February 7, 2001 TR CS-01-02

Maintaining Replicated Redirection Services in Web-based Information Systems

Koustuv Dasgupta[1] and Konstantinos Kalpakis[1,2]

Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, Maryland 21250

[1]Email: {kalpakis,dasgupta}@csee.umbc.edu

[2]Supported in part by the National Science Foundation under grant number IRI–9729495, and by NASA under Cooperative Agreement NCC5–315.

Maintaining Replicated Redirection Services in Web-based Information Systems

Koustuv Dasgupta[1] and Konstantinos Kalpakis[1,2]

February 7, 2001

Abstract

The growth in the commercial use of the Internet and the proliferation of new Web information services has led to a number of problems - overloaded servers, traffic congestion and increasing client latencies. Dynamic replication of information among geographically distributed Web servers is commonly used to address such problems of scale. A key challenge for such systems is to maintain an efficient and scalable *redirection service* that tracks the servers where documents are replicated, and assigns each client request to a server that can offer the "best" service.

We propose a new scheme for maintaining redirection services in Web-based information systems. The proposed scheme is based on adaptive replication of the entries of the location directory that provides the redirection service. Network-level support enables client requests to be redirected to appropriate servers in a quick and efficient manner. We integrate our scheme with a dynamic object replication protocol proposed in the literature. Simulations based on synthetic workloads and the Internet backbone map of CRL Network Services demonstrate that the replicated redirection service delivers significant improvements in terms of client response time, while successfully eliminating hot spots in the network and imposing minimal traffic overhead.

1 Introduction

The recent growth in the World Wide Web is rapidly moving us towards highly distributed and wholly interconnected information systems. The enormous potential for the commercial use of the Web and the consequent proliferation of new information services (e.g. Internet Hosting Services, E-Commerce sites, auction services, and Content Distribution Networks) have however led to a number of problems - overloaded servers, traffic congestion and increasing client latencies. As pointed out by Manley and Seltzer [13], the answer to the question "Why do users wait on the Web?" still eludes the research community.

Replication of information among multiple Web servers is commonly used to improve performance and support high request rates at popular Web sites. The new generation of Web architectures [1, 10, 15] rely on sophisticated algorithms to make replica placement decisions based on the access patterns of individual documents and load information from servers. Replica placement

[1]Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. E-mail: {kalpakis,dasgupta}@csee.umbc.edu. Phone: 410-455-3143. Fax: 410-455-3969. **Please send all correspondence to Prof. K. Kalpakis.**

[2]Supported in part by NSF under grant IRI–9729495, and by NASA under contract NCC5–315.

decisions are taken autonomously by participating servers and replicas of documents are created or deleted dynamically in geographically distributed locations. A key challenge for such large-scale systems is to design an efficient *redirection service* (also known as *directory service*) that tracks the servers where data objects are replicated and assigns each client request to a server that can offer the "best" service. Server selection algorithms include criteria like network topology, server availability and server load [5]. Clearly, the ability to locate replicas quickly and perform request distribution has critical implications for the user-perceived response time.

Various schemes have been proposed in the literature that perform redirection at the IP level through some address packet rewriting scheme [7], at the DNS level through mapping of URL-name to IP-address of one of the servers in a cluster [2], or at the application level [4] using redirection features of the HTTP protocol. None of these schemes, however, address dynamic changes in the replica locations of individual documents. Relais [12] tackles the problem by using a location directory that stores the contents of all caching servers, and replicating the entire directory at the entry points for the clients. Such a solution suffers from scalability problems when deployed on an Internet-scale system. In the same context, the Summary Cache protocol [8] requires that remote caches periodically exchange their contents via broadcast or multicast. Though Bloom filters [8] provide a compact representation of cache contents, the location information grows linearly with the number of caches times the number of objects in each cache, leading to significant message overhead.

Among other recent works, the CRISP [9] architecture uses a centralized or partitioned global directory that provides the redirection service. A couple of CRISP variants that asynchronously replicate the global directory (partially or in its entirety) across a set of mapping servers are also proposed. Zhang et al [18] suggests replacing directories with multicast queries to nearby caches. An alternative scheme by Tewari et al [16] uses a scalable hierarchy of hint caches to locate and access remote data objects and provide push-based replication. Another novel approach by Legedza and Guttag [11] uses network-level support to reduce the time needed to locate relatively unpopular documents in the server caches, as well as minimize latency for searching documents that are not cached.

This paper examines a new approach for maintaining redirection services in Web-based information systems. The proposed scheme aims at minimizing client response time and is based on selective, asynchronous replication of the location directory that provides the redirection service. We assume that the directory entry for each document contains a list of replica locations for the document, along with some usage statistics for each replica. We first describe an *adaptive* protocol that replicates directory entries across multiple sites (called *redirectors*) based on the access patterns of the individual entries. The premise of replication at a per-entry level lies in numerous findings that confirm the highly uneven popularity of Web documents [6]. Note that, in order to execute the request distribution algorithm, each replica of a directory entry (called *entry replica*) needs to be notified of periodic changes in the replica locations for the corresponding document. We describe a weak consistency protocol that propagates these changes, as background messages, to the different entry replicas. Finally, we enhance the scheme with an Active Network[11]–style *service discovery* protocol that allows replicated redirectors to intercept client requests at the network (router) level, and perform fast lookup and redirection. A novelty of our approach is that it does not depend upon the document replication or server selection algorithm that is deployed. Specifically, we have integrated our scheme with a dynamic object replication and migration protocol proposed by Rabinovich et al [15]. Simulations based on the Internet backbone map [3] of CRL

Network Services, a leading Internet Hosting Service, and a combination of synthetic workloads demonstrate that the replicated redirection service delivers significant improvements in terms of client response times, while successfully eliminating hot spots in the network and imposing minimal backbone traffic overhead.

The rest of the paper is organized as follows. In Section 2 we provide an overview of the system model and a brief discussion on Rabinovich et al's protocol [15] with primary focus on the redirection service. Section 3 describes the design of the replicated redirection service, along with some implementation details. In Section 4 we evaluate the performance of the scheme, and finally in Section 5 we conclude the paper.

2 Background

In this paper, we consider a Web-based information system consisting of a group of (geographically) distributed servers which maintain and provide access to a set of Web documents. The system model derives from one proposed by Rabinovich et al [14] to develop a dynamic replication and request distribution protocol for Internet objects. This section provides an overview of the model as well as a brief description of the protocol presented in [14, 15].

2.1 System Model

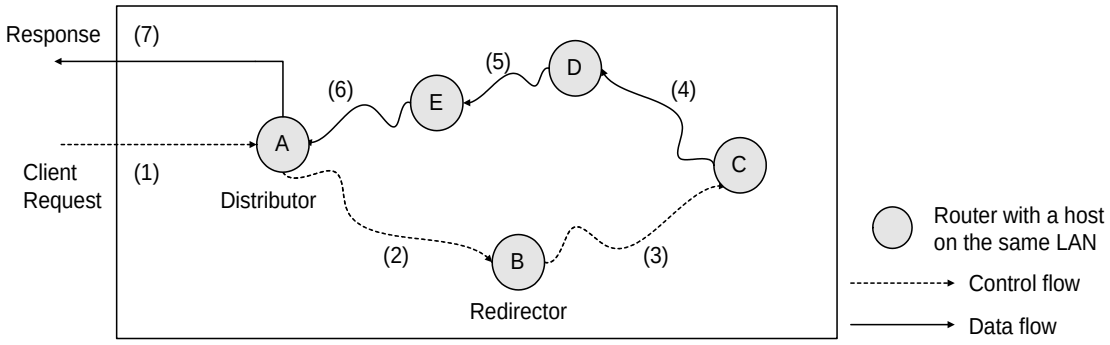


Figure 1: System Architecture.

The architecture described in [14] considers an Internet Hosting Service that provides access to a set of Web objects. It contains a set of *nodes*, each consisting of a router and a *hosting server* (or *host*), connected to each other by a LAN. Hosting servers maintain the data objects, while routers are connected via wide-area links to form the network backbone. External clients enter the system through a designated set of *gateway nodes*. These nodes contain *distributors* in addition to routers and hosts. A request from a client is directed to the "closest" gateway's distributor regardless of the requested document.[3] The distributor forwards this request (via a UDP message) to a centralized *redirector* which chooses the best host among those that currently have replicas of the requested object, and forwards the request (as a UDP message) to the host. Finally, the host responds with the data object directly to the distributor which passes it over to the client.

[3]The authors [15] suggest that existing DNS infrastructure can be leveraged to provide the mapping from the host name in the document URL to a distributor's IP address.

2.2 Rabinovich's Protocol for Dynamic Replication and Request Distribution

Rabinovich et al [15] proposed a protocol that integrates an algorithm for dynamically deciding the number and location of document replicas in the system along with an algorithm for distributing requests among the replicas.

Each host in the system runs a replica placement algorithm periodically which attempts to replicate the popular documents at servers that are closer to the clients. Dynamic placement decisions are taken based on replication, migration and deletion thresholds for each document, as well as low and high load thresholds for each server.

Redirect Request(Document x, Client c)
1 min-RD-count = $\min_i\{\text{RD-count}(x_i)\}$
2 d_{max} = distance in hops of the farthest replica from c
3 Loop through replicas in increasing order of distance from client c
4 foreach replica x_i do
5 let $d(x_i)$ = distance in hops of the replica x_i from c
6 let $rank(x_i) = d_{max} - d(x_i) + 1$
7 if RD-count$(x_i) \leq$ min-RD-count $\times$ rank(x_i)
8 Send request to i
9 RD-count(x_i) = RD-count(x_i) + 1
/* The least loaded replica trivially satisfies the above condition. */

Figure 2: Request Distribution Algorithm for document x.

The redirector is responsible for distributing client requests among the various replicas. Let x be a document and $x_1, x_2, \cdots, x_k$ be the replicas of x located at the hosts $s_1, s_2, \cdots, s_k$ respectively. We call $s_1, s_2, \cdots, s_k$ to be the *replica set* for x. Let the *request count* for x_i, i.e. RD-count(x_i), be the number of client requests that the redirector forwarded to the replica x_i. For each document x in the system, the redirector has a directory entry $RD(x)$. This entry stores the location of each replica x_i along with RD-count(x_i). Replica selection decisions are based on client-server proximity as well as request counts for each replica. Figure 2 gives a high level description of the algorithm.

Experimental results in [15] show that the algorithm succeeds in significant reduction of backbone traffic and elimination of hot spots by distributing the load among the various servers. However, the reduction in terms of average client response time is far less spectacular since every client request needs to take a *detour* to the global redirector. While the authors in [15] suggest hash-partitioning the URL-name space among multiple redirectors to distribute load and increase availability, the scheme does not help in terms of minimizing request latency. This is because each document still has a single directory entry in the system. Consequently, we examine an approach where the directory entries are selectively replicated at multiple nodes, potentially those closer to the gateways, which are capable of executing the request distribution algorithm. Note that a naive policy replicates each of the directory entries at all the redirectors. However, this would require every server to have enough storage capacity and further lead to significant increase in backbone traffic due to redirector messages that update the different replicas of the directory entries. Our strategy aims at replicating directory entries for the popular documents at redirectors closer to the clients, while taking into account the cost that will be incurred by the system to update the entry replicas.

3 Maintaining Replicated Redirection Services

In the proposed system, each node has a hosting server and a *redirector*, both connected to a backbone router on the same LAN. Initially, the directory entries for the documents are distributed among all the nodes by hash-partitioning the URL-name space. Thus, each document x initially has a unique directory entry stored at exactly one of the nodes. We call this entry the *primary directory entry* for x, and the corresponding node the *primary redirector* for x. A redirector is capable of executing the request distribution algorithm for each of the directory entries that it stores. We assume that the location of the primary redirector for each document and the location of the gateways, i.e. entry points for the clients, are available to all the nodes in the system.

Redirector messages are assumed to be of two types. A *REDIRECT* message is sent by a gateway's distributor to the redirector of the document requested by the client, and an *UPDATE* message is sent by a host s that creates or drops a replica of a document x to the redirector for x. An *UPDATE* message can also carry replica usage information from a host to a redirector. In the context of the replication protocol, we treat a *REDIRECT* request as a logical *read* operation and an *UPDATE* request as a logical *write* operation on the directory entry. For now we assume that reads and writes are performed on the closest replica of the directory entry. Section 3.2 describes the detailed mechanism. Moreover, writes are propagated to all replicas of the directory entry using a certain spanning tree of the nodes that have a replica of the directory entry.

3.1 The Adaptive Replication Protocol

The replication strategy is based on the Adaptive Data Replication (ADR) protocol proposed by Wolfson et al [17]. It is executed periodically by a redirector for each of the directory entries that it stores. Intuitively, the replication scheme $R(x)$ for a directory entry spans over a set of nodes (redirectors), remains connected at all times, and expands as the read activity increases and contracts as the write activity increases. Consider a client c located at a distributor D that requests a document x. Let r be the redirector for x. Let $D \rightarrow r_1, \cdots, \rightarrow r_n \rightarrow r$ be the internal router path taken by the *REDIRECT* request from D to r. The *preference path* between D and r is a sequence of redirectors co-located with the corresponding routers. Since *REDIRECT* requests are implemented as UDP messages, the preference path for each request can be efficiently extracted at the destination redirector using the IP header from the message and the *BGP* routing database [15].

Each redirector r maintains the following information for each directory entry $RD(x)$ it stores. For each neighboring redirector p that has appeared on the preference paths of some *REDIRECT* requests since the last execution of the replication protocol, r keeps count of the number of such appearances, referred to as the *read count* of p. For each neighboring redirector p, r also keeps count of the number of *UPDATE* requests it forwarded to p during the last period i.e. the *write count* of p.

We denote a *border* redirector as one that belongs to $R(x)$ and has at least one neighboring redirector that is not in $R(x)$. Each border redirector i executes the following test, at the end of each period, for each neighbor j that is not in $R(x)$:

Expansion Test: If the number of reads that i received from j in the last period is greater than k [4] times the total number of writes that i received from itself or from a neighbor different

[4] k is the adjustment factor chosen to be 10.

than j, then it is potentially beneficial to keep a replica of the directory entry $RD(x)$ at j. Thus, j joins $R(x)$. In the actual implementation, redirector i sends a "join" request to its neighbor j along with a copy of the directory entry $RD(x)$.

A *fringe* redirector is one that belongs to $R(x)$ and has exactly one neighboring redirector in $R(x)$. Note that a redirector i can be both a border and a fringe redirector. Let j be the neighbor of i in $R(x)$. Then i first executes the expansion test and on failing performs the following test:

Contraction Test: If the number of writes that i received from j during the last period is greater than k times the total number of reads that i received from itself or from a neighbor different than j, the benefit of keeping a replica of $RD(x)$ at i is outweighed by the cost of propagating all the updates to this replica. Thus, i informs j of its decision to exit from $R(x)$. Note that the primary redirector for a document x is never allowed to drop the directory entry for x.

3.2 The Service Discovery Protocol

In an environment where directory entries are replicated across multiple sites, a key challenge lies in searching the network for a redirector that stores the directory entry queried by a distributor (in case of a $REDIRECT$ request) or a host (in case of an $UPDATE$ request). We tackle this problem by integrating the routing of redirector requests with the routing services already provided in the network. The proposed scheme is motivated by recent works on Active Networks [11] in the context of cache routing.

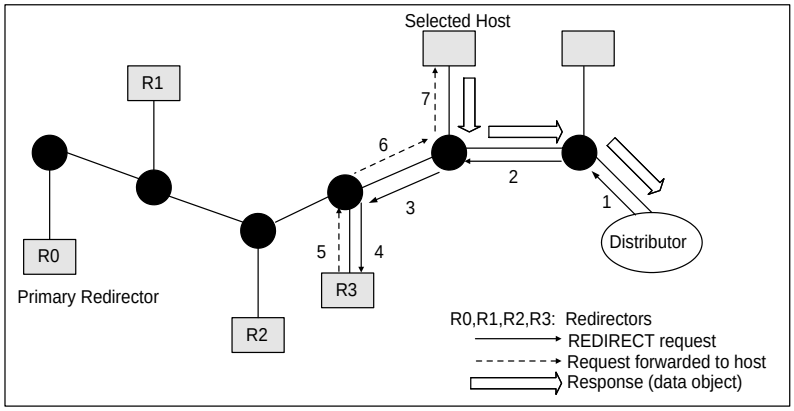


Figure 3: Illustration of the Service Discovery protocol.

In our approach, a $REDIRECT$ request for a document x travels from a distributor D towards the primary redirector for x ($R0$ in Figure 3) along a certain preference path. Note that this preference path consists of a sequence of redirectors (co-located with the border routers) that might potentially store replicas of the directory entry being queried. Each intermediate router maintains a URL-indexed table of documents for which the local redirector has directory entries; then for each $REDIRECT$ request *en route* to the primary redirector, the router performs a table look-up to check for a matching URL. In case of a *hit*, the $REDIRECT$ is simply send to the local redirector ($R3$ in Figure 3) that in turn forwards the request to an appropriate host. This scheme has two advantages : (1) it reduces the number of hops traveled by the $REDIRECT$ message from the distributor to a redirector, and (2) it reduces the total number of hops traveled by the request

from the distributor to a host. A $REDIRECT$ for a directory entry that is not replicated at any of the intermediate redirectors still needs to travel all the way to the primary redirector. Keeping routing decisions at the network level ensures reduced overhead due to hit/miss processing time. Indexing and look-ups on the router table by URL's can be made efficient by using MD5 hash codes of the URL strings [11]. The size of each router's table is controlled by the fact that only documents that have (directory) entry replicas with the local redirector have entries in the table. Also, techniques based on Bloom filters [18] can be used to make the table compact.

An $UPDATE$ request is sent from a host s to the primary directory entry for x and can carry three possible types of notifications. A *create* notification informs the directory entry about a new replica of x created at host s, a *drop* notification informs the directory entry about an existing replica of x that has been dropped from host s, and an *affinity* notification informs the directory entry about changes in replica usage statistics (for x) at host s. Note that $UPDATE$ notifications occur when a hosting server executes the replica placement algorithm [15]. Moreover, the changes are propagated asynchronously to all the entry replicas using background (UDP) messages. Consider an $UPDATE$ message for a directory entry $RD(x)$. Let $R(x)$ be the set of redirectors that have a replica for this entry. If the $UPDATE$ carries a *create* or *affinity* notification to the primary redirector, it is intercepted (as in case of a $REDIRECT$) by an intermediate redirector r that has a replica for $RD(x)$. The redirector r then forwards the message to each of its neighboring redirectors in $R(x)$. A neighboring redirector in turn forwards the $UPDATE$ to all neighbors other than r in $R(x)$, and so on until the notification reaches all the entry replicas. In case there are no intermediate redirectors with replicas of $RD(x)$, the $UPDATE$ reaches the primary redirector from where it is propagated to any other entry replicas. For a *drop* notification, however, we enforce that each $UPDATE$ goes first to the primary directory entry, without being intercepted at any intermediate redirector, and then it is propagated to all entry replicas. This guarantees that there is at least one copy of every document at any time in the system. Finally, a timestamp-based mechanism ensures that the operations resulting from the background messages interleave correctly, i.e. according to a causal specification, at the different entry replicas. Details are omitted due to lack of space.

4 Performance Evaluation

We have developed a JAVA-based event-driven simulator for studying and evaluating our protocols. For the current experiments, the underlying network topology is based on the Internet backbone map of CRL Network Services [3], a leading Internet Hosting Service. The backbone contains 33 nodes. Each node has a hosting server and a redirector connected to a backbone router on the same LAN. There are 1000 documents in the system that are initially distributed among the nodes in a round-robin fashion. We denote the node that initially hosts a document as the *home server* for the document. The directory entries for the documents are distributed among the redirectors by hash-partitioning the document-name space. Each node also serves as a gateway, i.e. entry point, for a client. The inter-arrival time between client requests is assumed to be exponentially distributed with a mean of 0.02 seconds. Table 1 provides the parameters used in our experiments.

The experimental results provided are based on the following synthetic workloads.

- **Zipf:** Previous studies [6] have shown that the reference probability for documents requested by a client, as well as documents hosted on a Web site, follow Zipf's Law. The law states that if documents are ranked according to their access frequencies, then the reference probability

Table 1: Simulation Parameters.

Parameter	**Value**
Size of Document	10KB
Size of Message	256 bytes
Link Bandwidth	100Mbps
Network Delay	10ms per hop
Low Watermark for a host	40 or 80 requests/sec
High Watermark for a host	50 or 90 requests/sec
Deletion and Replication Thresholds for Individual Documents	0.03 and 0.18 requests/sec
Placement Decision frequency for redirectors	every 100 seconds
Placement Decision frequency for hosts	every 50 seconds
Load Measurement Interval	every 10 seconds

for a document with rank i is proportional to $1/i$. The Zipf workload thus models the situation where popularity of documents vary in a highly uneven manner. A client requests documents according to Zipf's law, where the document number corresponds to its rank.

- **Hot Sites:** This workload models the situation where certain Web sites (and not individual documents) are more popular than others. Specifically, the nodes are divided into hot and cold sites, with 10% of the nodes being randomly labeled as hot. A client requests a random document from a hot site with probability 0.9, and a random document from a cold site with probability 0.1.

For each workload, we consider the following two schemes : (A) multiple redirectors with no replication of directory entries and (B) multiple redirectors with replicated redirection service as proposed in this paper. The replica placement algorithm for each document and the request distribution algorithm executed by each redirector are identical to Rabinovich et al's [15] protocol. For each of the schemes, we further simulate *low load* and a *high load* conditions by using low (lw) and high (hw) watermarks or thresholds, for each hosting server. The watermarks along with the inter-arrival time for client requests determine the maximum load that a host can accept. Specifically, for an average request rate of 50 req/sec, we simulate low load with $lw = 80$, $hw = 90$ and high load with $lw = 40$, $hw = 50$ req/sec. Note that Rabinovich's replica placement algorithm [15] takes into account the load imposed by each document on its host. Thus, setting smaller values for the watermarks (in case of high load) implies that servers which have their loads closer to the maximum thresholds are unable to replicate many of the popular documents even when it is beneficial in terms of client-server proximity. This in turn enables us to evaluate the performance of the two schemes under stressful conditions.

Fig. 4(a) shows the improvement in average response time for the Zipf workload under high load conditions. We see that scheme B leads to a significant reduction in the average client response time (about 46.3%) over scheme A. This improvement can be better understood from figs.4(b)-4(c). For each client request, we measure the number of client-to-server hops (via the redirector) traveled by the request message, and the number of server-to-client hops traveled by the response message. When there is no replication of directory entries, each client request needs to travel a fixed number of hops to reach the corresponding directory entry, before getting redirected to the "best" server. This explains the almost constant number of client-to-server hops in fig. 4(b) for scheme A.

However, in scheme B, the frequently accessed entries are gradually replicated closer to the clients and more $REDIRECT$ requests get intercepted by intermediate redirectors with entry replicas. The average number of client-to-server hops thus reduce by about 50.2% compared to scheme A. Fig. 4(c) shows the results for the average number of server-to-client hops. Since Rabinovich et al's [15] replica placement algorithm aims at reducing the distance traveled by the response message, it is quite expected that client-server proximity increases in scheme A as documents are replicated closer to clients. The more interesting observation is that the gains are even more using scheme B.

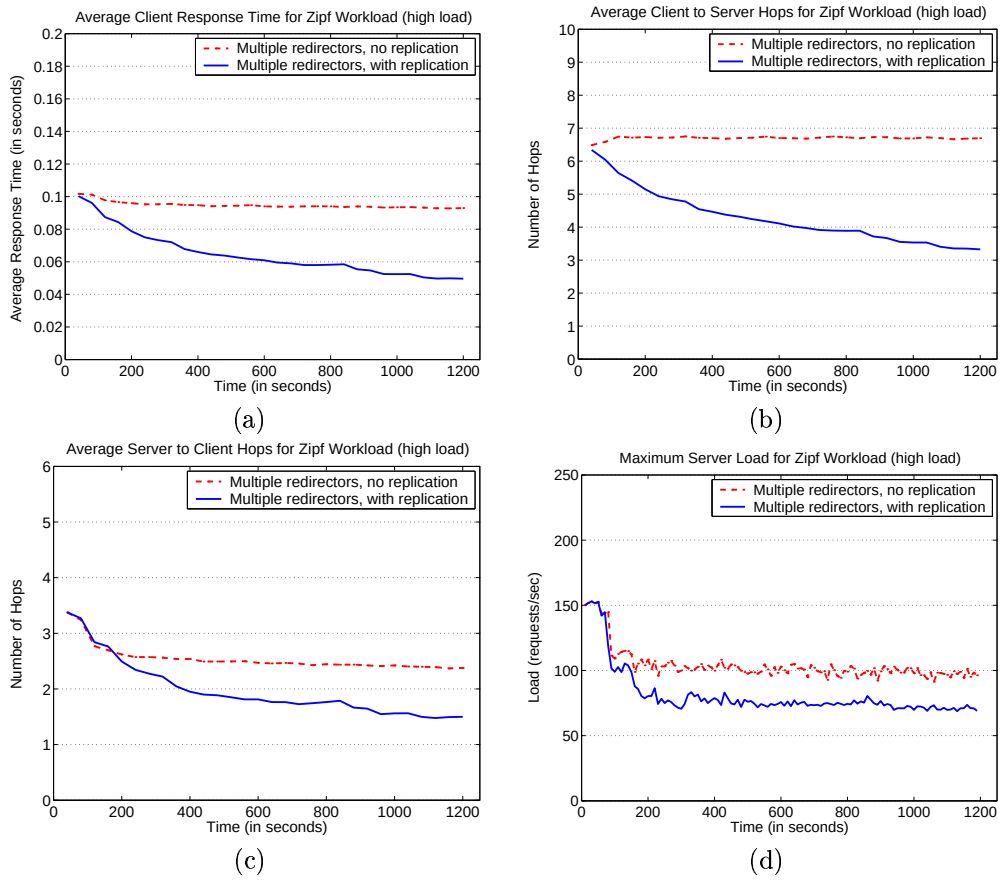
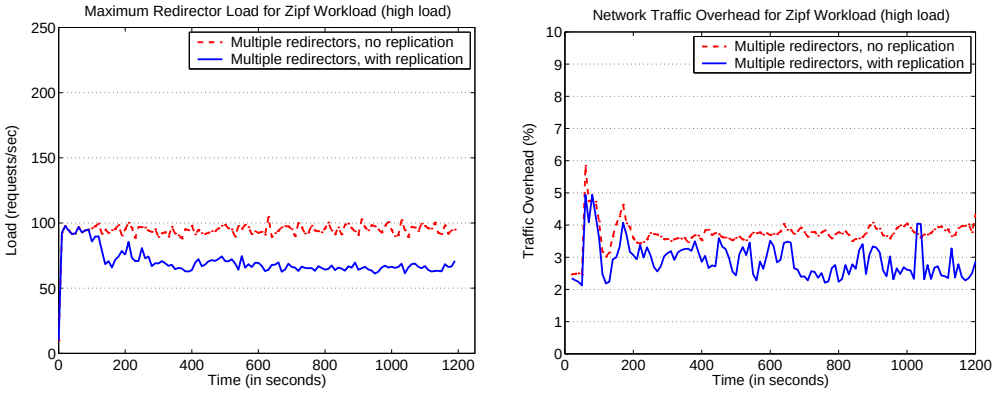


Figure 4: Performance Measures for Zipf workload (High Load).

Consider fig. 4(c) once again. We see that the average number of server-to-client hops in scheme B is about 36% lower than scheme A. This can be attributed to the server selection procedure at each redirector; specifically, with more entry replicas, the number and rate at which $REDIRECT$ requests arrive at each individual directory entry reduce significantly, and the request distribution algorithm redirects more requests towards servers that are closer to clients. The combined effect of figs. 4(b) and (c) is that the average number of round-trip hops and hence the average response time for client requests improve significantly.

A positive effect of request distribution by multiple entry replicas is also seen in the load imposed on the hosting servers. In fig. 4(d) we show the maximum load (requests serviced per second) for any server. Note that a decrease in the maximum server load signifies successful elimination of hot spots in the system. Moreover, when the system load is high, the replica placement algorithm is able to relocate a fewer number of documents to hosts that are closer to clients. Consequently, the

home servers for the most popular documents continue to remain hot spots. We observe that under high load conditions scheme B delivers better performance in terms of eliminating these hot spots. The improvement is quite appreciable (about 22.6%) compared to scheme A. While not shown in the figures, we observed that the average number of replicas per document for the two schemes were almost identical (close to 1.8 replicas per document). This signifies that the improvement was once again due to the server selection procedure, which in the case of scheme B, redirected more requests to hosts that are closer to clients, thereby reducing the burden on the overloaded home servers and hence the maximum server load.



(a) (b)

Figure 5: More Performance Measures for Zipf workload (High Load).

Fig. 5(a) shows the maximum load on any of the redirectors in the system. It is measured in terms of the number of $REDIRECT$ requests serviced per second. The results show a 28.7% reduction in the maximum redirector load for scheme B over scheme A. A better load distribution is achieved in scheme B, since $REDIRECT$ requests to the frequently accessed directory entries can be satisfied by intermediate redirectors which store entry replicas, thus alleviating the load on the primary redirectors.

Fig. 5(b) shows the network traffic overhead for the two schemes. This overhead is due to (1) replication and migration of documents, (2) control messages generated by the replica placement algorithm, (3) $REDIRECT$ and $UPDATE$ messages and (4) replication of the directory entries. Note that scheme B introduces additional message transfers due to replication of directory entries and $UPDATE$ notifications to the entry replicas. However, as shown in fig. 4(f), the traffic overhead (measured as a percentage of total backbone traffic) for scheme B is mostly lower than scheme A. We attribute this to (i) the fewer number of hops traveled by each $REDIRECT$ message and (ii) the fact that entry replicas are created selectively and infrequently (every 100 seconds). We observed that the average number of replicas per directory entry was as low as 2.2 for the Zipf workload. Moreover, the size of a directory entry involved in each transfer is quite small[5]. Finally, the overhead of propagating $UPDATE$ notifications remains minimal since changes in the replica set (or usage) of any document occur at infrequent intervals i.e. each time a server runs the replica placement algorithm. The above numbers also suggest that the redirectors have enough storage capacity for entry replicas, so that the replication scheme for each directory entry remains connected at all times.

[5]For a system with 33 hosts, the estimated size of a directory entry is at most 33 × (128-bits for IPv6 address + 4 bytes for request count + 4 bytes for replica usage information) = 792 bytes.

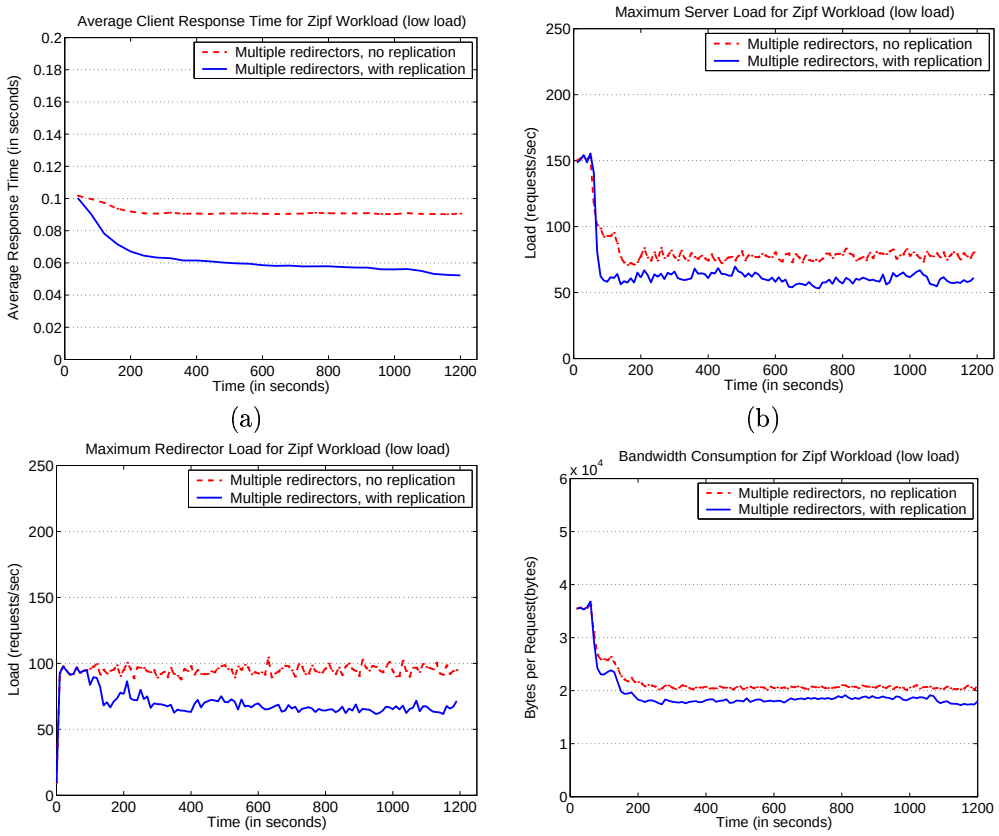


Figure 6: Performance Measures for Zipf workload (Low Load).

Figures 6 and 7 depict some of the performance measures for the two schemes using the Zipf workload (under low load condition) and the Hot Sites workload (under high load condition). The results show similar trends in improvement, for average client response time and maximum server load, using scheme B. The replicated redirection service, in scheme B, does especially well in eliminating hot spots for the Hot Sites workload, as shown in fig. 7(b). Figs. 6(c) and 7(c) show the maximum load on any redirector for each of the workloads. We observe that for the Hot Sites Workload, the maximum redirector load for scheme B is initially higher comapred to scheme A. This can be attributed to the loads on the primary redirectors that increase due to the replication of popular directory entries. However, as the simulation progresses, the replication scheme for each directory entry gradually moves towards the optimal scheme, and the maximum redirector load in the system falls below A. We also show the total bandwidth consumption, i.e. the sum of the total number of bytes transmitted on each network hop, for both of these workloads. Figs. 6(d) and 7(d) demonstrate that the reduction in backbone traffic (per client request) ranges from 15.8% for the Zipf workload to about 40% for the Hot Sites workload. The significant improvement in bandwidth consumption points to the fact that the replicated redirection service helps in minimizing the number of hops traveled by the client request as well as the response message.

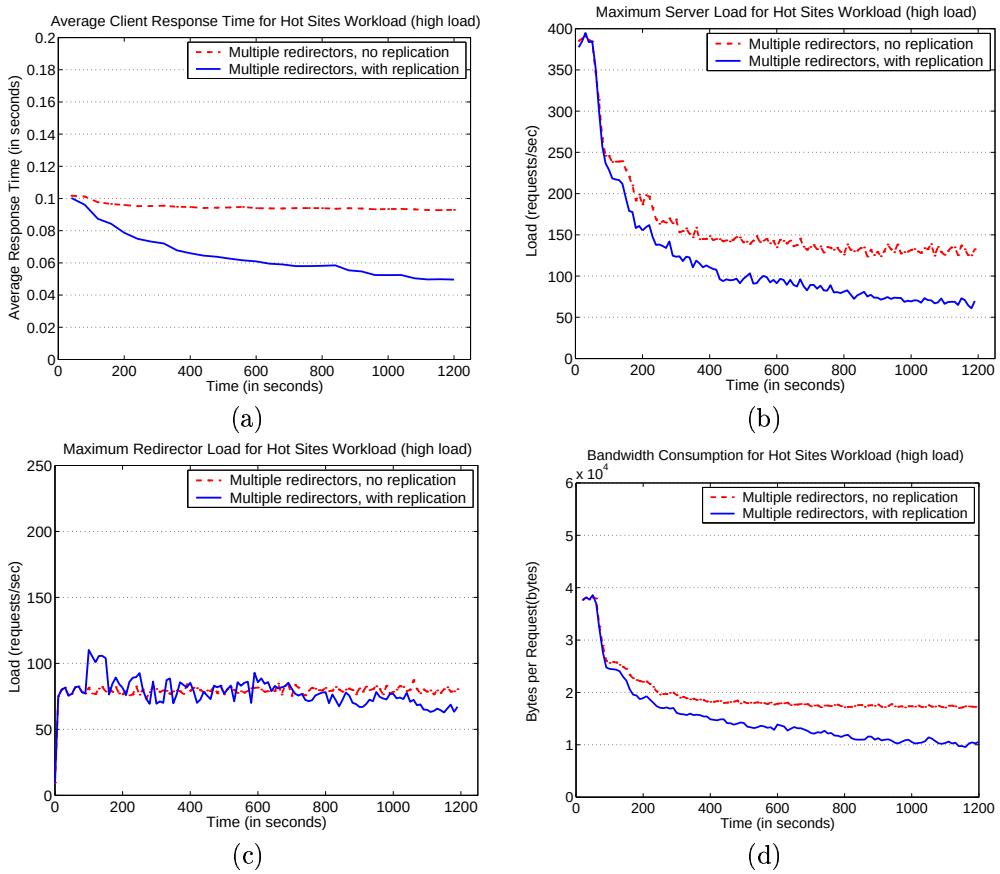


Figure 7: Performance Measures for Hot Sites workload (High Load).

5 Conclusions

The design of an efficient and scalable redirection service is a critical aspect for the performance of current Web-based information systems. We proposed a novel approach for maintaining redirection services in such systems. Our scheme is based on adaptive replication of the contents of the location directory that provides the redirection service at a per-entry level. We use network-level support to design a service discovery protocol that allows client requests to be redirected to the appropriate hosting servers in a quick and efficient manner. The replicas of the directory entries are updated using background messages. Our technique demonstrates how emergent networking technologies can contribute to the current state-of-the-art for Internet systems. The proposed scheme has been integrated with Rabinovich et al's [15] protocol for dynamic object replication and request distribution for Internet Hosting Services. Simulation results based on synthetic workloads show that the replicated redirection service delivers significant reduction of average client response time, and is also more successful in eliminating hot spots and reducing backbone traffic. In the future, we plan to evaluate the performance of the scheme using real Web traces.

References

[1] Akamai. *http://www.akamai.com.*

[2] Cisco's Distributed Director. *http://www.cisco.com/warp/public/751/distdir.*

[3] Internet Backbone Map Of CRL Network Services. *http://www.nthelp.com/images/crl.jpg.*

[4] WindDance Networks Corporation. *http://www.winddancenet.com.*

[5] R. L. Carter and M. E. Crovella. On the Network Impact of Dynamic Server Selection. *Computer Networks*, 31(23-24):2529–2558, 1999.

[6] C. Cunha, A. Bestavros, and Mark Crovella. Characteristics of WWW Client-based Traces. Technical report, TR-95-010, Boston University, April 1995.

[7] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web Server. In *Proceedings of 41st IEEE Computer Society International Conference*, Feb 1996.

[8] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *Proceedings of ACM SIGCOMM Conference*, 1998.

[9] S. Gadde, J. Chase, and M. Rabinovich. Directory Structures for Scalable Internet Caches, TR CS-1997-18. Technical report, Duke University, Department of Computer Science, November 1997.

[10] A. Heddaya and S. Mirdad. WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. In *Proc. of 17th IEEE Conf. on Distributed Computing Systems*, May 1997.

[11] U. Legedza and J. Guttag. Using Network-level Support to Improve Cache Routing. In *Proceedings of the 3rd International WWW Caching Workshop*, June 1998.

[12] M. Makpangou, G. Pierre, C. Khoury, and N. Dorta. Replicated Directory Service for Weakly Consistent Replicated Caches. In *Proc. of the 19th IEEE Conf. on Distributed Computing Systems*, May 1999.

[13] S. Manley and M. Seltzer. Web Facts and Fantasy. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, December 1997.

[14] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic Replication on the Internet. Technical report, ATT Labs Technical Report HA6177000-98305-01-TM, March 1998.

[15] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service. In *Proc. of the IEEE Conf. on Distributed Computing Systems*, May 1999.

[16] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *Proc. of the 19th IEEE Conf. on Distributed Computing Systems*, May 1999.

[17] O. Wolfson, S. Jajodia, and Yixiu Huang. An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems*, 22(4):255–314, 1997.

[18] L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web Caching. In *Proceedings of the NLANR Web Cache Workshop*, June 1997.