A Scalable and Distributed WWW Proxy System

K.L.E. Law, B. Nandy, A. Chapman
Nortel Limited, Ottawa

Abstract

With the wide-spread popularity of the World-Wide Web (WWW), network and server load increases dramatically. Caching proxies have been introduced to improve the system performance with the assumption that a page will be fetched many times before it is destroyed or modified. However, the increase on the user requests has driven up to the limit of the proxy performance. In this paper, we present a distributed proxy server architecture that can increase the service availability, provide system scaleability coupled with load balancing capability. The system employs TCP-based switching mechanism which has a finer session granularity and more dynamic control on resource allocation.

1. Introduction

The World-Wide Web (WWW) provides a platform that enables fast deployment of multimedia applications, it also demonstrates the value of wide-area information sharing. However, the dramatic increase in networked information access through the WWW has utilized the underlying infrastructure to its limits. Recent studies show that Web consumes more Internet bandwidth than any other applications [1]. The WWW faces many challenges as it continues to scale to accommodate the ever increasing number of users. Among those are increased server and network load, network latency and inadequate security. Proxy servers are used to reduce the network load and latency by migrating the work load close to the clients. Therefore, a good caching policy and server system design for the proxy are required.

A proxy server is an application gateway which operates at the HyperText Transfer Protocol (HTTP) [2] layer. The basic function of a proxy server is almost identical to a HTTP server in transferring client requested documents. Furthermore, it is able to interpret and modify a HTTP request before forwarding it to the end WWW server. This is because the proxy server has the caching function. If there is a cache hit, it delivers the found documents to the client locally, thus reducing the network load and transmission latency. Apart form local caching, a proxy server can also act as a firewall machine, in this regard, the response time may not be as important as the security issue.

The caching performance of a proxy has serious impact on the overall network performance, especially, since there are only a few point of access from a corporate network to the Internet. A bad proxy architecture can slow down the operations of an intranet and slow down the flow of networked information. Although the caching strategy plays an important role in the system performance, in this paper, we focus on the proxy system design.

There are several distributed architectures [3] used for improving the proxy server performance. For example, there is a hierarchical caching system which arranges proxies in different levels When a document is not found in a local cache, the local proxy contacts the regional proxy to query about the requested document. If it is also not found, the regional proxy will contact a supercache. In this hierarchical scheme, the proxy servers on the next higher level should be more powerful because they serve more people and have more hits than the local caches. Hence we need to strategically place these regional caches with larger storage capacity and high-speed access links. There are several drawbacks with this architecture. If a document is not in any cache, this multiple redirection will waste unnecessary network bandwidth and the precious time to do all the proxy communications. The caches in the hierarchy do not collaborate, and the hierarchy itself is constructed *ad hoc* by the caching server administrators. This configuration is error-prone and difficult to keep up to date, and processing the rules can load down the server.

Cache coherence is also a problem in the hierarchical proxy architecture. Different cache servers may each have a copy, and possibly different copies, of the same original document, possibly with different expiration dates. The information provider has no way to guarantee that the users will get the current version, nor even any way to know where copies are cached or what version a user might be receiving. Thus, the problems of maintaining cache consistency are greater in the hierarchy, mainly

because the Web servers and caches do not communicate with each other to keep the information up to date.

The Harvest system is an enhanced proposal which treats the physically hierarchically arranged caching servers as a flatly distributed cache. The caches communicate with each other and transfer data using the Internet Caching Protocol (ICP) [4]. The ICP use User Datagram Protocol (UDP) as transport for greater efficiency over the network. A local cache will query the neighboring cache to see if there exists the needed document. If so, the Harvest cache retrieves the document from the neighbor and relays it to the client. This configuration would be useful for an organization with several networks connected by a backbone network. Each local network could have a cache, and all the caches could share with each other. It should be much faster to retrieve a document from another cache on the same backbone than from the originating Web server, and also reduces the traffic on the Internet.

Though the Harvest system can increase the cache hit rate, it does not relieve the peak load requests. When there are too many client requests, the Harvest system, similar to other WWW servers, is unable to handle any incoming client requests. Hence in this paper, we present a new WWW proxy architecture that can achieve the following goals:

1. scalable proxy server arrangements,
2. high availability of information service, and
3. dynamic load balancing of traffic loads to different proxy servers.

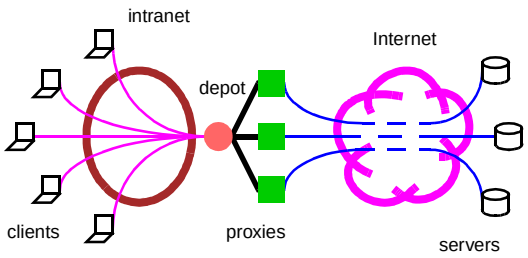
In the paper, we propose to use an intermediary device which we call it a “depot”. This device works only up to the Transmission Control Protocol (TCP) layer and switches each TCP session between a selected proxy server and a client. In Section 2, we discuss this proposed depot proxy system. The functionality of the depot is further investigated in Section 3. Some interesting related issues will be discussed in Section 4. The performance of our testbed is presented in Section 5. Finally, conclusion and future works will be discussed in Section 6.

2. Depot Proxy Model

As shown in Figure 1, the proposed depot proxy system locates between an intranet and the Internet. A firewall is usually connected between the proxies and the Internet to protect the internal resources and information. In this project, we focus on the design of an intermediary device which sits between the proxies and the clients. This internetworking unit is called a “depot”.

In the configuration of the depot proxy system, we establish two levels of load management. Since we can provide many depot proxy systems, the first level is from an internal client to a depot, the second one is from the depot to a proxy. Given an alias site name of a depot system, a client contacts an intelligent DNS server [5] and is given back an IP address of the depot based on locality or other factors. The second level of load balancing is via the depot which dynamically distributes transactions among a group of proxy servers. All proxies are able to terminate the HTTP requests. Hence, two load balancing algorithms can be provided in the model, one is in the DNS server and another one is in the depot. Each local client should go to a local depot, this usually gives faster response time and reduces unnecessary traffic in the networks.

FIGURE 1. A Simplifed Version of a Depot Proxy System.



Load balancing is a common network optimization problem, but we can select the coefficients and parameters in a weighted cost function on an *ad hoc* basis. The selected metrics for the load balancing equations should identify the bottlenecks of the system. Possible indices for choosing proxy could be the available cache size, disk bandwidth, response time from depot to proxy, the proxy server CPU loading and the number of active and pending requests [6].

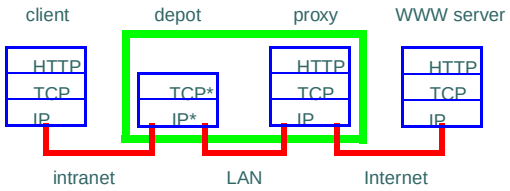
To achieve scaleability, we can easily attach one more proxy to the depot. This is because the depot handles only simple functions. When a depot operates to its limitations, we can add a new depot with a new IP address under the same alias name. With the newly added depot, we create another cluster of proxy servers on the network.

3. Functions of the Depot

The HTTP is a stateless protocol which helps reduce the complexity of server implementation in order to improve the processing speed. Therefore, a HTTP server responds to a client request without the need to relate to any previous or subsequent requests. The choice of a physical server itself becomes immaterial to the transactions. Based on this principle, we are able to switch different TCP sessions to different proxy servers. We call this mechanism TCP-based switching.

The depot uses this TCP-based switching mechanism. It works at the TCP layer but does not terminate any TCP sessions. It examines all incoming packets sent between the clients and proxy servers. Figure 2 shows the protocol stack structure of the system model. On the depot, we do not need to fully implement the complete TCP and IP protocols, since the depot is merely a packet forwarding device with minimal overhead that can help increase the service rate.

FIGURE 2. The Protocol Stack Model of the Depot Proxy Architecture.



HTTP: HyperText Transfer Protocol
TCP: Transmission Control Protocol
IP: Internet Protocol
*: not a complete implementation

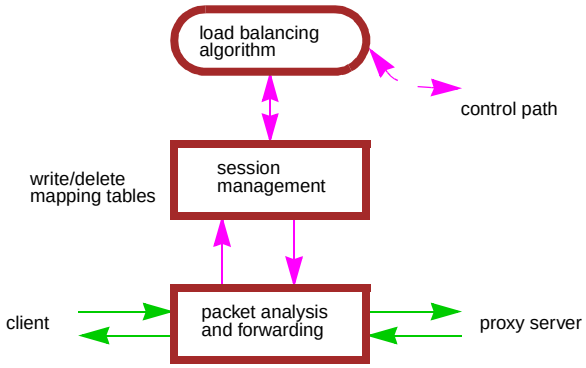
Specific functions of a depot are to:

1. inspect all packets in both directions at IP and TCP levels,
2. forward packets of existing sessions to the correct proxy server,
3. choose a proxy server and begin packet forwarding for a new session,
4. clean up when sessions end, and
5. watch for and handle anomalous states and packets.

In Figure 3, we show the functional block diagram of a proposed depot. For a new session, the packet analyzer identifies the TCP connection setup request and forwards the information to the session management block. This block is pre-loaded with the next proxy to be allocated. This pre-load is done by running a load balancing algorithm on the basis of knowledge of the server statistics and perhaps network states. Currently, we use the simple round-robin scheduling discipline.

If the session is already allocated and recognized by the packet analyzer, it will read in the correct proxy IP address from its tables and forward the packet onward. Packets in the reverse direction are also analyzed in order to have a complete view of the state of the session.

FIGURE 3. The Functional Block Diagram of the Depot.



4. Related Issues

4.1. TCP State Tracking Issue

A TCP session is state-ful within each session between a client and a server. Employing TCP-based switching mechanism, the depot should handle all arriving TCP/IP packets as transparently as possible; otherwise, the introduction of an intermediary may disrupt the normal TCP logic between the two end systems.

In the TCP state transition diagram, there are eleven states [7]. For the client-server model, the number of possible states is reduced, for example, SYN_SENT state will never occur in the Web server. This reduces the state tracking requirement in the depot. However, out-of-sequence situations can occur in the networks, different permutations of state transitions can arise which slightly increase the complexity of the state tracking routine. Therefore, any packet received by the depot does not guarantee the reception of the packet by the other end subsequently. As a result, the state tracking function of the depot simply guesses the current TCP states at each end by investigating the “flag” information of all arriving TCP packets.

The crucial function of the state tracking routine in the depot involves the creation and deletion of the table entries. Several types of TCP/IP packets can initiate the creation and deletion of table entries. Whenever an SYN packet arrives at the depot from client without matching any table entry, the depot will create one entry for the connection and forward it to the server. Since TCP is an asymmetric closing protocol, we should handle the closing of a connection in depot with caution. There exists a TIME_WAIT state at the server even though the TCP connection at the client side is removed. This TIME_WAIT will stay for 2 MSL (maximum segment lifetime)[1] to handle those wandering packets in the Internet. As a result, a

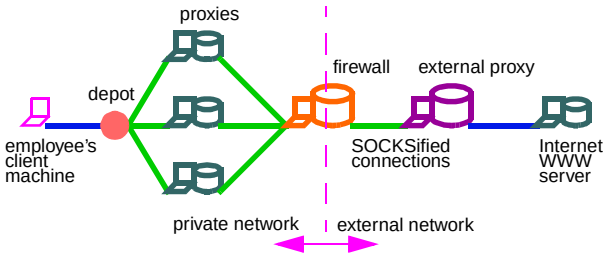
table entry should also remain in the depot to handle those estranged packets for 2 MSL time.

The state tracking routine also handles several other operations, for example, proper update of acknowledgment, checking the packets to see if they get delivered within the sliding window, and the handling of reset packets.

4.2. Firewall and Security

In a private network, the proxy server can have dual purposes: to provide data caching and to provide access security. To provide better services, we should separate these two functions. Hence a firewall device is introduced that usually locates between proxies and external networks. Its primary function is to protect internal resources from the external uncontrolled Internet users.

FIGURE 4. A System Configuration with Proxies and Firewall for Outbound Access.



Clients outside the firewall must abide by a rigid structured methods for network traffic, defined by the firewall, to gain access to resources on the inner network. Typically, the firewall administrator must register all external users. For outbound services, a firewall is a circuit gateway imposes structure at the operating system layer, specifically in the network library of TCP/IP network calls: `socket()`, `connect()`, and `bind()`. All proxies need to be SOCKSified [10] to access through the firewall machine. For our proposed distributed proxy system, we may setup new proxy server on demand. Hence, it would be difficult to SOCKSify each of them unless all of them are using identical system platform. In Figure 4, we show a system architecture which puts an external SOCKSified proxy right outside the firewall. Then we can neglect the hassle of SOCKSification of all proxies within the private network.

1. The duration of 2MSL differs for different implementations, it is 60 seconds for NetBSD.

4.3. Cache Coherence

In the proposed architecture, there is a concern about the cache consistency among all proxies. Each home page may induce several TCP connections, different TCP sessions could go to different proxies under our scheme. In accessing some dynamic Web sites from a browser, one of the proxies may keep a stale document while the others contain up-to-date information. As a result, an access request might result a mixture of old and new information on the same page. This page-wise inconsistency phenomenon is undesirable. With the assumption that a dynamic Web site usually attracts a large number of requests which in turn help synchronize the cache information in different proxies. Hence, the situations of page-wise inconsistency at the client end may occur with very low probability.

There are several methods to tackle this problem. In HTTP/1.1 specification, there are many cache control directives. Reference [8] explains the usage of different HTTP headers for cache control. For example, "If-Modified-Since: *date*" can be used to check if a page is modified according to the advertised *date*. This technique is known as conditional GET. In order to guarantee obtaining the latest copy from a WWW server, we can use "Pragma: no-cache" in the HTTP header. Currently, both CERN and Netscape proxies will sent conditional GET upon receiving "Pragma: no-cache" directive. This is known as naive coherence mechanism. Though guaranteeing fresh copy, this method generates a lot of wide-area query traffic which is simply as bad as requesting a fresh new complete copy from the remote WWW server. The header "Cache-Control" in HTTP/1.1 can explicitly control the cache in a proxy server. However, "Cache-Control" cannot solve the page-wise inconsistency problem in our distributed proxy system. Therefore, the best method is to use a light-weighted protocol to synchronize the cache copies among the proxies. The Server Cache Synchronization Protocol (SCSP) [9] is a work in-progress for the cache consistency purpose. As we have mentioned before, the ICP is used to query the neighboring caches to find the cache objects. This protocol can also be enhanced easily to synchronize proxy caches by checking the version of a cache object.

4.4. System Reliability

Reliability of the system is a crucial factor to make a product successful. This multiple proxy configuration in the system design provides proxy level fault tolerance when a proxy server fails to operate properly. Upon detecting a dysfunctional proxy, this proxy will be removed from the distribution list of the depot. This detection is based on a simple timed out function upon any

client request. All subsequent TCP requests will be forwarded to the other properly operating proxy servers. Upon reception of revival information from this proxy again, the depot will then put it back into the active list.

When a proxy fails in the middle of a TCP session, the depot cannot simply re-direct the connection to other proxy. This is because the depot has no mechanism to setup the corresponding TCP session states within a TCP session to another proxy, such as the number of bytes sent, the current window size and sequence number etc. At the moment, we simply scrap the lost TCP session.

The depot presents itself as the single point of failure in the system architecture; therefore, we should have a standby sparing for the system fault tolerance. We have choice of using hot or warm standby. A sparing standby depot machine is always in an alert mode to assume the normal depot operations upon detecting a failure in the operating depot. For the hot standby, the replica actively monitors all input streams. Both the operating depot and the replica have virtually identical TCP state information about all connections. Detection of the depot software failure is not easy. We can implement a voting system which requires at least three pieces of depot software running concurrently. Bus-level voting mechanism can be implemented to vote on data periodically. However, this methodology requires tight synchronization of hardware and intimate knowledge of the hardware platform. A voting system also assumes that the software executing on redundant machines is correct and that design errors in processor that may cause simultaneous errors do not exist.

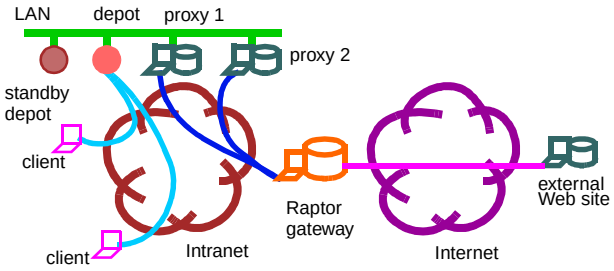
For the warm standby, the operating machine periodically sends the state information to the secondary to update the standby's database. In this situation, the two databases shall not be the same most of the time, but the goal of the warm standby is to reduce the number of affected TCP connections when failure occurs. We shall design the depot program with the concept of software testability, and implement a self-checking finite-state machine or "watch-dog" module to identify whether fault has occurred during the software run-time. Also in our design, both primary and secondary machines are able to detect the failure of the depot program on the other machine. If the standby detects the failure of the primary's depot program, it will assume the normal depot function immediately.

5. Testbed Implementation

At Nortel, we set up a testbed to check the feasibility and performance of this distributed proxy concept. The trial system consists of three machines connecting through 10Mbps shared medium Ethernet drops: one Pentium 133MHz machines running NetBSD 1.0 operating system serves as a depot, HP 755 and HP 712 are the two proxies connecting in parallel. The discrepancies in the performance of the HP 755 and HP 712 will require more careful explanations on the subsequent performance that we obtain in the following.

The number of connections per second (or requests per second), the number of bytes transferred per second (or throughput) and the round-trip response time are the common performance indices for a Web server [3]. The number of connections per second relates to the server performance, while the last two concern about the network performance. Moreover, the cache hit rate is an important measure for the performance of the proxy servers.

FIGURE 5. A Simplified Testbed Configuration at Nortel.



Depot: Pentium 133MHz, NetBSD 1.0
Proxy 1: HP 755
Proxy 2: HP 712

For the system performance evaluation, a benchmark software [11] is available to test the performance of a WWW server. We modified this software to test the performance of the proxy servers. This software allows us to set up concurrent connections to jam a server system and it will re-attempt to make connection even it is refused at the first place. Apart from the time granularity, the file size also constitutes a critical factor in determining the number of connections per second. This is because the larger the size of a file, the longer it takes to finish a request. Since the testing environment is not fully under control and the corporate network is quite busy, we will try to interpret the results as accurate as possible. We picked an evening to stress the proxies with two different methods:

1. We append the "Pragma: no-cache" to instruct the proxies to "conditional GET" the file information from the end WWW server. If there is no modification from the cached copies, the proxy delivers the information from its local cache. Since the cache will be flushed from time to time, the proxy will still fetch the copy at

the end WWW server occasionally. With this technique, any client request triggers some network flow of information.

2. We test the proxy with simple GET header, since we are accessing the same document in the test. Apart from the first GET, all the other should have 100% cache hit unless the cache is flushed. In this test, the proxy functions much like a WWW server.

For each of the above two cases, we do the testing in three different configurations:

1. only proxy 1 serves all Web requests without depot involvement,
2. only proxy 2 serves all Web requests without depot involvement, and
3. two proxies are loaded in round-robin fashion through the depot system.

In the experiment, we run the benchmark program with different numbers of simultaneous connections. The client on the intranet accesses a WWW server on the other end through the testing systems. Table 1 shows the test results using 25 simultaneous connections with a total of 1000 consecutive requests. Since the tests involve the use of intranet, the transient network state can affect the performance in each configuration.

TABLE 1. 25 concurrent requests for 1000 consecutive requests, with "Pragma: no-cache".

System	file size	Connections per second	Gross Throughput (KBytes/s)	Average Connection Time (ms)
HP 755	100 Bytes	8.08	2.21	2
	1 KBytes	7.97	9.32	2
	10 KBytes	6.12	62.64	8
HP 712	100 Bytes	3.37	1.32	2
	1 KBytes	3.37	4.44	2
	10 KBytes	3.32	34.93	2
Depot Proxy	100 Bytes	6.73	2.20	4
	1 KBytes	7.84	9.68	8
	10 KBytes	6.02	61.67	10

TABLE 2. 50 concurrent requests for 1000 consecutive requests, with "Pragma: no-cache".

System	file size	Connections per second	Throughput (KBytes/s)	Average Connection Time (ms)
Depot Proxy	100 Bytes	9.72	3.21	117
	1 KBytes	8.11	10.00	197
	10 KBytes	8.59	88.73	91

While using "Pragma: no-cache", we plot the results in Table 1 and Table 2. The performance of the depot proxy system falls in-between the two stand-alone HP systems as shown in Table 1. This is partially because the CPU power of the HP 712 is comparatively slower than the HP 755 which takes longer time to process network information. As a result, the depot proxy system performance falls in the middle. For the average connection time, the current depot program is in store-and-forward mode. Though the servers are running in parallel, hence inevitable delay may incur at the depot.

We increase the number of concurrent requests to 50 to each system. Only the depot proxy system can withstand the load, each stand-alone HP simply cannot handle the traffic requests and cannot complete the testing. In Table 2, we show the performance of the depot system under 50 simultaneous connections with 1000 completed requests. From these tests, the depot proxy system proves to be valuable in achieving the service availability objective.

TABLE 3. (a) HP 755, 21000 consecutive requests, with normal GET (~100% cache hit).

Number of concurrent requests	file size	Connections per second	Gross Throughput (KBytes/s)	Average Connection Time (ms)
10	100 Bytes	26.66	7.52	5
	1 KBytes	26.68	32.21	7
	10 KBytes	24.22	252.43	11
25	100 Bytes	24.88	7.02	276
	1 KBytes	25.70	31.02	317
	10 KBytes	22.40	233.53	365
50	100 Bytes	25.40	7.16	433
	1 KBytes	22.43	27.07	414
	10 KBytes	22.61	235.94	463
75	100 Bytes	24.50	6.91	508
	1 KBytes	22.58	27.25	277
	10 KBytes	22.20	231.37	312
100	100 Bytes	24.21	6.83	522
	1 KBytes	27.02	32.62	192
	10 KBytes	22.64	236.01	439
125	100 Bytes	22.58	6.37	581
	1 KBytes	24.35	29.39	464
	10 KBytes	22.40	233.76	577

(b) HP 712, 1000 consecutive requests, with normal GET (~100% cache hit).

Number of concurrent requests	file size	Connections per second	Gross Throughput (KBytes/s)	Average Connection Time (ms)
10	100 Bytes	92.89	45.33	3
	1 KBytes	57.31	81.00	3
	10 KBytes	46.33	492.52	15

Number of concurrent requests	file size	Connections per second	Gross Throughput (KBytes/s)	Average Connection Time (ms)
25	100 Bytes	70.87	34.59	4
	1 KBytes	93.90	132.75	3
	10 KBytes	56.52	602.09	22
50	100 Bytes	89.27	43.56	62
	1 KBytes	86.04	121.61	30
	10 KBytes	66.19	704.35	85
75	100 Bytes	78.66	38.39	60
	1 KBytes	90.93	128.55	31
	10 KBytes	68.69	731.02	106
100	100 Bytes	95.37	46.59	38
	1 KBytes	50.55	71.43	68
	10 KBytes	69.70	742.15	99
125	100 Bytes	84.38	41.18	60
	1 KBytes	92.69	131.04	45
	10 KBytes	75.36	803.75	69

(c) Depot proxy system, 1000 consecutive requests, with normal GET (~100% cache hit).

Number of concurrent requests	file size	Connections per second	Gross Throughput (KBytes/s)	Average Connection Time (ms)
10	100 Bytes	56.84	21.97	15
	1 KBytes	53.65	70.32	11
	10 KBytes	37.00	390.32	48
25	100 Bytes	55.03	21.35	139
	1 KBytes	54.34	71.30	142
	10 KBytes	36.30	383.29	221
50	100 Bytes	55.23	21.53	142
	1 KBytes	59.48	78.19	137
	10 KBytes	29.75	314.45	363
75	100 Bytes	57.96	22.73	110
	1 KBytes	51.25	67.52	186
	10 KBytes	18.99	200.29	167
100	100 Bytes	53.94	21.31	160
	1 KBytes	58.97	77.92	185
	10 KBytes	18.55	196.80	300
125	100 Bytes	56.92	22.64	229
	1 KBytes	50.35	66.60	289
	10 KBytes	16.36	174.35	1105

On the second tests, the result is shown in Table 3. The HP 712 has superior caching performance when compared to the HP 755. Therefore, on almost 100% cache hits, the HP 712 can sustain up to almost 80 requests per second disregard of the file size while the HP 755 can only provide 22 requests per second. The other reason is that the HP 712 is installed solely for the testbed while the HP 755 is the daily operating proxy in the intranet. The table size in the HP712 will be very small which help speeding up the cache search. With the depot proxy system, the performance again falls in-between of the two stand-alone machines. However, when the file size is large such as 10 KBytes as shown in Table 3(c), the performance of the depot system degrades drastically. The reason was found that the socket size in the UNIX kernel is limited to 56 KBytes in the system, hence the buffer overflow scenario occurred frequently when the hit rate is high and the file size is large. If we can increase the value, we should be able to push the performance up to around 50 requests per second.

6. Conclusion and Future Works

In this paper, we have outlined a depot system architecture which can scale the WWW proxy services. It facilitates the incremental growth of the proxy system, provides simple means to control and maintain the servers, and maximizes the availability of the service as a whole. To further improve the quality of services in the system, we shall investigate the mechanism in improving cache hit rate, for example, modifying the ICP for caches communication as well as using the SCSP for the cache coherence issue.

Reference

[1] NSFNET Backbone Service. NSFNET Traffic Distribution Highlights, **`http://nic.merit.edu/nsfnet/statistics/1995/`**.

[2] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, “Hypertext Transfer Protocol - HTTP/1.1”, work in progress, Aug. 12, 1996.

[3] N.J. Yeager, R.E. McGrath, *Web Server Technology*, Morgan Kaufmann Publishers Inc., 1996.

[4] D. Wessels, K. Claffy, “Internet Cache Protocol (ICP), Version 2”, work in progress, Nov. 1996.

[5] T. Brisco, “DNS Support for Load Balancing”, RFC 1794, Apr. 1995.

[6] N.G. Smith, “The UK National Web Cache: A State of the Art Report”, *Proc. 5th Intl. WWW Conf.*, Paris, 1996.

[7] J. Postel, “Transmission Control Protocol”, RFC 793, Sep. 1, 1981.

[8] A. Dingle, T. Partl, “Web Cache Coherence”, *Proc. 5th Intl. WWW Conf.*, Paris, 1996.

[9] J.V. Luciani, G. Armitage, J. Halpern, “Server Cache Synchronization Protocol (SCSP) - NBMA”, work in progress, Jun. 1996.

[10] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, “SOCKS Protocol Version 5”, RFC 1928, Mar. 1996.

[11] ZeusBench/1.0, **`http://www.zeus.co.uk/`**.