follows the Zipf's law for popularity at real Web servers [29]. Resent research has shown that a power-law or

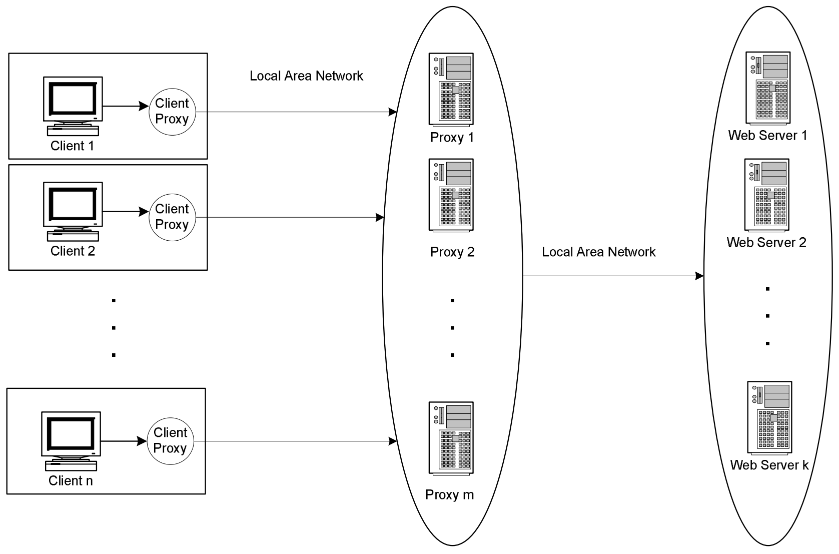


Fig. 3 - Proposed Experimental Setup

ZIPF-law distribution is very suitable to describe the experienced request pattern on a proxy server [30]. In our simulation, clients inject requests for all existing servers based on a ZIPF-distribution. Future work with a real system shall broaden these limitations.

Preliminary simulations have shown that the outcome of the simulations is highly dependent on the request pattern.

Performance Comparison Tests

We plan to setup four squid proxies and four clients. A set of five web servers on the Local Area Network will be used as the destination, for all these tests.

Standalone (Hierarchical) Proxy Setup: In the standalone setup, all the squid proxies are independent and they are not sharing any information and cache.

ICP Enabled Proxy Setup: In the second test environment, we enable the ICP in all the squid proxies. They can now query their peer proxies for any object before asking the web server.

Super Proxy: This technique is closest to our framework. We simply setup a PAC file on all the servers and configure the browsers to automatically setup using this PAC file.

Changing infrastructure

In order to test the performance of the framework, under changing conditions we will simulate an increase in requests. In the first half, we will keep on removing proxies, with constant load. While in the second half of the test, we will increase the load first to burden the current servers and then introduce new proxies.

EXPERIMENTAL RESULTS

The experiments are currently in progress and no results can be presented at this stage.

CONCLUSIONS

We presented a distributed network service framework , which was illustrated with its implementation as a scalable web proxy framework. Distributed services are the need of the time and we have demonstrated with our work that existing services can be improved greatly by switching to distributed and cooperative solutions.

FUTURE DIRECTIONS

The next step in improvements is to implement a integerated security architecture. Providing access rights on services and objects.

This framework may be adapted to provide other services such as the currently popular streaming media services. It can evolve as a general framework for scalable network services.

This framework gives us the ability to involve the client in the decision making for service request. We can look for furthur ways to harness the clients unused power to simplify the server end software and take it towards truly distributed system.

REFERENCES

[1] F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System", in *AFIPS Conference Proceedings,* 27, pp. 185-196, 1965.

[2] C. Maltzahn, K. Richardson, and D. Grunwald. "Performance Issues of Enterprise Level Web Proxies", in *Proceedings of the SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 1997.

[3] A. Rousskov. "On Performance of Caching Proxies", http://www.cs.ndsu.nodak.edu/rousskov/research/cache/squid/profiling/papers, 1996.

[4] D. Wessels and K. Claffy. "RFC 2186: Internet Cache Protocol (ICP), Version2", September 1997.

[5] R. Tewari, M. Dahlin, H. M. Vin, and J. B. Kay, "Beyond Hierchies: Design Considerations for Distributed Caching on the Internet", UTCS Technical Report: TR98-04

[6] Povey, D., Harrison, J., "A Distributed Internet Cache",

[7] Dykes, S. G., C. L. Jeffery, and S. Das, `Taxonomy and Design for Distributed Web Caching", in *Proceedings of the Hawaii International Conference on System Science*, 1999.

[8] Rodriguez, P., C. Spanner, and E. W. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching", in: *Proceedings of the Fourth International WWW Caching Workshop*, 1999.

[9] Cohen, J., N. Phadnis, V. Valloppillil, and K. W. Ross, "Cache Array Routing Protocol V.1.1", 1997.

[10] KeithW. Ross, "Hash-Routing for Collections of Shared Web Caches", *IEEE Network*, pp. 37–44, November/December 1997.

[11] K. Doi, "Super Proxy Script: How to make distributed proxy servers by URL hashing", White Paper: http://naragw.sharp.co.jp/sps/, August 1996.

[12] Wu, K. and P. Yu, "Load Balancing and Hot Spot Relief for Hash Routing Among a Collection of Proxy Caches", in *Proceedings of the 19th International Conference on Distributed Computing Systems*, 1999.

[13] Karger, D., T. Leighton, D. Lewin, and A. Sherman, "Web Caching with Consistent Hashing" in *Proceedings of the WWW8 Conference*, 1999.

[14] E. D. Katz, M. Butler, and R. McGrath, "A scalable HTTP server: The NCSA prototype," *Computer Networks and ISDN Systems,* vol. 27, pp. 155-164, 1994.

[15] Han, Jaesun, "A Socket-Level Redirection Mechanism for Providing Internet Scalability", *6th Samsung Humantech Thesis,* http://www.samsung.com/AboutSAMSUNG/SocialCommitment/HumantechThesis/WinningPaper6.htm

[16] Eric Anderson, David Patterson, and Eric Brewer, "The MagicRouter: An application of fast packet interposing", Submitted to OSDI 1996, May 1996

[17] Jefferey Mogul, Richard Rashid, and Micheal Accetta. "The Packet Filter: An Efficient Mechanism for User-leve; Network Code", in *proceedings of SOSP'87: The 11th ACM Symposium on Operating Systems Principles*, 1987.

[18] Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari. "A scalable and highly available web server", in *Proceedings of IEEE COMPCON'96*, pp. 85-92, 1996.

[19] A. Bestavros, M. Crovella, J. Liu, and D. Martin, "Distributed Packet Rewriting and its Application to Scalable Web Server Architectures," in *Proceedings of 6th IEEE International Conference on Network Protocols*, IEEE Computer Society Press, Los Alamitos, California, pp. 290-297, 1998.

[20] Long Le and Dorian Miller, "Load Balancing Web Proxy Service", http://www.cs.unc.edu/~dorianm/academics/comp243/loadproxy.pdf, December 2000

[21] H. Garcia-Molina. "Elections in a Distributed Computing System", *IEEE Trans. on Computers*, 31, pp. 48--59, January 1982

[22] Greg Barish and Katia Obraczka, "World Wide Web Caching: Trends and Techniques"

[23] I. Melve, "Inter-cache communication protocols", *IETF WREC Working Group Draft*, 1999.

[24] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. "World wide web caching - the application level view of the internet", *IEEE Communications Magazine vol. 35* June 1997.

[25] K. Claffy and D. Wessels. "ICP and the Squid Web Cache", 1997.

[26] L. Fan, P. Cao, J. Almeida, and A.Z. Border. "Summary cache: A scalable wide-area web cache sharing protocol", Computer Sciences Department University of WisconsinMadison, 1998.

[27] E. Anderson and David A. Patterson, "Extensible, Scalable Monitoring For Clusters of Computers", in *Proc. Large Installation System Administration Confere (LISA XI)*, 1997.

[28] D. K. Gifford, "Weighted Voting for Replicate Data", in *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, pp 150-159, 1979

[29] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", in *Proceedings of the 1998 ACP SIGMETRICS Intl. Conference on Measurment and Modeling of Computer Systems*, pp. 151-160, July 1998

[30] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", Technical Report 1371, Computer Sciences Dept, Univ. of Wisconsin-Madison, April 1998