Development for a Simple Client-based Distributed Web Caching System

Jong Ho Park* and Kil To Chong**

*Department of Electronics and Information Engineering, Chonbuk National University, Chonju 561-756, Korea
(Tel: +82-63-270-2478; Fax: +82-63-270-2451; Email:q1253@chollian.net)

**Department of Electronics and Information Engineering, Chonbuk National University, Chonju 561-756, Korea
(Tel: +82-63-270-2478; Fax: +82-63-270-2451; Email:kitchong@chonbuk.ac.kr)

Abstract: Since the number of user-requests increases dramatically on the Internet, the servers and networks can be swamped unexpectedly without any prior notice. Therefore, the end-users are waiting or refused for the responses of the contents from the originating servers. To solve this problem, it has been considered that a distributed web caching system efficiently utilizes structural elements of the network. Because a distributed web caching system uses the caches that are close to end-users on the network, it transmits the contents to users faster than the original network system. This paper proposes a simple client-based distributed web caching system(2HRCS) that client can directly perform object allocation and load balancing without an additional DNS for load balancing in CARP (Cache Array Routing Protocol) and GHS (Global Hosting System) that are the recent distributed web caching system protocol. The proposed system reduces the cost of setup and operation by removing DNS that needs to balance the load in the existing system. The system has clients with consistent hashing method, so it extends its environment to other distributed web caching system that has caches of different capacity. A distributed web caching system is composed and tested to evaluate the performance. As a result, it shows superior performance to consistent hashing system. Because this system can keep performance of the existing system and reduce costs, it has the advantage of constructing medium or small scale CDN (Contents Delivery Network).

Keywords: Caching, 2HRCS, Simple Distributed Web Caching

1. Introduction

Since the delivery of the information over the web has been recognized as a new medium, internet users have been explosively increased in these days. The requests which are excessive against bottle necks or servers make the probability of delay or failure highly. Several researches that use the shared cache have been conducted to solve this problem. These researches are classified as a communication method [4][5][6][7] and a hash-based method [8][9][10][11][1][3] according to the method that finds and allocates the object among shared caches. The methods that use a communication waste the network resources among caches and the duplicated allocation of the object in those methods causes the waste of storing spaces. However, the methods that use a hash function propose the solution of these problems. Although CARP(Cache Array Routing Protocol)[1] of Microsoft corporation and Consitent Hashing[3] on GHS(Global Hosting System)[2] of Akamai Technologes differ in utilizing a hash function, these are general methods that recently are used.

In CARP, the loads over the shared caches are regulated through DNS [14] and the performance of load balancing is improved by the optimization of object allocation among caches, tuning the TTL(Time To Live) value of DNS [22]. The consistent hashing which is used in GHS uses an addictional DNS for the allocation and load balancing. Consequently, CARP and consistent hashing system have the disadvantage of high costs caused by their installation and maintenance, and they also have the disadvantage of delay caused by their hugeness and complexity.

If a client directly accesses the cache without additional DNS, simple distributed web caching system could be organized without the installation of dedicated DNS and could reduce delay within DNS. Especially, the small-scale CDN on local environment can be implemented with low costs, the load of DNS will be able to diminish in the existing network system. Therefore, this paper developes a simple client-based distributed caching system(2HRCS) that maintains the performance of existing consistent hashing system and reduces the processing time. The developed system is applied to the network that has different cache capacities because the caches in real world is different from each other with respect to their capacities. The system configuration and the problem of shared cache protocol are discussed in Section 2. Details about proposed client-based distributed web caching system are described in Section3. Simulation results and Conclusions are described in Section 4 and 5, respectively.

2. Shared Cache Protocols

The shared cache protocols are classified as the method implementing ICP and the hash routing method according to the methods that allocate the object over shared caches, and they have been developed through the mutual compensation of weak points in each method.

As a shared cache method, ICP[8] is to overcome the limits, such as processing time, capacity of storage , and number of user connections, that are caused by multiple users sharing a single cache. In order to find objects, the system with ICP basically performs a communication among shared caches through UDP/IP. In worst case, it communicates with N-1 caches excluding a primary cache. Therefore, when objects are not in the cache, ICP method causes delay and wastes the bandwidth of network because communication packets are

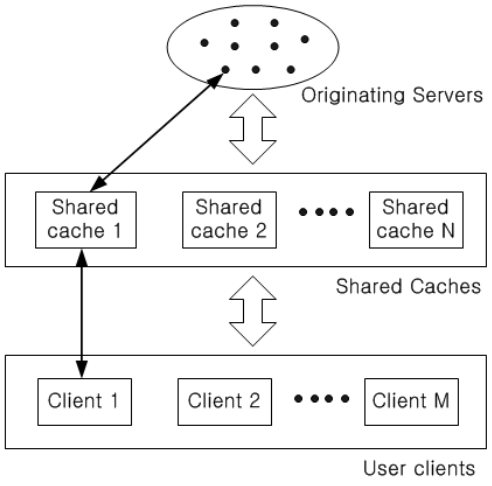

Fig. 1. A distributed web caching system of client-type.

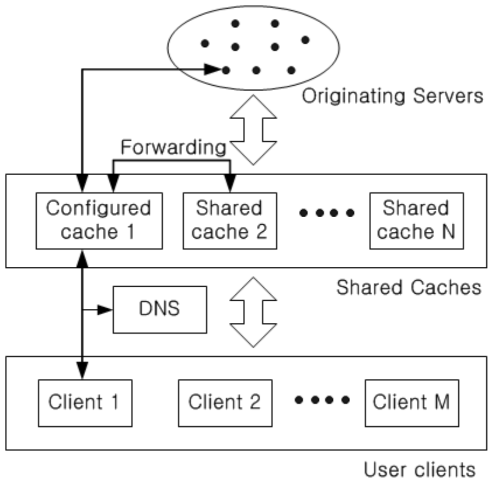

Fig. 2. A distributed web caching system of cache-type.

excessively produced. Also, it wastes the storing space since objects requested by clients are duplicated in each cache.

In order to ameliorate these problems, the methods that minimize the duplication of objects have been proposed. In these mehods, clients with a basic hash function allocate objects to corresponding caches by hashing the URL[13] of objects to be requested[1][3][8][9][10][11]. These methods belong to a hash routing protocol. They are classified as a client-type and a DNS-type, viewed in the compostion of system. Also, they are classified as a hash routing method[8][9][10][11][1] with ICP and a consitent hashing method[3] without ICP, viewed in the method of finding objects.A client-type that is a basic concept represents a ideal configuration that clients have a perfect hash function needed to allocate objects. As shown in Figure 1, after client1 makes a direct connection to shared cache1, an object is directly sent to client1 if it exists in the cache. Otherwise, it is copied to the cache and is transmitted to client1 if it is not in the cache. Although the configuration of this client-type can be accomplished by the modification of a client browser, the modication is hard to realize. Therefore, the system that is approximately a client-type is configured by the modifications of a cache-type as shown in Figure 2 and a DNS-type as shown in Figure 3. Actually, the system configurations of CARP[1] and consistent hashing[3] are a cache-type and a DNS-type, respectively.

In ICP, the methods[8][9][10][11] with simple hash routing use a basic hash function. A basic hash function is expressed as $h(u) = f(u) \text{mod } p$ in a distributed caching environment, where u is URL, $h(u)$ is hash function that makes fixed length of hashed value, and p is the number of caches. Consequently, the corresponding location of hashed value that represents the URL of an object can be determined within shared caches. That is, the communiation process that was necessary to find an object can be minimized by the direct configuration among clients and caches, reducing the duplication of an object at a maximum level.

However, every URL should be repeatedly hashed and allocated to caches whenever the addition or the removal of caches would occur. Therefore, Hit rate could be deteriorated by increased error or failure for user requests. Each of CARP[1] and consistent hashing[3] solves this problem in different ways. Clients use robust hashing in the configuration of CARP[1] that is shown in Figure 2. That is, hash function is used to create each pairs of $h(u, c_1), h(u, c_2), \cdots, h(u, c_n)$, where u is a hashed value of URL and c is a hashed value of cache domain name. When an object is corresponding the URL of n-th shared cache that has highest value among hashed values, it is requested to patch from configured cache1. The load of web cache system is regulated by DNS when client1 patches the IP address of configured cache1, if the domain name of configured cache1 is set in client1. Configured cache1 requests an object by forwarding the URL of shared cahce1 that has a highest hashed value, and it forwards an object to client without copying as soon as it receives a corresponding object. In order to apply to the environment that has different cache capacities, CARP can be utilized in proportion to the probability that hashed value of URL could be allocated to cache. That is, it can be extended to heterogeneous system. However, it cannot reduce the DNS usage or a delay on DNS.

Figure 3 shows the system of consistent hashing [3] on GHS [2]. Client uses a simple hash function to make a hashed value of URL u, and creates the domain name with hashed value in order to allocate the URL to the N imaginary caches. For instance, the URL of v101.cnn.com can be made. In this case, top-level DNS finds low-level DNS of distributed web caching system close to a user using domain delegation. And, client patch the IP of the real cache and requests the URL, after low-level DNS allocates a corresponding object to the cache. To begin with, cache sends an object to client after patching it from original web server. Then, cache will send an object on its storing space if client requests it afterward. Consistent hashing was not intended to expend to the heterogeneous system, because it established on the basis of efficient cache capacity. Furthermore, it produces a delay on DNS although it is quite different from CARP[1]. And, it could bring about the maintenance problem of DNS if the system grows large.

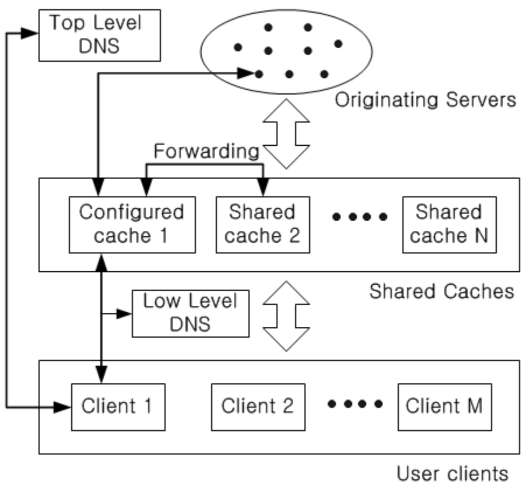

Fig. 3. A distributed web caching system of DNS-type.

3. Development of a Simple Distributed Web Cache System with Client-based

The proposed client-based distributed web caching system in this paper is hybrid robust hash routing client system with single-tier configuration. The proposed system will be written as 2HRCS from now on. Each client is configured to include consistent hashing algorithm and uses new load balancing algorithm that is different from old one. As, shown in Figure 4, this system consists of many programs, such as CP Helper, Proxy Helper, and client. Throughout this paper, the load information consists of a load information flag and a list of hot pages, and the cache information includes the load information and IP addresses.

CP Helper is server-side application program, and it periodically receives and stores the cache information of each cache. When clients initially connect to a web server, CP Helper transmits the information to clients. That is, clients receive the cache information of shared caches that is close to them. Because CP Helper has a load information flag that is used to balance the load, clients can allocate objects to shared cache when they connect to the CP Helper.

Proxy Helper can produce a load information flag on the basis of maximum processing capacity that represents the total number of HTTP requests served by cache during unit time. To produce the flag, it is coded with Perl because Perl can manipulate the enormous log file of each shared cache at a high speed. And, C language is used to realize TCP/IP communication. On the response of CP Helper's request, Proxy Helper analyzes the log file of cache. It finds the hot pages(hot virtual names) that have relatively high number of requests with respect to total number of requests. Then, it makes the corresponding load information flag(H:high, M:middle, L:low), according to the maximum processing capacity allowed by a cache.

A redirecting client browser is implemented with HTTP 1.1[12]. And it receives the cache information from CP Helper, fetching the HTML source from a web server. Initially, client tries to find an existing DNS on Internet when it

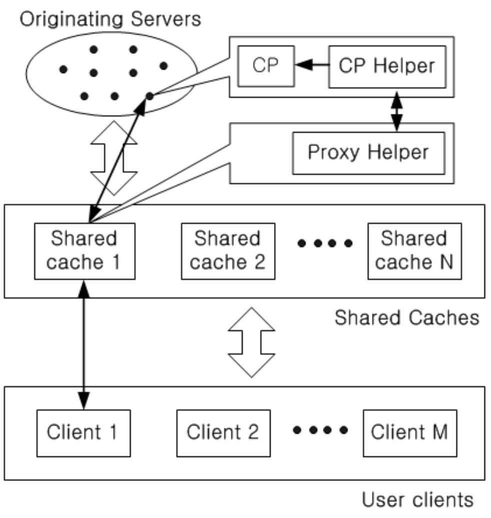

Fig. 4. The Hybrid Robust Hash Routing Client System(2HRCS).

gets the IP address of CP server. Afterwards, it allocates the object URL that was originally contained in HTML source to caches without DNS, using builtin consistent hashing and load balancing algorithm. That is, a client does not use DNS except when it gets the IP address corresponding to the domain name of CP. The development of a redirecting client browser leads an existing distributed web caching system to a simple system without using DNS, so it makes a delay on DNS less.

To balance the load, 2HRCS operates in two ways. Firstly, it executes hard load balancing when the number of request per unit time is close to the maximum processing capacity, because the number of requests on a cache could temporarily show the dramatic increase. When client accesses the CP Helper, it receives the load information flag that is defined with three levels. And then, client allocates all hot pages to the cache that has low level of the load information flag. Therefore, the load is distributed to the cache gradually and certainly, whenever the connection of client is increased. In last case, load balancing would be activated if hot page that has the considerably high number of requests exists, although total requests are lower than maximum processing capacity. In this case, distributing hot pages does not waste the storage space of a cache because the number of hot pages that will be duplicated to other cache is quite small. Client distributes the hot page that has the highest number of requests to the cache that has the lowest load. Although this load balancing would be slower than the first case, it makes the uniform load balancing of shared caches because it distributes requests more precisely. The load balancing is performed by setting up the range of the load information flag for each cache in proportion to the maximum processing capacity, whether the maximum processing capacity is same across shared caches or not. The advantage of this system is that the load balancing can be realized in a heterogeneous distributed web caching system that its shared caches have different processing capacities each other. Without DNS or complicated calculation, this system uses simple load information flags that were originally received from CP Helper.

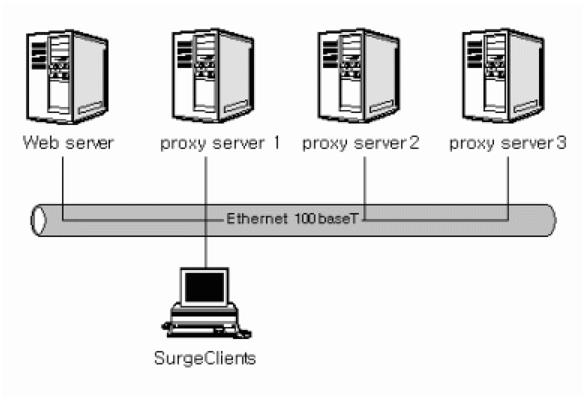

Fig. 5. A simulation system.

4. Simulation

4.1. System Setup for Simulation

Generating the desired amount of requests could be complicated in existing client, so we modified Surge[15] that is developed in Boston University. Hash function[16] is utilized to produce hashed value for the URL of object. As soon as Surge starts to run, each client thread fetches a list of caches from CP Helper. And it gets the load information according to requests from caches. A client uses the consistent hashing algorithm, produces 1000 imaginary caches by means of modulo-arithmetic, stores up hashed value for IP Address of real caches in a binary tree, allocates imaginary caches to real caches, and performs the subdivision of blocks with copied caches. Afterwards, using hashed values of URLs, a client finds the information of real cache IP address that is corresponding to the imaginary cache name, and receives an object from the cache after requesting its URL. To perform this experiment, the parameter of Zipf's law should be tuned to make popularity taking account of a characteristic of cache[17][18][19]. If all files of a server are requested, the number of requests for a file is expressed as $p = \alpha\gamma^{\beta}$, where α is a positive constant, and $\gamma = \frac{Events_{Doc}}{Events_{Total}}$ is rank of files. The estimated value β is supposed to be 1 when web server takes workload, and it is tuned to a smaller value than 1 when cache server takes load. In experiments, the β is set to 0.8. As an operating system, Linux Redhat 7.2 kernel version 2.4.7 is employed to install the request generator. And the modified Surge is set up in IBM compatible PC.

CP Server is configured to provide contents using Apache release 1.3[21] that is generally used in Pentium III machine with Linux. Additionally, CP Helper is developed on CP Server. Client accesses Apache and CP Helper simultaneously. Like a client, Surge is modified to perform the function that requests object files. To analysis a logging information, access_log on /var/log/httpd/ is used. This file contains access IP, access time, HTTP command, and error message. Therefore, the load information of web server can be analyzed by PERL script or C language. Cache servers are implemented with Squid release 2.3[20] on three IBM compatible machines to configure a distributed web caching system. Each server responds with default port to HTTP requests. Proxy Helper on each cache is installed to analyze Squid log file and communicate with CP Helper. When CP Helper requests the load information, Proxy Helper transmits the information to CP Helper after analyzing the log file for a time interval. Load information is set to be overlapped data using smaller requesting interval. And, the load balancing can cope with the dynamic changes of cache loads. The network configuration of the experimental system is shown in Figure 5.

4.2. Simulation Results

When the same amount of requests is given to each distributed caching system that each cache has same processing capacity per unit time, the load of distributed caching system with consistent hashing is shown in Figure 6, and that of 2HRCS is shown Figure 7. The performance of load balancing per unit time on each proxy is better in 2HRCS than in consistent hashing. When caches have different processing capacities, that is, the distributed web caching network is a heterogeneous system, the load of consistent hashing system is same as Figure 6 and the that of 2HRCS is shown in Figure 8. In this case, processing capacities of each cache are voluntarily set to 2000 in proxy 1, 3000 in proxy 2, and 8000 in proxy 3.

As shown in Figure 8, requests on proxy1 and proxy2 is distributed to proxy3. In this figure, x axis and y axis represent a time and the number of requests, respectively. The differences of average hit rates are negligible as shown in Table 1. Using clients, 2HRCS can perform the load balancing and object allocation without DNS, so can minimize a delay on DNS.

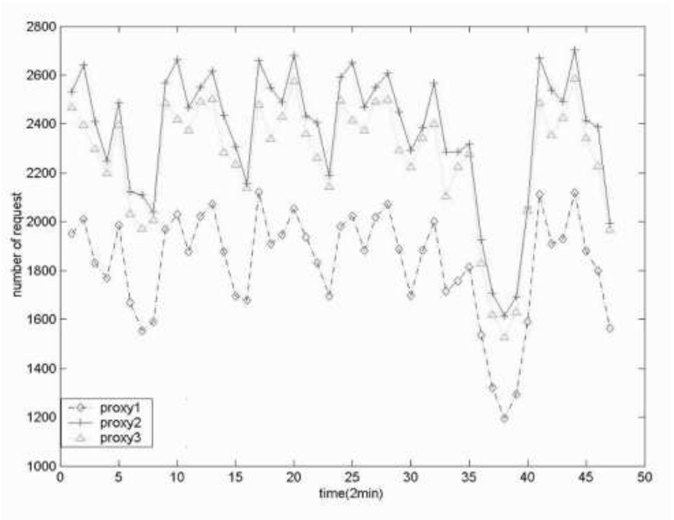

Fig. 6. The load of Consistent hashing in homogeneous system.

5. Conclusions

In this paper, several problems of protocols on existing distributed web caching system have been investgated. Through the modification of client browser that has consis-

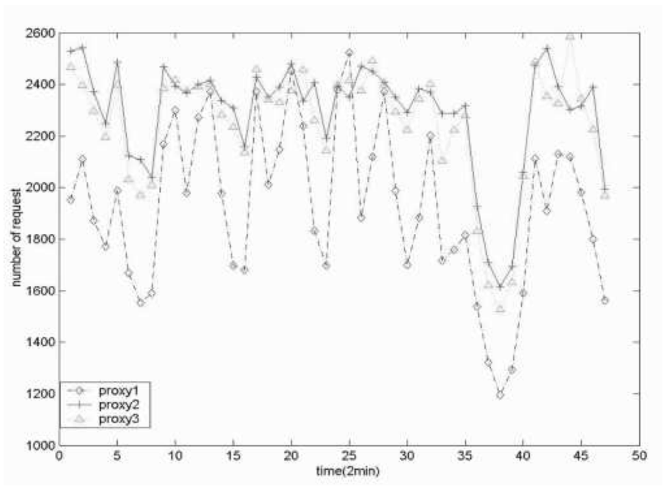

Fig. 7. The load of 2HRCS in homogeneous system.

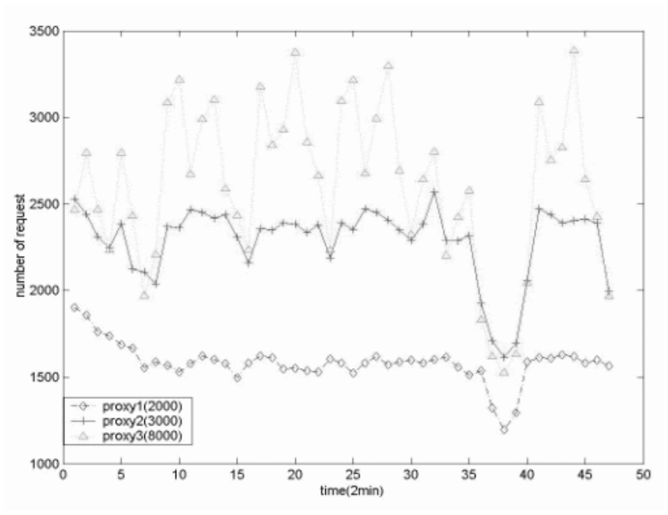

Fig. 8. The load of 2HRCS in heterogeneous system.

tent hashing, the client-type system is configured to simplify the system without additional DNS that is an essential element in CARP[1] and GHS[2]. The proposed system is implemented to balance the load on client side, regulating the generation limits of the load information flag in proportion to the maximum requests of each distributed cache. The simulation results show efficient performance in realistic distributed web caching system that deals with different caches in capacity.

In conclusion, proposed system can reduce the expenses of installation and maintenance, because it is simplified to remove the DNS that is necessary to perform load balancing in other systems. Since it extends the consistent hashing to the heterogeneous environment, its performance is superior to the consistent hashing system in real distributed web caching systems. Therefore, this system can be applied to small or middle-sized CDN with low costs, keeping the performance of existing system unchanged. Furthermore, this client-type distributed web caching system could be configured more efficiently, if the setting bounds of load information flag is dynamically adjusted according to other load parameters that will be added to caches in the future.

Table 1. Average hit rate of each proxy(shared cache).

Cache	Consistent Hashing in homogeneous system	2HRCS in homogeneous system	2HRCS in heterogeneous system
Proxy1	99.155 %	99.145 %	99.130 %
proxy2	99.384 %	99.410 %	99.358 %
proxy3	99.510 %	99.495 %	99.730 %

References

[1] V. Valloppillil and K. W. Ross, *Cache array routing protocol v1.1. Internet Draft*, http://www.globecom.net/ietf/draft/draft-vinod-carp-v1-03.html, 1998.

[2] Leighton, et al., *Global hosting system. US Patent, 6,108,703*, http://www.delphion.com/cgi-bin/viewpat.cmd/US06108703_, 2000.

[3] D. Karger, et al., *Web caching with consistent hashing*, In Proceedings of the 8th International World Wide Web Conference,May 1999.

[4] A. Chankhunthod, et al., *Hierarchical internet object cache*, In USENIX, 1996.

[5] R. Malpani, et al., *Making world wide web caching servers cooperate*, In forth International World Wide Web Conference, 1995.

[6] S. A. Gadde, et al., *A taste of crispy squid*, In Workshop on Internet Server Performance, http://www.cs.duke.edu/ari/cisi/crisp, 1998.

[7] L. Fan, et al., *Summary Cache:a scalable wide-area web-cache sharing protocol*, Technical Report 1361, Computer science dept., University of wisconsin, 1998.

[8] D. Wessels and K. Claffy, *Internet cache protocol(ICP) version 2.*, RFC 2187, http://icp.ircache.net/rfc2187.txt, 1997.

[9] V. Valloppillil and J. Cohen, *Hierarchical HTTP routing protocol. Internet Draft*, http://icp.ircache.net/draft-vinod-icp-traffic-dist-00.txt, 1997.

[10] D. G. Thaler and C. V. Ravishankar, *Using named-based mappings to increase hit rates*, To appear in IEEE/ACM Transactions on Networking, 1997.

[11] Sharp, *Super proxy script:How to make distributed proxy servers by URL hashing*, White Paper, http://naragw.sharp.co.jp/sps/, 1996.

[12] R. Fielding, et al., *Hypertext transfer protocol-HTTP/1.1*, RFC 2068, http://www.faqs.org/rfcs/rfc2068.html, 1997.

[13] T. Berners-Lee, et al., *Uniform Resource Locators(URL)*, RFC 1738, Network Working Group, 1994.

[14] P. Albitz and C. Liu, *DNS and BIND(3rd edition)* O'Reilly & Associates Inc., 1998.

[15] P. Barford and M.E. Crovella, *Generating representative web workloads for network and server performance evaluation*, In Proceedings of ACM SIGMETRICS Conference, 1998.

[16] R. Rivest, *The MD4 Message-Digest Algorithm*, RFC 1320, Network Working Group, 1992.

[17] A. Mahanti and C. Williamson, *Web proxy workload characterization*, Technical report, Department of Computer Science, University of saskatchewan, 1999.

[18] C. Cunha, et al., *Characteristics of WWW client-based traces*, Technical report BU-CS-95-010, Department of Computer Science, Boston University, 1995.

[19] M. Arlitt, et al., *Workload characterization of a web proxy in a cable modem environment*, HP Labs Technical report HPL-1999-48, 1999.

[20] D. Wessels, *Squid Web Proxy Cache*, http://www.squid-cache.org.

[21] R. McCool, *The Apache Software Foundation*, http://www.apache.org.

[22] X. Tang and S. T. Chanson, *Optimal Hash Routing for Web Proxies*, In Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS), 2001.