RC 21420 (96708) 09MAR1999 Computer Science/Mathematics

Research Report

Load Balancing and Hot Spot Relief for Hash Routing among a Collection of Proxy Caches

Kun-Lung Wu, Philip S. Yu
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

IBM Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Load Balancing and Hot Spot Relief for Hash Routing among a Collection of Proxy Caches

Kun-Lung Wu and *Philip S. Yu*

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Correspondent: Kun-Lung Wu
Tel: 914-784-6615; Fax: 914-784-7455
email: klwu@watson.ibm.com

Abstract

This paper studies an *Adaptable Controlled Replication* (ACR) approach to load balancing and hot spot relief for a collection of loosely-coupled proxy caches coordinated by a hash routing protocol, such as CARP. Hash routing partitions the entire URL space among the proxy caches, creating a single logical cache. Each partition is assigned to a cache server. Client requests to a cache server for non-assigned-partition objects are forwarded to sibling caches. Duplication of cache contents is eliminated. However, in the presence of access skew, the load levels of the cache servers can be unbalanced, limiting the benefits of hash routing. We propose an adaptable controlled replication of non-assigned-partition objects in each cache server to reduce the possibility of repeatedly forwarding hot-spot request to the same server. As a result, cpu utilizations among the cache servers are more balanced and the problem of hot-spot references can be relieved. Trace-driven simulations are conducted to study the effectiveness of ACR. The results show that (1) access skew exists, and the load of the cache servers tends to be unbalanced in hash routing; (2) with a relatively small amount of ACR, such as 10% of the cache size, significant improvements in load balance can be achieved without a noticeable degradation in caching effectiveness; (3) ACR provides a very effective remedy for load imbalances due to hot-spot references; and (4) increasing the cache size does not improve load balance unless replication is allowed.

Index Term: proxy caching, adaptable controlled replication, load balancing, hot spot relief, hash routing, WWW.

1 Introduction

The growth in the usage of the World Wide Web has been increasing exponentially. As a result, response times for accessing web pages have in some cases become unsatisfactorily slow. One popular approach to improving the web access time is to employ proxy cache servers between the client browsers and the content servers. In fact, more and more organizations, such as Internet Service Providers and commercial corporations, are using a collection of cooperating cache servers to help improve the response time as well as reduce traffic to the Internet [22, 23, 13, 1, 16].

Hash routing, such as the cache array routing protocol (CARP) [19], is an emerging approach to coordinating a collection of loosely-coupled proxy caches [18, 16, 19]. It is a deterministic hash-based approach to mapping an URL object to a unique proxy cache. Hashing thus partitions the entire URL space among the caches, creating a single logical cache spread over many caches. Each proxy cache is responsible for the web objects belonging to the assigned partition. No replication of any cached object exists among the sibling caches [19]. In a hash routing protocol such as CARP, a client sends an HTTP request to its configured cache server. The configured cache server computes a hash function [1] based on the URL of the requested object. If the requested object belongs to the assigned partition, the cache server either returns the object to the client from its cache, if found, or fetches it from the content server. On the other hand, if the requested object belongs to a non-assigned-partition, then the cache server forwards the request to the sibling cache that is responsible for the object. After receiving the object from the sibling cache, it returns the object back to the client. In order not to replicate any object among the caches, a cache server does *not* place an object on its own cache, if the object does not belong to the assigned partition [16].

Without replication, the effective cache size is the total aggregate of all sibling caches. Global cache hit rate is thus maximized and the total incoming traffic from content servers due to cache misses is minimized. However, in the presence of access skew, accesses to a hot page will be repeatedly forwarded by other servers to the same cache server, potentially creating an overloaded server. With unbalanced load among the cache servers, the benefits of hash routing can be severely reduced.

In this paper, we study an *adaptable controlled replication* (ACR) of non-assigned-partition

[1] The hash function can also be computed by the clients. However, some of the older browsers may not be equipped with such capabilities. As a result, we focus only on the server-executed hash routing.

objects in each cache server to reduce or eliminate the possibility of repeatedly forwarding requests for a hot page to the same cache server. Namely, when a configured cache server receives an object from a sibling cache, a copy of the object is also placed on the configured cache, even though it is a non-assigned-partition object. With replication, references to a hot page can be serviced from the local cache, reducing the possibility that the owner of the page becomes overloaded. Note that replication reduces the effective aggregate cache capacity and thus could increases the total incoming traffic from the content servers. Thus, the amount of buffer space for replication may have to be controlled. We propose a unique cache management scheme that allows a controlled amount of non-assigned-partition objects to stay in the cache only if they are frequently referenced. If the replicated objects are not recently referenced, they will be quickly replaced by other assigned-partition objects. In other words, the amount of replication is adaptable to the changing reference patterns. As a result, the degradation to the total caching effectiveness is minimal.

Trace-driven simulations were conducted to study the effectiveness of adaptable controlled replication on load balance and hot-spot relief. Specifically, we examined the sensitivities of the maximum utilization of all cache servers to the skew in the assignment of client browsers to cache servers and the amount of accesses to a hot page. The results show that (1) access skew does exist and the load among the cache servers tends to be quite unbalanced in hash routing, even if clients are uniformly assigned to the cache servers; (2) with a limited replication of non-assigned-partition objects, significant improvements can be achieved in load balance among cache servers without a noticeable degradation in total caching effectiveness; (3) adaptable controlled replication is very effective in reducing load imbalances due to hot-spot references; and (4) increasing the cache size does not improve load balance unless replication is allowed.

As an alternative to hash routing, the *internet cache protocol* (ICP) [21] can be used to coordinate a collection of loosely-coupled proxy cache servers. ICP is an application-layer protocol running on top of user datagram protocol/internet protocol (UDP/IP). It is the engine used by the Harvest/Squid cache software to coordinate and share a hierarchy of proxy caches [3, 20]. In a single-tier collection of sibling caches, ICP works as follows. A client sends a request to its configured cache server. If this configured cache server cannot satisfy the request, it broadcasts a query using ICP to all the other sibling caches. If at least one sibling cache has the object, the configured cache server then sends an HTTP request for the object to the first sibling that responds to the query with an ICP hit message. Upon receiving the object, the configured cache stores a copy in its cache and forwards the object to the client. If no sibling cache responds after a time-out, then the configured cache server fetches the object from the content server. As a result, copies of the same object can be simultaneously present in multiple caches. In addition, it is very costly to locate an object because a large number of messages are generated and a long wait time is needed in the case of a time-out.

Various load balancing techniques based on domain name servers (DNS) have been proposed [9, 10, 5] for the Web, especially for the content servers. These techniques manipulate the host name to IP address mapping to distribute requests among a clusters of servers. Another approach to constructing scalable web servers is to use a front-end router to distribute incoming requests either at the TCP level [7, 8] or at the IP level [17]. There are also many papers on the general issues of proxy web caching, such as [2, 4, 23, 13, 1, 16]. However, none of the above deals specifically with load balancing and hot-spot issues of hash routing for a collection of loosely-coupled proxy caches.

Replication in hash routing for proxy web caches is similar to shared virtual memory [12], distributed file system [15] and remote caching [11] in that they all try to use multiple caches to cooperate on caching. However, there are distinct differences among them. Most importantly, these prior works do not partition the object space among the cache servers. Shared virtual memory and distributed file system decide which workstation to cache a shared object on a demand basis from program executions and file access patterns, respectively. In remote caching, the objective is to find a sibling cache that can cache an object once the object is about to be evicted from another cache. In contrast, the hash partition for which a cache server is responsible is generally fixed, unless the hash function is changed. Furthermore, the amount of replication is specifically controlled and managed.

The paper is organized as follows. Section 2 describes the details about hash routing with adaptable controlled replication. Section 3 presents the simulation model, system parameters and workload models used in the simulations. Section 4 then shows our performance results from the trace-driven simulations. Finally, Section 5 provides a summary of the paper.

2 Adaptable controlled replication for hash routing

Fig. 1 shows the architecture of a collections of loosely-coupled proxy web caches, which can be connected by a local area network (LAN) or a regional area network. The proxy

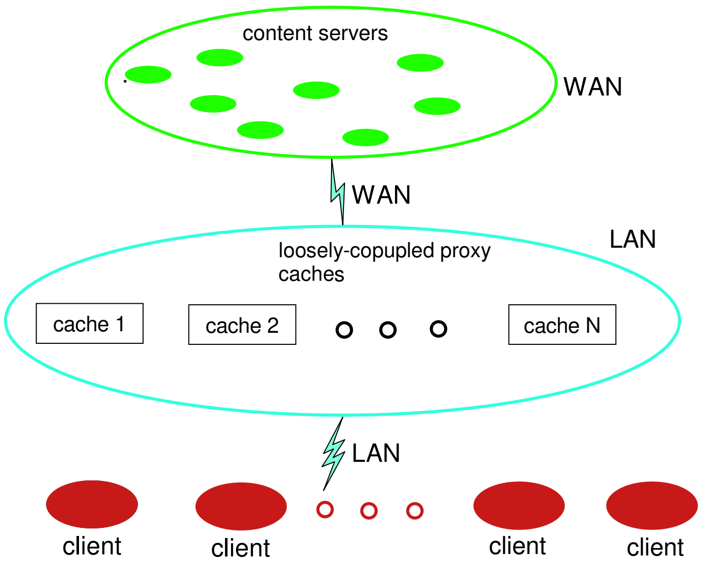


Figure 1: A collection of loosely-coupled proxy caches connected by a LAN.

web caches are siblings; namely no hierarchical order exists among them. Each client is configured to connect to one of the sibling caches.

With local replication, there are two kinds of objects in each cache. One belongs to the assigned partition and the other belongs to any of the non-assigned partitions. Objects that do not belong to the assigned partition represent replicated objects. To control the amount of replication and to make it adaptable to the changing reference patterns, we use two LRU stacks. Fig. 2 shows the implementation of the two LRU stacks. One is called *local* LRU stack and the other called *regular* LRU stack. Objects belonging to the assigned partition of a cache server can be in either local or regular LRU stack, but locally replicated objects (i.e., objects not in the assigned partition) can only be in the local stack. The aggregate size of the two stacks amounts to the total cache size of a server. When checking if an object is present in a cache, both LRU stacks are searched. On a cache hit, the object is moved up to the top of the corresponding LRU stack.

New objects to be cached are pushed into the top of the local LRU stack. Conceptually, the bottom of the local LRU stack in Fig. 2 represents a threshold in the cache. Any non-assigned-partition object will be discarded once it is pushed beyond the threshold. In our implementation, the objects at the bottom of the local LRU are always replaced by any new objects to be cached. However, if a replaced object belongs to the assigned partition, then a copy of it is pushed into the top of the regular LRU stack, which in turn will push out other

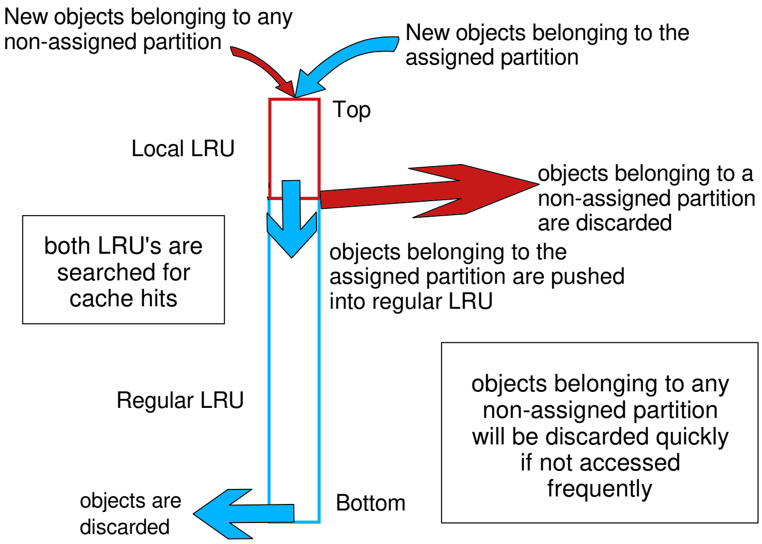


Figure 2: Two LRU stacks for adaptable controlled replication.

objects at the bottom of the regular LRU stack.

This two-LRU cache management effectively controls local replication without degrading overall caching effectiveness. Objects belonging to a non-assigned partition can stay in the cache only if they are frequently referenced. If there is little reference locality to the non-assigned-partition objects, they will be quickly replaced. In other words, the existence of the local LRU stack does not severely reduce the cache capacity for the assigned-partition objects. Hence, the locally replicated objects can effectively capture reference locality without unnecessarily occupying cache space. Namely, the amount of replication is adaptable to the dynamic reference patterns.

The modified hash routing protocol with adaptable controlled replication works as follows. Each client is configured to a particular cache server, called *direct server* in this paper. When a client sends an HTTP request to its direct server, it first examines if the object can be found in its own cache by checking both the local and regular LRU stacks. If yes, the object is returned to the client. If not, then the configured direct server computes a hash function based on the request's URL and sends an HTTP request to a unique sibling cache that is responsible for the hash partition to which the URL belongs. This unique sibling cache is called the *partition owner* of the requested object. The partition owner in turn tries to satisfy this request from its cache. If the object can be found, then the partition owner sends the object back to the direct server via an HTTP reply message. If not, the partition owner fetches the object from the content server, places a copy on its cache, and then sends it back to the direct server [2]. Once the direct server receives the object, it places a copy of the object on its cache and then forwards the object back to the client. In the worst case, the object is sent from the content server, to the partition owner, to the direct server and finally to the client, all through HTTP messages. In summary, in the modified hash routing protocol, (a) the direct server first checks its cache for the requested object upon receiving a request from a client; (b) the direct server also stores a copy of an object on its own cache upon receiving it from the partition owner; and (c) separate LRU stacks are used to manage controlled local replication of non-assigned-partition objects in each cache server.

3 Simulation implementation

We implemented a trace-driven simulator that models the modified hash routing with adaptable controlled replication among a collection of sibling caches. Table 1 shows the definition of various system parameters and their default values used in our simulations. Traces were used to drive a collection of 8 proxy cache servers. For each cache server, we implemented a CPU server and a buffer manager. For the CPU server, we implemented a FIFO service queue. The service time for processing an HTTP message, a request or a reply, is T_{http}. The CPU service time for looking up an object from its cache or storing an object into its cache is T_{cache}. And the CPU service time for computing the hash function based on a request's URL is T_{hash}. We assumed that $T_{cache} = 0.5 \times T_{http}$ and $T_{hash} = 0.1 \times T_{http}$ [3]. The mean message delays for the LAN and WAN are T_{lan} and T_{wan}, respectively. In our simulations, we assumed that the ratio of T_{http}:T_{lan}:T_{wan} is approximately 1:5:50 [6]. We chose 0.04 second as the default T_{http} so that the average CPU utilization during the peak hours will be in the range of $50\% \sim 80\%$ for various traces.

For each buffer manager, we implemented two LRU stacks: one local LRU and one regular LRU. The sum of the two stack sizes is the total cache size for the proxy cache server. To prevent an extremely large object from replacing many smaller objects, we set a threshold α for a cached object [1]. Only if the size of an object is less than the threshold, can it be cached.

[2]Note that the direct server can also be the partition owner of an object. In that case, no HTTP messages would be forwarded.

[3]We varied the sizes of T_{cache} and T_{hash} relative to T_{http} and found that the results were not sensitive to T_{cache} and T_{hash}.

Notation	Definition (Default values)
N	total number of sibling caches (8)
T_{http}	CPU overhead for an HTTP request/reply message (0.04s)
T_{cache}	CPU overhead for looking up an object from its cache or storing an object into its cache ($0.5 \times T_{http}$)
T_{hash}	CPU overhead for computing URL-based hashing ($0.1 \times T_{http}$)
T_{lan}	mean message delay from one cache to a sibling cache (0.22s)
T_{wan}	mean message delay between a cache and a content server (2.2s)
α	threshold (as a percentage of cache size) used for preventing a large object from being cached (10%)

Table 1: System parameters.

Proxy traces collected between 08/29/1996 and 09/09/1996 by Digital Equipment Corporation (now merged with Compaq Computer Corporation) were used to drive our simulations [14]. The sizes of the 12 days' traces varied from about 300K to over 1.3M entries. For this paper, we concentrated our reports on two large traces with over 1.3M entries. Each trace entry contains timestamp, client, server, size, type, URL and other information. The client, server, and URL were already mapped into unique integers to protect privacy. These traces were used to represent requests made to the 8 proxy servers in our simulations. Each request is assigned to one of the proxy servers based on its client ID. Then, a hash function is applied to the URL to find the partition owner of the URL.

In our simulations, client to direct server assignment follows a Zipf-like distribution, that is, a parametric distribution where the probability of selecting the i-th item is proportional to $1/i^{(1-x)}$. This distribution, Zipf(x, L), for a set of L items is given by $q_i = c/i^{(1-x)}$ for each $i \in \{1, \cdots, L\}$, where $c = 1/[\sum_{i=1}^{L} 1/i^{(1-x)}]$ is a normalization constant [24]. Fig. 3 shows various distributions of clients among 8 proxy servers, including Zipf(1.0, 8) (corresponding to a uniform distribution), Zipf(0.75, 8) and Zipf(0.5, 8). Zipf(0.5, 8) is the most skewed case while Zipf(1.0, 8) is the least skewed case. Note that the skew shown in the cases of Zipf(0.75, 8) and Zipf(0.5, 8) in Fig. 3 represents only the distribution of client assignments among proxy servers. It is not the distribution of direct requests among the proxy caches. The request-to-server distribution is determined by both the traces and the client-to-server assignment. Skew in the client-to-server assignment can be representative of differences in the number of active browsers making requests during a particular period of time. For example, assume that there are 100 clients assigned to each proxy cache. At a particular

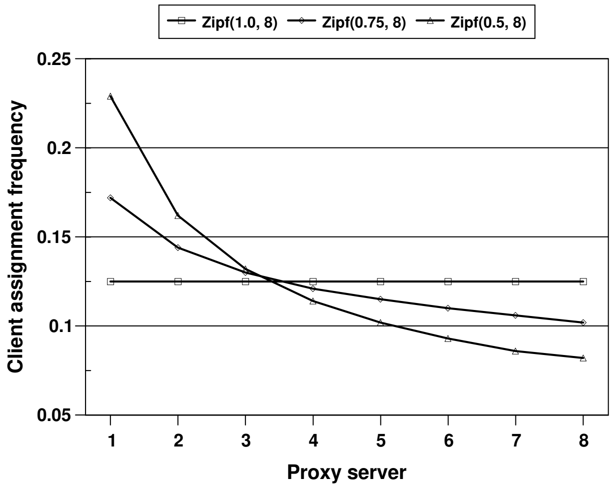


Figure 3: Various distributions of clients among 8 proxy servers.

point of time, there may be 80 active browsers making requests to proxy A while only 20 active browsers making requests to proxy B.

To model references to a hot spot, we injected requests to a specific page [4] from different clients during the simulations in addition to the traces. Each client is equally likely to generate such requests to the hot page. We varied different percentages of total references to this hot page in our simulations. For example, assume that 5% of the total references are going to the hot page. There is a 95% chance that the next request for the simulation will come from the trace, while a 5% chance it will be a request for the hot page. The arrival time of a request for the hot page is determined by the simulation time when such a request is issued, and the client ID of a hot-page request is chosen randomly from a uniform distribution.

To determine the proper cache sizes for our simulations, we calculated through simulations the maximal buffer size needed for each proxy cache if no replacement were to be required. We called this *MaxNeeded* [1]. In general, for the large traces, *MaxNeeded* is about 700–750Mb for each proxy cache. In other words, the *MaxNeeded* for the entire trace is about 5Gb to 6Gb. For most of our simulations, 350Mb was used as our default cache size, representing about 50% of *MaxNeeded*. Different cache sizes were also examined in our simulations.

[4] The URL of this hot page was set to 5 and its size was set to 35,928 bytes.

4 Results

4.1 Performance metric for load balance

A good performance metric to measure the load level and overloaded instances is the cpu utilization of the proxy caches. In particular, looking at the most utilized proxy cache server, one can deduce if the load level of a collection of proxy caches is balanced or not [5]. Hence, the effectiveness of adaptable controlled replication on load balance is evaluated by focusing on the maximum proxy utilizations observed during the peak hours of each trace between 5:00 am and 12:00 noon Pacific Time. We focus on the utilization level of the proxy cache server with the maximum utilization among all servers at each time instant. The server with the maximum utilization changes over time. However, if the maximum utilization is low, it means that no proxy server is overloaded at that time. For example, assume that there are three proxy servers. If their utilizations are 60%, 65% and 70%, respectively, at time t_1, and 90%, 60% and 40%, respectively, at time t_2. The maximum utilization at t_1 is 70% and that at t_2 is 90%. With a maximum utilization of 90%, the system may have a load balancing problem at t_2. By examining the histogram or cumulative frequency of the maximum utilization, we can determine the percentage of time during the observation period at least one of the proxy servers is overloaded (say exceeding 90% utilization).

In this paper, we use the *cumulative frequency* of the maximum utilization among all proxy cache servers as the performance metric for load balancing. Specifically, at the occurrence of an observation or sampling instant, we compute the utilization for each proxy cache server in the previous observation period. Then we track the cumulative frequency of the maximum utilizations during the peak hours. Fig. 4 shows the distribution of the maximum utilization among eight proxy cache servers for three traces using hash routing without replication. There was no skew in the client-to-server assignment and no additional references to a hot spot in these experiments. For comparison, we also plotted an ideal case for a well-balanced system with mean utilization at 75%. In this ideal case, the probability of the maximum utilization being less than 75% is zero. But, the probability of the maximum utilization being less than or equal to 75% is one. Hence, the distribution chart resembles a step function at 75%. Among the three traces in Fig. 4, the 9-4-1996 trace has a smaller degree of load imbalance problem while the 9-6-1996 trace has a larger degree. For instance, the probabilities of the maximum utilization being less than or equal to 90% are about 0.42 and 0.93 for the 9-6-1996 trace and the 9-4-1996 trace, respectively. In other words, there is

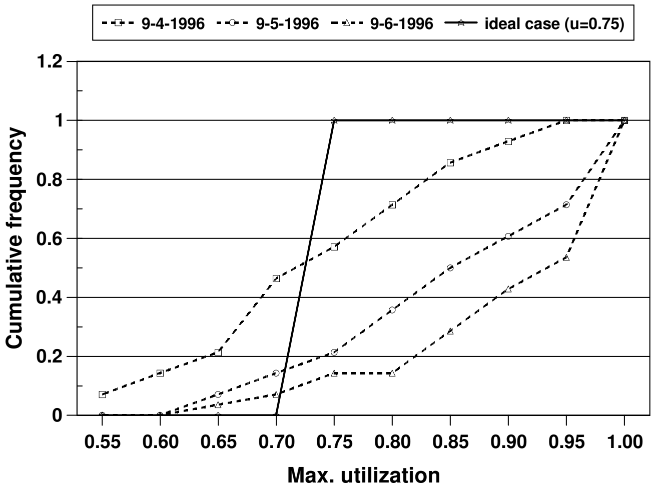


Figure 4: Distributions of the maximum utilization for three traces.

a 58% (i.e., $1 - 0.42$) chance that there is an overloaded proxy server during the peak hours (5:00 am to 12:00 noon) for the 9-6-1996 trace, while there is only a 7% (i.e., $1 - 0.93$)chance for the 9-4-1996 trace.

In general, the load of the system is more balanced if the cumulative frequency chart moves towards the upper-left corner and less balanced if the chart moves towards the lower-right corner. Fig. 4 demonstrates that even if there is no additionally injected hot-spot references and no skew in the client-to-server assignment, the load among a collection of proxy servers can be quite unbalanced. To understand the impacts of hot-spot references and skew in client-to-server assignment on load imbalances, Fig. 5 shows the distributions of the maximum utilization for the 9-6-1996 and 9-4-1996 traces. The impacts of hot-spot only, skew in client-to-server assignment only, and both are presented. These charts show that both hot-spot references and skew in client-to-server assignment can cause severe load imbalances among proxy cache servers in hash routing without replication. For instance, the probability of the maximum utilization being less than or equal to 90% drops from 0.93 to 0.68 when a 10% hot-spot reference and a Zipf(0.5, 8) skew were introduced to the 9-4-1996 trace. It is even more severe in the 9-6-1996 trace, where it drops from 0.42 to 0.15, suggesting that there is an 85% (i.e., $1 - 0.15$) chance that at least one proxy server is over 90% utilized.

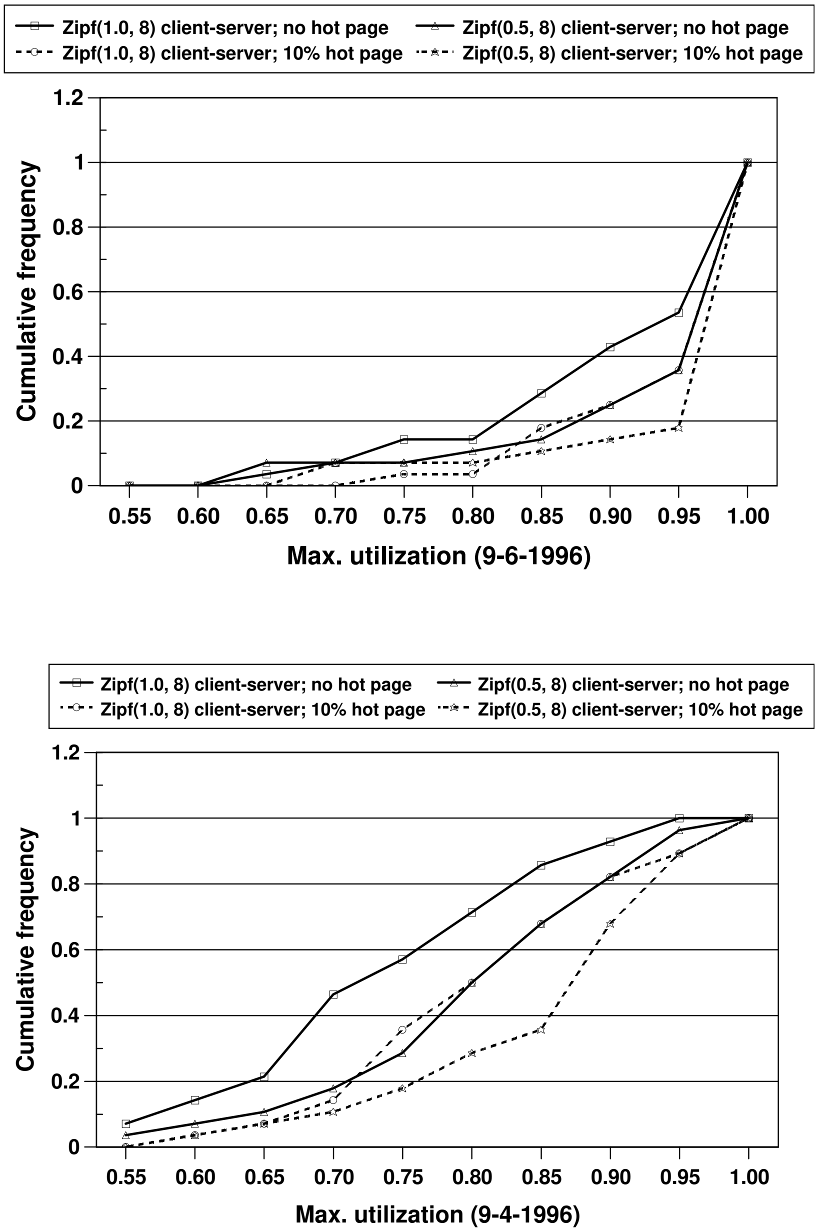


Figure 5: Impacts of hot-spot and skew in client-server assignment on load imbalances.

4.2 The impact of replication amount

Fig. 6 shows the impacts of various amounts of replication on load balance. For this set of experiments, we used a Zipf(0.5, 8) distribution for the client-to-server assignment and allowed 5% of the total requests to go to the hot-spot page. Four different amounts of replication were evaluated, 0%, 10%, 30% and 50% of the cache size. In general, there are improvements on load balance among the proxy caches with a larger amount of replication for the 9-6-1996 trace. However, for the 9-4-1996 trace, there is no difference among the 10%, 30% and 50% amounts of replication. For the 9-4-1996 trace, the cumulative frequency for the maximum utilization being less than or equal to 90% is one for all three cases. If not otherwise specified, we will use 10% of the cache size as our default replication amount for our simulations.

In Fig. 6, significant improvements were observed in load balance for both traces with a 10% replication. For example, the probabilities of the maximum utilization being less than or equal to 90% improve from less than 0.2 to 0.3 and from 0.8 to 1.0 for the 9-6-1996 trace and the 9-4-1996 trace, respectively. In other words, with a 10% replication for the 9-4-1996 trace, no proxy cache is over 90% utilized during the peak hours. However, without replication, there is a 20% (i.e., $1 - 0.8$) chance that at least one proxy cache is over 90% utilized.

One potential tradeoff for a larger amount of replication is the reduction in cache capacity for assigned-partition objects, causing an increase in total incoming traffic to the proxy caches from content servers. Fig. 7 shows the total incoming traffic to the collection of proxy caches during the observation period (5:00 am to 12:00 noon) with different amounts of replication. Here, the total incoming traffic measures the total amount of data fetched from the content servers to the proxy caches during the observation period. In general, compared with the no-replication case, more incoming traffic is generated for the cases with various amounts of replication. However, only marginal increases in incoming traffic were observed as the replication amount increases from 0% up to 50%.

4.3 The impact of hot-spot references

Fig. 8 presents the distributions of the maximum utilization with various amounts of hot-spot references for both 9-6-1996 and 9-4-1996 traces. In these simulations, there was no skew in the client-to-server assignment. In general, if a larger portion of total references are going

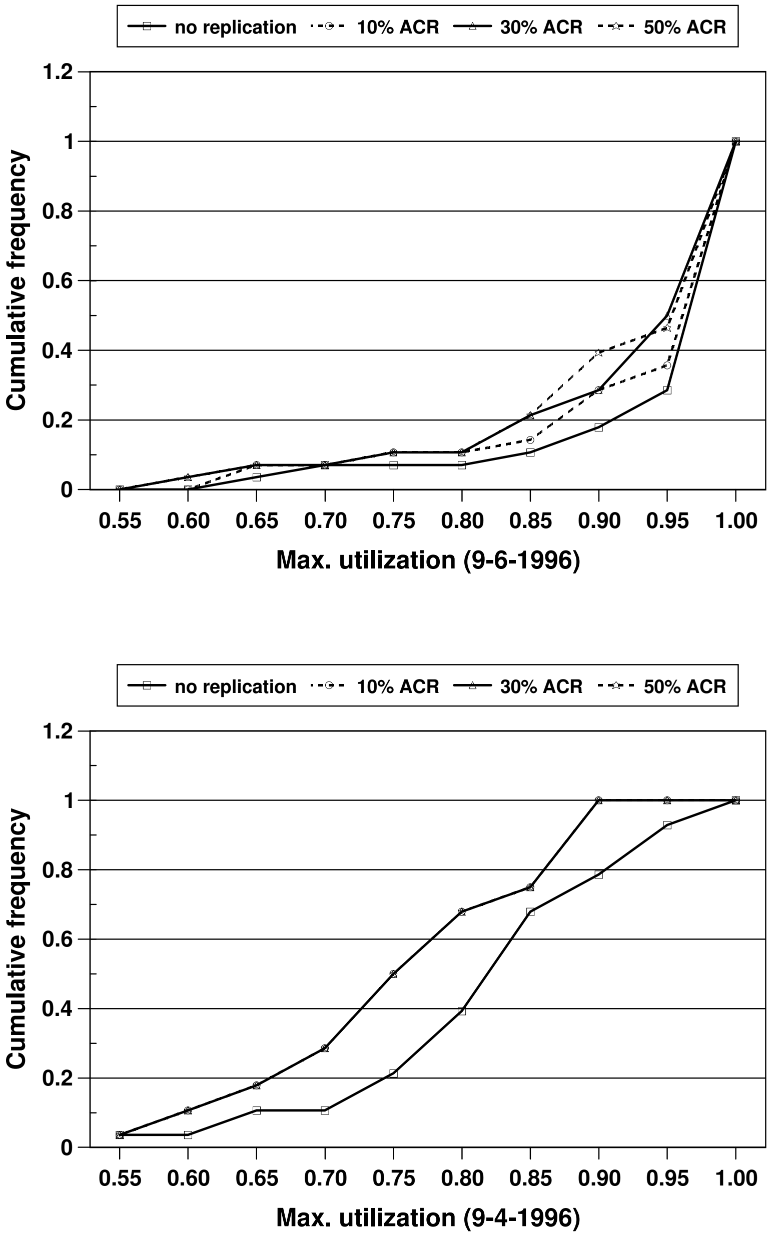


Figure 6: The impact of adaptable controlled replication on load balance.

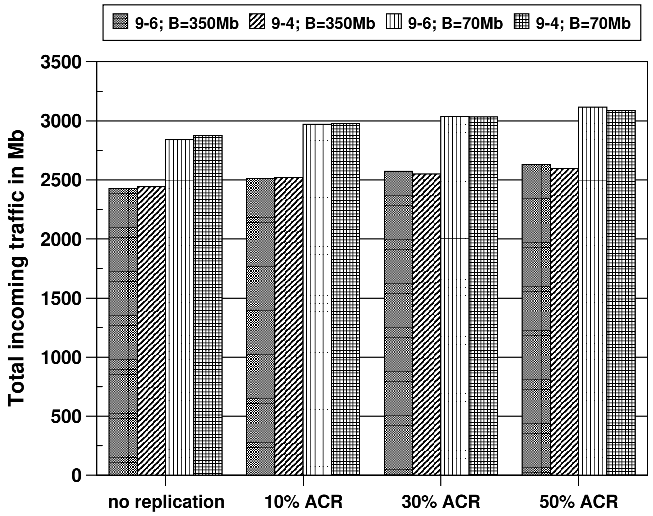
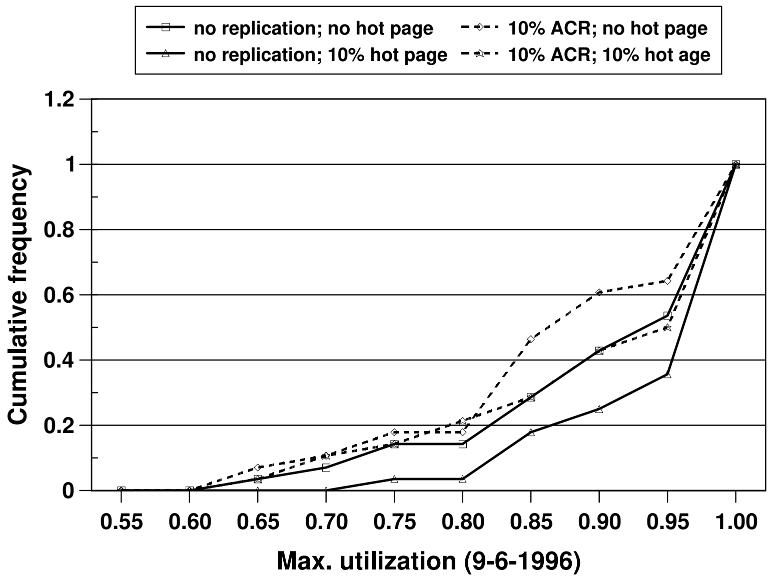


Figure 7: The impact of adaptable controlled replication on total incoming traffic.

to the hot spot, then the system will be less balanced. However, with a 10% replication, one can achieve a significant reduction in load imbalances caused by hot-spot references. For example, without replication, the probability of the maximum utilization being less than or equal to 90% drops from 0.429 to 0.25 for the 9-6-1996 trace if 10% of the references are going to the hot spot. It drops from 0.929 to 0.821 for the 9-4-1996 trace. However, with a 10% replication, the probability can be improved from 0.25 to 0.429 for the 9-6-1996 trace and from 0.821 to 0.964 for the 9-4-1996 trace. The improvement is even more significant if we look at the probability of the maximum utilization being less than or equal to 75% for the 9-4-1996 trace. It improves from 0.357 to 0.714 for the case with 10% of the references going to the hot spot.

4.4 The impact of skew in client-to-server assignment

Besides hot-spot references, the skew in client-to-server assignment can also cause significant load imbalances. Fig. 9 shows the distributions of the maximum utilization with various degrees of skew in the client-to-server assignment. In these sets of simulations, there were no hot-spot references. In general, the more skewed the assignment is, the more unbalanced the system becomes, with or without replication. But, a 10% replication is very effective in reducing load imbalances caused by skews in the client-to-server assignment. For instance, for the 9-6-1996 trace, the probability of the maximum utilization being less than or equal



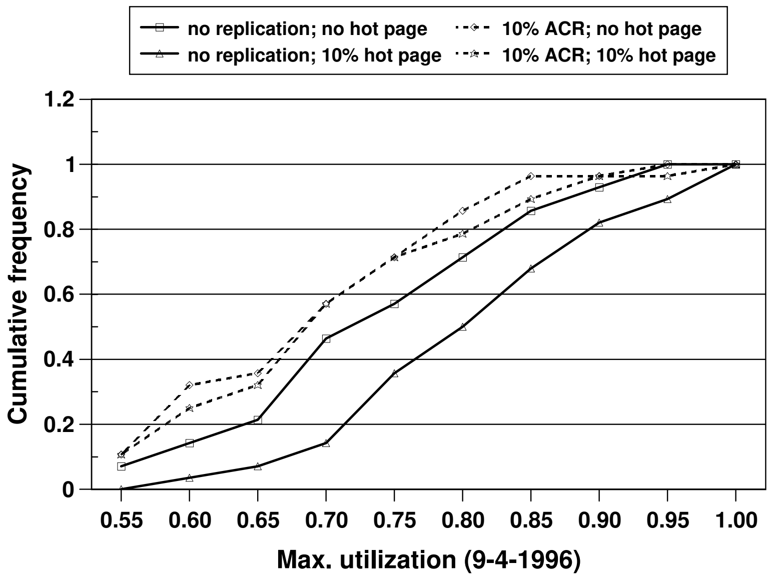


Figure 8: The impact of hot-spot references on load balance.

to 90% drops from 0.429 to 0.250 when the client-to-server assignment follows a Zipf(0.5, 8) distribution. However, with a 10% replication, this probability improves from 0.250 to 0.536. In other words, there is a 75% (i.e., $1 - 0.25$) chance that at least one proxy server is over 90% utilized if no replication is employed, but such a chance reduces to 46.4% (i.e., $1 - 0.536$) if a 10% replication is employed.

4.5 The impact of cache size

Finally, we examine the impacts of different cache sizes on the effectiveness of adaptable controlled replication. Fig. 10 shows the distributions of the maximum utilization for three different cache sizes, 10Mb, 70Mb, and 350Mb. For these simulations, there were no skew in the client-to-server assignment and no hot-spot references. Without replication, just increasing the cache size would not alleviate the load imbalances among the proxy cache servers. This is due to the fact that the load on the most utilized proxy cache is mostly forwarded to it from other fellow proxy caches. Hence, increasing the cache size would not reduce the number of requests being forwarded to the same most loaded proxy cache. However, with a 10% replication, better improvements can be achieved with a larger cache size. With replication, the number of requests being forwarded can be reduced as the cache size increases. As a result, the load on the most utilized proxy can be further reduced. Moreover, even if the cache size is very small, such as 10Mb (or about 1% of the *MaxNeeded*), adaptable controlled replication can have a noticeable improvement on load balance among proxy caches using hash routing.

5 Summary

In this paper, we examined load balancing and hot-spot reference issues of hash routing among a collection of proxy web caches. Simple hash routing, such as CARP, maximizes the aggregate cache utilization by eliminating duplication of cache contents. But it can cause load imbalances in the presence of access skew. We proposed an implementation of adaptable controlled replication for non-assigned-partition objects in a hash routing protocol. We studied the effectiveness of the adaptable controlled replication through trace-driven simulations. The results showed that (1) access skew does exist and the load among a collection of proxy caches tends to be quite unbalanced in hash routing, even if clients are uniformly assigned to cache servers; (2) with a small amount of replication, significant improvements

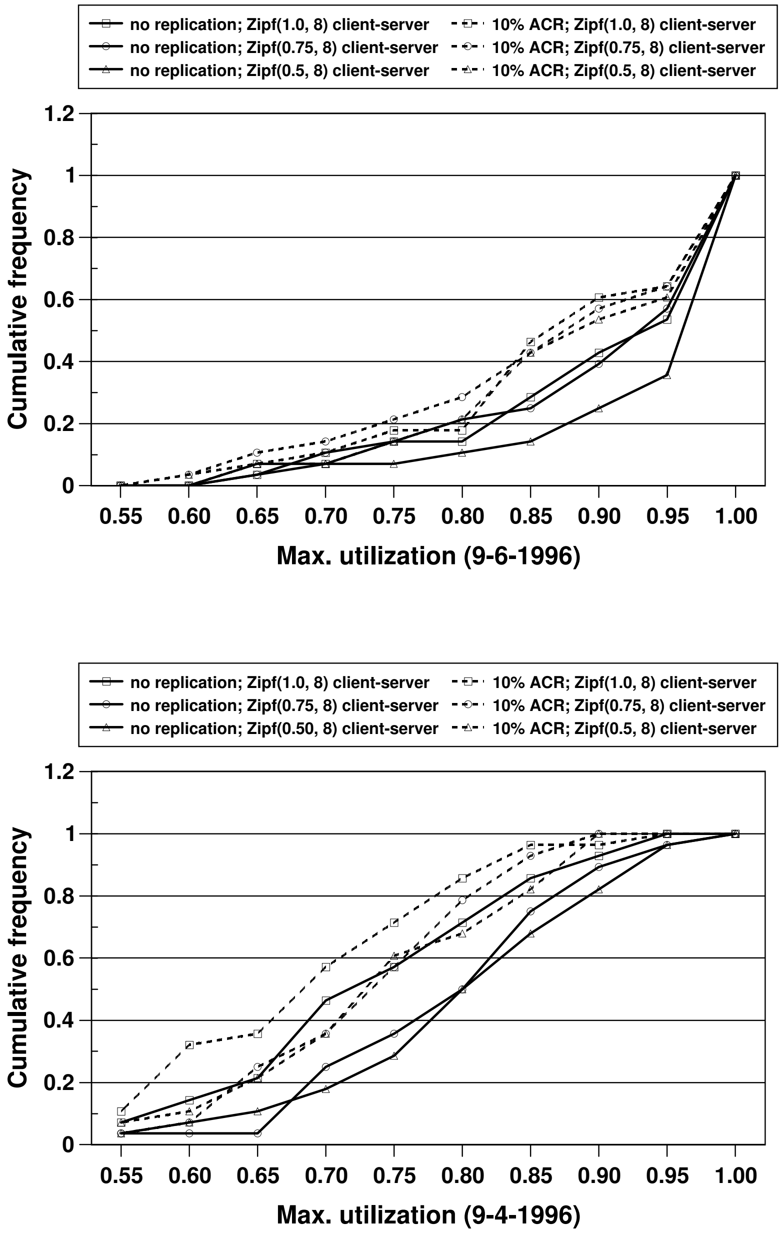
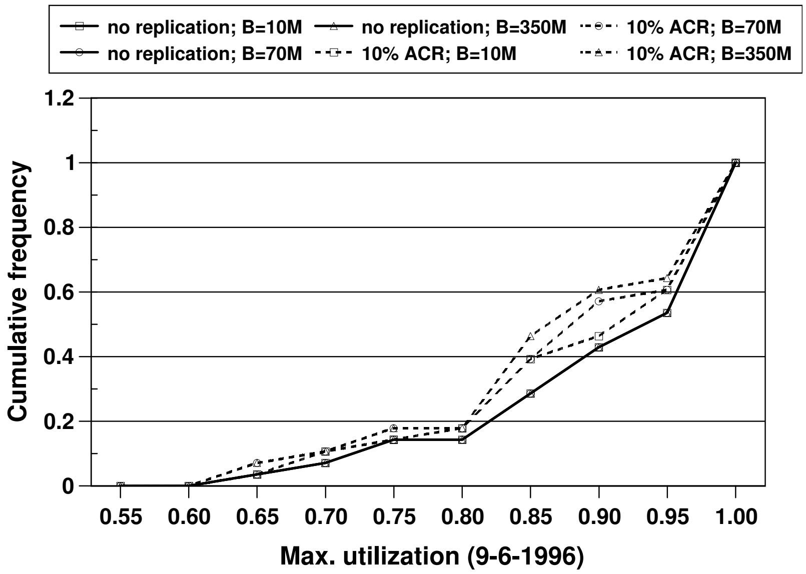


Figure 9: The impact of skew in client-to-server assignment on load balance.



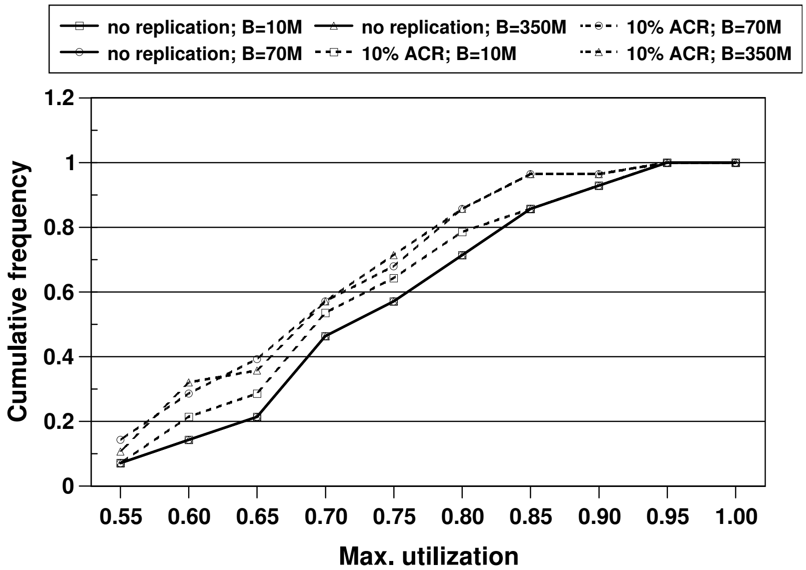


Figure 10: The impact of cache size on load balance.

in load balance can be achieved without degrading the total caching effectiveness; (3) adaptable controlled replication is very effective in reducing load imbalances caused by skew in the client-to-server assignment and hot-spot references; and (4) increasing the cache size does not improve load balance unless replication is allowed.

References

[1] M. Abrams et al. Caching proxies: Limitations and potentials. In *Proc. of 4th Int. World Wide Web Conference*, 1995.

[2] C. C. Aggarwal et al. On caching policies for web objects. Technical report, IBM T. J. Watson Research Center, 1996.

[3] C. M. Bowman et al. The Harvest information discovery and access system. In *Proc. of 2nd Int. World Wide Web Conference*, pages 763–771, 1994.

[4] A. Chankhunthod et al. A hierarchical internet object cache. In *Proc. of 1996 USENIX Technical Conference*, 1996.

[5] M. Colajanni, P. S. Yu, and D. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Trans. on Parallel and Distributed Systems*, 9(6):585–600, June 1998.

[6] M. D. Dahlin et al. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. of 1st Symp. on Operating Systems Design and Implementation*, 1994.

[7] D. M. Dias et al. A scalable and highly available web server. In *Proc. of IEEE COMPCON Conf. on Technologies for the Information Superhighway*, pages 85–92, Feb. 1996.

[8] G. Goldszmidt and G. Hunt. NetDispatcher: A TCP connection router. Technical Report RC 20853, IBM T. J. Watson Research, May 1997.

[9] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27:155–163, 1994.

[10] T. Kwan, R. McGrath, and D. A. Reed. NCSA's World Wide Web server: Design and performance. *IEEE Computer*, pages 68–74, 1995.

[11] A. Leff, J. L. Wolf, and P. S. Yu. Replication algorithms in a remote caching architecture. *IEEE Trans. on Parallel and Distributed Systems*, 4(11):1185–1204, Nov. 1993.

[12] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. on Computer Systems*, 7(4):321–359, Nov. 1989.

[13] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. In *Proc. of 4th Int. World Wide Web Conference*, 1995.

[14] Digital Equipment Corporation (merged with Compaq). Digital's web proxy traces. http://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html, 1996.

[15] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the sprite network file system. *ACM Trans. on Computer Systems*, 6(1):134–154, 1988.

[16] K. W. Ross. Hash-routing for collections of shared web caches. *IEEE Network Magazine*, pages 37–44, Nov.-Dec. 1997.

[17] Cisco System. Scaling the internet web servers. http://www.cisco.com/, Nov. 1997. White Paper.

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

[19] V. Valloppillil and K. W. Ross. Cache array routing protocol v1.0. Internet Draft, http://ircache.nlanr.net/Cache/ICP/draft-vinod-carp-v1-03.txt, Feb. 1998.

[20] D. Wessels. Squid internet object cache. http://squid.nlanr.net/Squid/, 1998.

[21] D. Wessels and K. Claffy. Internet cache protocol version 2. Internet Draft, http://ds.internic.net/internet-drafts/draft-wessels-icp-v2-00.txt.

[22] N. J. Yeager and R. E. McGrath. *Web Server Technology: The Advanced Guide for World Wide Web Information Providers.* Morgan Kaufman, 1996.

[23] P. S. Yu and E. A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. *Computer Networks and ISDN Systems*, 30:215–224, 1998.

[24] G. K. Zipf. *Human Behaviour and the Principles of Least Effort.* Addison-Wesley, Cambridge, MA, 1949.