Development of a Distributed Web Caching Network through Consistent Hashing and Dynamic Load Balancing

Hwan Chang*, Jong Ho Park**, Ju Ho Park***, Kil To Chong**** *Regular Members*

ABSTRACT

This paper focuses on a hash-based, distributed Web caching network that eliminates inter-cache communication. An agent program on cache servers, a mapping program on the DNS server, and other components comprised in a distributed Web caching network were modified and developed to implement a so-called "consistent" hashing. Also, a dynamic load balancing algorithm is proposed to address the load-balancing problem that is a key performance issue on distributed architectures. This algorithm effectively balances the load among cache servers by distributing the calculated amount of mapping items that have higher popularity than others. Therefore, this developed network can resolve the imbalanced load that is caused by a variable page popularity, a non-uniform distribution of a hash-based mapping, and a variation of cache servers.

I. Introduction

The growth of Web traffic has led to a considerable increase in the amount of traffic over the Internet. This traffic causes a swamped server and network, increased latencies at the end user, and reduced network bandwidth available for other requests.

A request might travel through multiple caching systems on its way to the original server, since cache servers are introduced to address these problems. Caching reduces network bandwidth usage, lessens user-perceived delay, and lightens loads on the original servers [1]. As caching has become an important topic, several cooperative or distributed caching architectures have been proposed to ameliorate the problems related to fault tolerance and scalability that are the disadvantages of the single cache architecture [2][3][4][5][6]. However, most of these systems consume excess bandwidth with packets caused by inter-cache communication, although these systems have higher hit rates with increased cache spaces and client populations. Also, there are some distributed architectures without inter-cache communication, such as the hash-based request redirecting scheme, TCP-based switching, and TCP-based packet rewriting scheme [7][8][9].

The developed network in this paper chooses a so-called consistent hashing method. A dynamic load-balancing algorithm is then implemented to dynamically resolve the unbalanced load that is caused by a different page popularity and a status variation of cache servers.

II. Consistent hashing and load balancing

1. Consistent hashing

To distribute the objects across the cache servers, clients should know which cache to query for a specific object. Hashing is a desirable approach for that purpose, because it is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.

A standard hashing function is represented by $h(u) = f(u) \bmod p$ in this distributed network,

* Department of Control and Instrumentation Engineering, Chonbuk National University(chang921@chollian.net),
** Department of Mechatronics Engineering, Chonbuk National University(q1253@chollian.net),
*** Department of Computer Science, Chonbuk National University(juhpark@hotmail.com),
**** Division of Electronics and Information Engineering, Chonbuk National University(kitchong@moak.chonbuk.ac.kr)
논문번호 : 020367-0823, 접수일자 : 2002년 8월 23일

where u is a bit string that represents URL (Uniform Resource Locators), $f(u)$ is a general hash function that makes a fixed-length value of hashed result, and p is the number of available cache servers. Our distributed caching network uses so-called consistent hashing that is described in [7] and is able to map a URL to a cache server without regard to the variation of the number of available cache servers, and there is no need to send the status information of caches to clients.

Ideally, the URLs are mapped into the unit circle. The URLs are assigned to the closest cache going clockwise or counterclockwise. If one cache is removed from this network, the mapping points of all URLs do not change and some URLs should be remapped to other cache severs.

2. Mapping with consistent hashing

To implement this concept on a real network, the virtual cache servers are introduced to map the URL with the unchanged number p, which is considered to be a live cache in clients. Then every virtual cache instead of a URL is mapped into the unit circle. The mapping with consistent hashing is divided into two parts, client and DNS(Domain Name System), as shown in Figure 1. A client keeps the invariable mapping information between URLs and virtual caches via the hash function and the modulo-p division function. And a DNS keeps the variable mapping information between virtual caches and real caches via communication with cache servers.

3.Dynamic load balancing

The numbers of copies of the cache servers could be spread into the unit circle to produce a more uniform distribution of virtual caches to real caches [10]. However, increasing the copies of the cache servers is not sufficient to achieve a uniform distribution of virtual caches to the caches, although some extent of distribution is achieved by simply increasing the copies because there could be a random nature of a hash-based mapping. Also there are many possibilities that could disrupt the load balance, and the biased load can be severe in case of hot pages or a higher request rate. Thus, a dynamic load balancing algorithm is implemented to dynamically resolve the biased load.

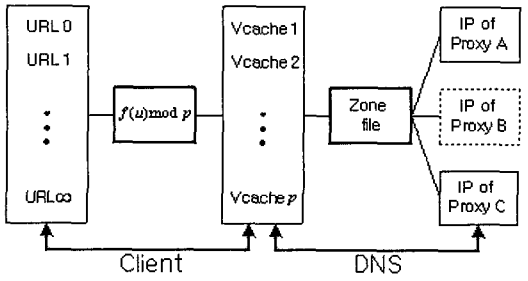

Fig. 1 Mapping for consistent hasing

The load information on cache servers is periodically collected by a kind of agent program and is sent to a DNS server. Another mapping program that is performed on a DNS server analyzes the load information and then modifies the mapping information in a zone file of a DNS server to balance the load among cache servers.

A mapping program that is performed on a DNS server periodically classifies a cache server as either busy, idle, or normal node. The values that determine the busy line and the idle line are defined by user, according to the desired level of deviation. And, these values are represented by $v(v \geq 0)$ and $w(w \geq 0)$, where M is the mean value of the all nodes and M' is the mean value of the busy nodes and the idle nodes. As a real workload, the load for the distribution in this algorithm represents the number of requests during desired seconds..

Following pseudo-code shows the detailed procedure of the proposed algorithm, where n_i is the load of a node i, p is the number of busy nodes, and q is the number of idle nodes.

1 request/receive the load information
2 calculate the standard deviation, D
3 **if** D > desired limit
4 define the node with $M+v$ and $M-w$
5 calculate the mean value,

$$M' = \frac{1}{p+q}\left(\sum_{i=1}^{p} n_i + \sum_{i=1}^{q} n_i\right)$$

6 determine the surplus load,

$$DL = \sum_{i=1}^{p} (n_i - M)$$

7 **for** $j \leftarrow 1$ to q

8 load to be distributed to idle node j,

$$DL \cdot \frac{\sum_{i=1}^{p} n_i - n_j}{\sum_{i=1}^{p} n_i}$$

9 calculate the number of popular virtual caches

10 modify/reload the zone file (round-robin DNS)

11 wait for desired seconds

The load metric that is the primary concern throughout this paper is the rate of HTTP (Hypertext Trasnfer Protocol) requests per second to a cache server. The request rate to a cache server is selected according to the hit rate and the effect of hot pages, without taking into consideration the performance differences among the cache severs.

Ⅲ. Network configuration

1. Modification of Request generator

For workload generation, we used the tool called Surge [11] that was developed at Boston University. Each client system in Surge could be a set of users that always talks to the fixed local cache server without modification, in the case where the cache sever was used between server and client. Surge was modified to send requests to the corresponding cache severs according to the hashed results. As a hash function, MD4 [12] is implemented to hash the URL. MD4 is generally known as a faster algorithm, among standard hash functions in the realm of cryptology. In every request, the number of a set of virtual caches divides the 128-bit output of MD4. Consequently, a DNS query is sent to resolve the corresponding real cache server among the set of virtual caches. At that point, the TCP(Transmission Control Protocol) connection is made to the real cache server.

After tuning Surge to compile under Linux, C language functions of MD4 and modulo-1000 are linked up with Surge.

2. Development of Log file analyzer

A distribution of Squid-2.1 Release is installed as a cache proxy server [13]. To communicate with the DNS server, a kind of agent program runs on the cache that analyzes the Squid's access log and sends the load information to the mapping program on the DNS server. When a DNS server queries an agent program, it replies with the load information that is represented by the desired format, after analyzing the log file during the last desired seconds.

The communication module of the agent program is coded using C language, and the log analyzing module of that program is coded using Perl script. Because the log files have a massive amount of data and should be analyzed with the least possible delay, Perl is considered a very powerful scripting language for that purpose.

3. Development of mapping program

A BIND 8.2.3 distribution is installed to configure a DNS server [14]. A mapping program is developed to satisfy the consistent hashing function and dynamic load balancing function. The program was developed with C++ language and complied under Linux platform to control the BIND process on the DNS server. The binary tree is utilized for storing the hashed values of all cache servers that are comprised of virtual caches. In case of variation, all virtual caches are rewritten into copied caches with consistency under the order of a binary tree. That is, some of the virtual caches with higher popularity that correspond to the surplus load are distributed to all the cache servers or desired cache servers using DNS Round Robin [15].

4. Setup of distributed caching network

Figure 2 shows a diagram of the distributed Web caching network that is implemented to achieve the consistent hashing and the dynamic load balancing algorithm. To request a desired URL, Client sends the query with virtual cache name that is produced by the hash function and the modulo-p arithmetic. And then, DNS resolves

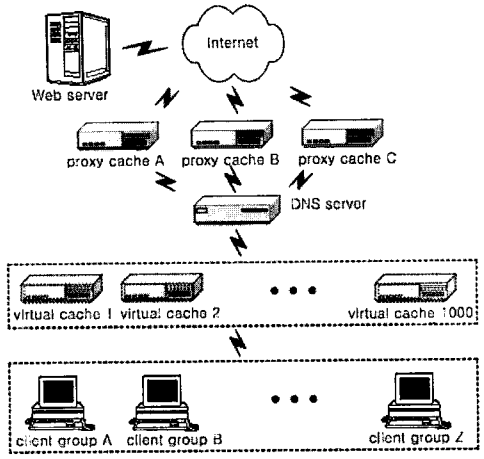

Fig. 2 Distributed Web caching network

the virtual cache name into the real IP(Internet Protocols) number of one of the live cache servers.

The mapping program on DNS server requests the status and the load information of cache servers every 30 seconds. An agent program of the cache server replies to it with a request rate after analyzing the log file during the last 30 seconds. The mapping program modifies the zone file, and then BIND is reloaded with the modified zone file and continuously serves as the virtual cache resolver.

Ⅳ. Simulation results

1. Comparison with single networks

Figure 3. (a) shows that the developed network has a relatively lower miss rate than one of three single caching networks, which leads eventually to a higher hit rate, a lower latency, and a lower bandwidth demand. Figure 3. (b) shows the stacked graphs when three single cache architectures are incorporated into a cooperative caching network with ICP [16]. The amount of ICP that would be increased is critical when the request rate is increased or the capacity of a cache is increased, whereas this distributed caching network does not have a chance of having an increased rate of communication packets.

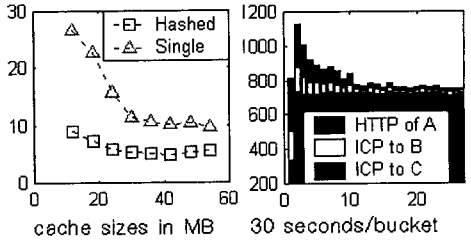

Fig. 3 (a) Miss rates, (b) Request rates

2. Performance of dynamic load balancing

To test the operation of a dynamic load balancing against the varying status of cache servers, the processes of cache servers are intentionally manipulated to make the changes of status. Figure 4. (a) shows the result when a cache server *B* is died at 10 minutes and a cache server *C* is died at 20 minutes, successively. Also, this network restores the original mapping information on returning the previous condition, as shown in Figure 4. (b).

To prevent from being congested into a specific cache, the previous method that is introduced in [7] spreads all of the hot virtual caches to all of the other virtual caches. Then, the number of virtual caches is slowly reduced by subtracting one at a time. However, in this paper, the status

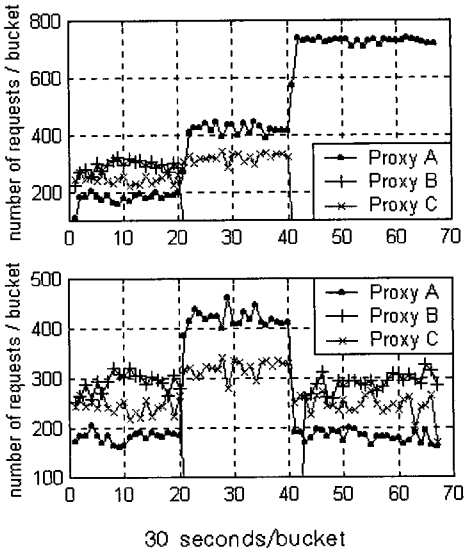

Fig. 4 Operation of dynamic load balancing

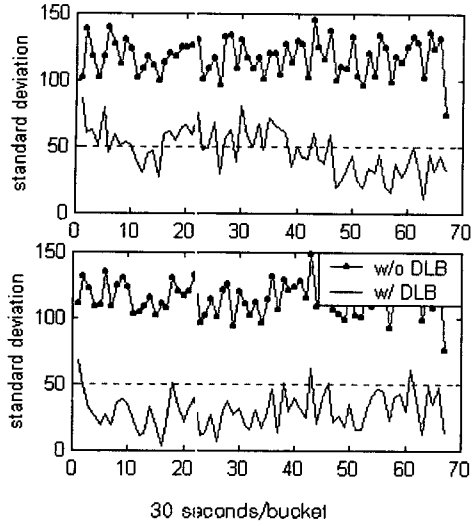

Fig. 5 Standard deviation difference between spreading and reduction method(upper) and calculated distribution method(lower)

and load of all real caches are checked and calculated at every requested time and the surplus load of busy node is dumped into the idle node if the deviation among caches breaks the desired level. Therefore, this dynamic load balancing method responds faster than previous method that spreads into all caches including normal nodes, as we can see in Figure 5. Lower lines of the both figures show the standard deviations of the distributed networks using each dynamic load balancing method.

In both cases, the standard deviation in each method is declined below the desired level of 50. That is, the algorithms appropriately distribute the load among initially unbalanced caches. However, there is difference between response speeds. Actually, this amount of the biased load can be appeared in case of the modified number of live caches or the presence of hot pages, though virtual caches are initially well distributed. Also, Figure 6 shows that degradation of the hit rates does not exceed the 5 percent of the original hit rate. Hit rate is a key performance measure of a cache server and this result means that the proposed system maintains a higher hit rate as other distributed cache networks do.

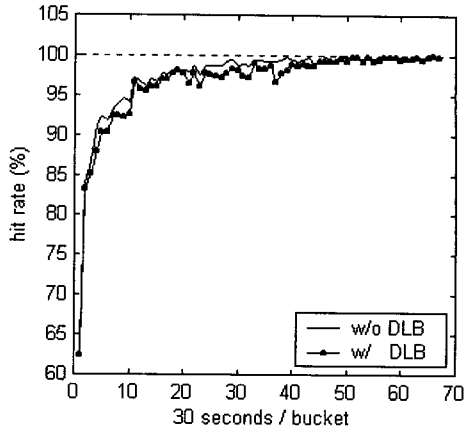

Fig. 6 Performance of hit rates

V. Conclusion

To implement a so-called consistent hashing, all components that comprise a distributed Web caching network were modified and developed, respectively. And a dynamic load balancing algorithm was applied to dynamically balance the load, and it uses the real load on cache servers as a basis of distribution. Moreover, this algorithm continuously classifies the real caches as busy, normal, and idle node and distributes the surplus load of busy nodes to idle nodes at every requested time to handle the balancing problem effectively and fast. Also, a higher hit rate is maintained without severe degradation.

Applying this simple algorithm, a DNS sever can dynamically resolve the unbalanced load that is caused by a different popularity among pages, a non-uniform distribution of a hash-based mapping, and a variation of the number of live cache servers in the distributed caching network. Also, this paper presents the possible applications for a more advanced algorithm which is capable of handling changes in a real network environment, using the real workload other than request rate on cache servers as a basis for distribution.

References

[1] B.D. Davison, "A web caching primer", *IEEE*

Internet Computing, Vol. 5, Issue 4, pp. 38-45, 2001.

[2] A. Chankhunthod, et al., "A hierarchical Internet object cache". *Proceedings of the 1996 USENIX Technical Conference*, pp. 153-163, January 1996.

[3] L. Fan, et al., "Summary cache: A scalable wide-area web cache sharing protocol", *IEEE/ACM Transactions on Networking*, pp. 281-293, Volume 8, Issue 3, 2000.

[4] S. Gadde, et al., "A Taste of Crispy Squid", *Workshop on Internet Server Performance (WISP98)*, June 1998.

[5] R. Malpani, et al., "Making World Wide Web Caching Servers Cooperate", *Proceedings of the 4th International World Wide Web Conference*, pp. 107-110, December 1995.

[6] P.S. Yu and E.A. MacNair, "Performance study of a collaborative method for hierarchical caching in proxy servers", *Proceedings the 7th International World Wide Web Conference*, pp. 215-224, April 1998.

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

[8] K.L.E. Law, et al., "A scalable and distributed WWW proxy system", *Proceedings of ACM Multimedia '97*, 1997.

[9] L. Aversa and A. Bestavros, "Load balancing a cluster of web servers: using distributed packet rewriting", *Proceedings of the 2000 IEEE IPCCC*, pp. 24-29, 2000.

[10] D. Karger, et al., "Consistent hashing and random trees : Distributed caching protocols for relieving hot spots on the World Wide Web", *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 654-663, 1997.

[11] P. Barford and M.E. Crovella, "Generating representative web workloads for network and server performance evaluation", *Proceedings of the ACM SIGMETRICS Conference*, pp.151-160, 1998.

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

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

[14] Internet Software Consortium, "ISC BIND", http://www.isc.org/products/BIND.

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

[16] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), version 2, RFC 2186", *Network Working Group*, September 1997.

장 환(Hwan Chang) 준회원

1999년 2월 : 전북대학교 제어계측공학과 졸업
2002년 2월 : 전북대학교 제어계측공학과 석사
<주관심 분야> 자동제어, 신호처리, 컴퓨터 네트워크

박 종 호(Jong Ho Park) 준회원

1997년 2월 : 전북대학교 농업기계공학과 졸업
2000년 3월 ~ 현재 : 전북대학교 메카트로닉스공학과 석사과정
<주관심 분야> 컴퓨터 네트워크, 시스템공학, 자동제어

박 주 호(Ju Ho Park) 준회원

1997년 2월 : 전북대학교 전산통계학과 졸업
2000년 3월 ~ 현재 : 전북대학교 전산통계학과 석사과정
<주관심 분야> 실시간 시스템, 운영체제, 컴퓨터 네트워크

정 길 도(Kil To Chong) 정회원

1984년 5월 : 오레곤 주립대 기계공학과 졸업
1986년 12월 : 죠지아 공대 기계공학과 석사
1992년 12월 : 텍사스 A&M 대학 기계공학과 박사
<주관심 분야> 컴퓨터 네트워크, 시간지연시스템, 자동제어