Performance of Dynamic Replication Schemes for an Internet Hosting Service

Amit Aggarwal
University of Washington
Seattle, WA
amit@cs.washington.edu

Michael Rabinovich
AT&T Labs – Research
Florham Park, NJ 07932
misha@research.att.com

Abstract—

The paper explores schemes for *dynamic replication and migration* of web objects in the context of an Internet hosting service. It describes a replica placement algorithm for deciding the location and number of replicas of an object as well as request distribution schemes for choosing among currently available replicas. We compare two classes of request distribution algorithms – namely feedback and non-feedback based. Further, we compare dynamic replication to a static replication scheme. We have simulated the algorithms using synthetic workloads as well as a real trace from a hosting service. Measurement and analysis show that dynamic replication significantly reduces bandwidth consumption and latency, removes hot spots from the network and smooths out bursts in bandwidth demand while imposing only a low network traffic overhead. For example, on the trace, our algorithm reduces bandwidth consumption by as much as 52% while imposing a traffic overhead of only about 5%. Results of the simulation also demonstrate that at high loads, the ability to predict the effect of a migration or replication is important for any system that takes dynamic replica placement decisions.

I. Introduction

The increasing popularity of the world wide web has resulted in a number of problems - overloaded servers, traffic congestion and increasing client latencies. These problems are bound to grow as the explosive growth of the Internet continues. Replication, or mirroring, of web servers is commonly used to address these problems of scale. Distributing requests among replicas is an interesting issue, especially since it involves the two, often conflicting, goals of distributing load and increasing client-server proximity. A number of sites in the Internet are mirrored and a number of schemes like DNS round robin [6], Distributed Director [9] and others [7], [15], [28] address the issue of request distribution. However, the mirroring is done statically and most of the request distribution schemes are designed for a static set of replicas.

We believe that static replication is not a very feasible solution. Firstly, statically deciding what and where to replicate based on access patterns is difficult and error-prone. This task is especially difficult (and important) for large hosting services which provide access to web objects belonging to third party information providers. Without appropriate new technology, system administration related to object placement may become a factor limiting the scale of these hosting applications. Secondly, the Internet is dynamic, with changes in topology and routing [23]. These changes can make static replicas ineffective. On the other hand, dynamic replication can adjust to them. Finally, access patterns to documents change over time. Certain events can increase the demand for a particular web site, suddenly overloading it. It is not feasible to a-priori predict these changes in demand. Even if such irregular patterns could be predicted, statically mirroring for the worst case might be prohibitively expensive. Dynamic replication can adjust to changes in demand and more optimally utilize the available server resources. We looked at the document access patterns in a 90-day HTTP trace from AT&T's Easy-WWW hosting service. Apart from the diurnal pattern that web traffic follows [16], we observed variations in access patterns over much larger time-scales. For a number of documents, the demand remained negligible for most of the 90-day period except for an interlude of a few hours or days.

Dynamic replication is similar to caching, in the sense that it creates copies of content. The main difference is that replication is done under the control of the service provider. So, issues like deployment and replica consistency are simplified. Moreover, many documents on the web can be replicated but not cached; for example, dynamically generated pages and personalized pages. Issues like copyright protection, usage metering, advertisement *etc* further limit the scope of caching. Thus, caching, while clearly useful, is complimentary to service-based replication, which is evidenced by a wide use of mirrors.

Our work focusses on the problem of dynamic replication for an Internet hosting service offerred by a large (i.e., global-scale) Internet Service Provider (ISP). Previously, we described a protocol for dynamic replication in this enviroment [27] and a scalable architecture for a hosting service that supports dynamic replication [25]. In this paper, we study the performance of various approaches to dynamic replication and also compare it with static replication using the general framework proposed in [25], [27].

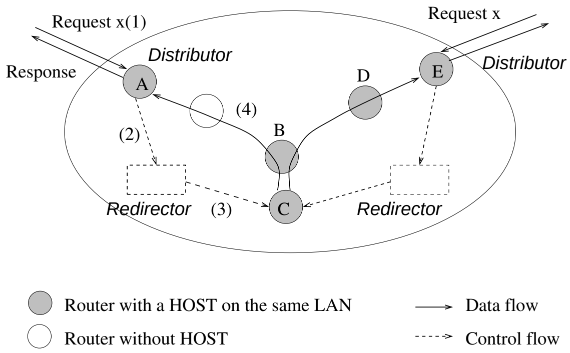

Fig. 1. Architecture of the system

In particular, we describe two classes of request distribution algorithms – feedback-based and non-feedback based. The feedback-based algorithm is more intuitive and performs well at low loads. The non-feedback scheme, on the other hand, is theoretically well grounded. Its main advantage is that it allows hosts to predict the effects of potential replica set changes on future request distribution, thereby making it suitable for a system where hosts take dynamic replica placement decisions.

Our performance study consists of two complimentary parts, the simulation using synthetic workloads and a trace-driven simulation. Synthetic workloads, due to their regularity, allow us to gain intuition about the behavior of and the tradeoffs among the considered approaches, as well as to study the sensitivity of the algorithms to various parameters. The trace study provides a more definitive prediction of real-life performance. Both studies used the topology of Worldnet, the AT&T ISP (for the synthetic part, we also studied the larger UUNET backbone topology). For the trace study, we used a 90-day trace of HTTP requests to EasyWWW, the AT&T hosting service.

Overall, our study shows a high effectiveness for dynamic replication. On the trace, it reduces bandwidth consumption by as much as 52% while the traffic overhead due to object replications and migrations stays within about 5% of the (reduced) total traffic. It also fullfils its promise of flexibility by ajusting to an artificial perturbation in demand patterns, during which static replication shows significant performance degradation.

The rest of the paper is organized as follows. Section II describes the architecture of the system. Section III presents our replica placement algorithm while section IV presents the two request distribution schemes. In sections V and VI, we describe the simulation of the algorithm and present some results. Section VII discusses related work and finally, section VIII contains conclusions.

II. System Model

Figure 1 shows the architecture of the system ([25] discusses it in greater detail). Consider an ISP that also provides a hosting service (for instance, Worldnet). External clients enter the backbone through special routers called border routers. Customers of the ISP may be connected to any of the backbone nodes. Hosted documents reside on *hosting servers*, or **hosts** for short. Each host is located on the same LAN as one of the backbone routers.

Each host runs the *replica placement algorithm* (section III) and dynamically creates and deletes replicas. Requests from clients are handled in the following manner:

1. When a client requests a document, the host name from the document URL is resolved into the IP address of a host which is co-located with the border router through which packets to the client are routed. We call this host the **distributor**. In effect, client requests are directed to their entry point into the backbone. The mapping from the host name to the distributor IP address is relatively static since it is independent of the number and location of replicas. Therefore, the existing DNS infrastructure can be leveraged to provide this mapping[1].
2. The distributor, after receiving the client request, forwards it to the **redirector** module, which runs the *request distribution algorithm* (section IV) and determines the best host to service the request. The redirector is a logical entity; its physical location may vary depending on the algorithm it executes. It could be replicated and co-located with every distributor if all the information it needs is locally available with each distributor. On the other hand, if the algorithm needs global information, the redirector might be located at a centralized location (see section IV-B for scalability and fault-tolerance issues related to this alternative).
3. The redirector chooses the *best* host for the request and forwards the request to this host.
4. The host replies with the data to the original distributor, which then responds to the client.

In the above model, the path from the host serving the request, to the client, through the distributor is crucial since the response (which is typically much larger than the request) is transmitted on it. Therefore, the redirector chooses a host that is *close* to the distributor to serve the request.

We do not consider the issue of replica consistency in this work, assuming it is handled by weak-consistency update propagation or replica invalidation. This is because multiple studies have shown that an overwhelming majority of Web accesses are to objects that either do not change on user access, or simply accumulate usage statistics, a commutative update that is easily reconciled (see, e.g., [12], [32]). Note that just because consistency is not a central issue, one cannot brush the Internet replication

[1] However, the resolution does depend on the client location. [9] provides an implementation of DNS name resolution based on origin of the requester.

problem aside by simply replicating every object on every server. First, even with cheap disk storage, the global scale of the system we are considering makes such solution impractical, for maintenance and administration reasons if nothing else. Second, the problem would still remain as to how to distribute requests among these replicas. In fact, we will see that for some request distribution approaches, having excessive number of replicas hurts the performance.

For our experiments, we replicate and migrate at the granularity of pages. A coarser granularity reduces the overhead of control state but increases the overhead of transferring and storing little used pages within large units. A detailed study of the tradeoffs and an efficient heuristic for dynamically grouping objects is left for future work.

For the rest of the paper, x_s refers to a replica of document x placed on host s. We will not distinguish hosts from backbone routers with which they are co-located on the same LAN. For instance, we will say that replica x_s is routed through host q on its way from host s to the client, meaning that x_s is routed through the router co-located with host q. An *access count*, $count(s1, x_{s2})$, refers to the number of times replica x_{s2} of document x travelled through host $s1$ on its way to a client from host $s2$, which processed the client's request. In particular, $count(s, x_s)$ is the number of times replica x_s was accessed and is a measure of its load. The counts are averaged over a moving time window. For instance, in figure 1, after processing the two requests shown, $count(B, x_C) = count(C, x_C) = 2$, while counts for A, D and E are 1.

III. Replica Placement Algorithm

This section briefly outlines the replica placement algorithm ([27] provides a more detailed description). Our algorithm has two major goals:

- Increase client server proximity: Replicas should be placed in such a manner that they become *closer* to the clients, thereby reducing latency and bandwidth consumption.
- Distribute load among servers: The replicas should be placed keeping in mind the load of the hosts in the system. Migrating a large number of replicas to a particular host might increase its load beyond acceptable limits.

Each host periodically runs the replica placement algorithm and takes decisions autonomously independent of the other hosts in the system. Section III-A describes how the algorithm increases client-server proximity while section III-B focuses on the load distribution aspect.

A. Client-Server Proximity

Each host s in the system maintains a list of *candidates* for each document replica that it hosts. A candidate is a host through which the replica has been routed as part of a response from host s to a client. For each candidate c, host s maintains the access count, $count(c, x_s)$, defined earlier. A high count implies that it is potentially beneficial to migrate or replicate the document to the candidate. Hosts use routing path information to maintain candidate counts[2]. Whenever a document is sent to a distributor as part of a response, the counts for all the hosts on the path to the distributor are incremented[3].

```
for each replica x_s present on this host
  if count(s, x_s) < U
    delete x_s
  else
    Attempt to migrate x_s to farthest candidate c st
    count(c, x_s) > MIGR_RATIO × count(s, x_s)
    If migration not successful
      if count(s, x_s) > M
        Attempt to replicate x_s to farthest candidate c st
        count(c, x_s) > REPL_RATIO × count(s, x_s)
```

Fig. 2. The replica placement algorithm executed at a host s

Figure 2 shows the algorithm at a very high level. A host deletes a replica if its load is below the deletion threshold (U) and it is not the sole replica in the system. If the load is above U, the document is migrated to a candidate such that at least $MIGR\_RATIO$ fraction of all responses are routed through it, where $MIGR\_RATIO$ must be greater than 0.5. This ensures that on migration, the replica would be closer to a majority of the requests, thereby improving client-server proximity. Secondly, it prevents a situation in which a replica is migrated back and forth between two nodes. Moreover, we migrate the replica to the farthest candidate which satisfies the above heuristic in order to reduce the adjustment time before the desired replica placement is reached (we call this time the *responsiveness* of the system, not to be confused with the latency for servicing client requests). In case migration fails because the heuristic is not met for any of the candidates or because of load constraints (section III-B), then we attempt to replicate the document to a candidate through which at least $REPL\_RATIO$ of the responses are routed. Replication is done only if the replica load is greater than the replication threshold, M, so that the benefit of replication out-

[2]We assume that hosts have access to the routing tables. [25] presents details about how the relevant information can be extracted from the routing tables.

[3]The total information maintained by a host is $|hosted\ documents| \times |candidates| \times size\ of\ candidate\ count$. Even a large ISP has of the order of 50 nodes which bounds the number of candidates. Assuming the size of the count for each candidate to be about 4 $bytes$, the total data is $O(200n)$, where n is the number of documents present on the host. Assuming n to be of the order of a million documents, the memory requirement is about $200MB$. This is a reasonable space overhead for modern server configurations. Further, we can cluster documents together so that they are moved together as a unit. For instance, an HTML file might be clustered along with all its inlined images, thereby reducing the value of n.

weighs its cost[4].

B. Load Balancing

Each host maintains an estimate of its load. Apart from the number of requests handled by the host, the load might reflect other factors like CPU and memory usage. The load estimate is updated when a new replica is created or an old one deleted. This allows a host to relocate multiple objects at once, without waiting for new access statistics after each migration or replication. Section IV describes how these load estimates are computed.

The replica placement algorithm uses two mechanisms to ensure that the load of a hosts remains below a threshold.

1. If the load estimate of a host increases above the high water mark (HW), it shifts into the *offloading* mode. In the offloading mode, a host migrates and replicates objects to other hosts even if it is not beneficial from the proximity perspective. Offloading stops when the load estimate falls below the low water mark (LW).
2. A host accepts a migration or replication only if its load estimate is below LW. For instance, if the host does not have sufficient memory, it will stop creating new object replicas.

IV. Request Distribution

This section describes the request distribution algorithm. The challenge in designing the request distribution algorithm lies in combining the goal of load distribution with that of choosing a replica that is close to the client. Choosing replicas in a round robin manner [6] would distribute the load but neglect proximity. On the other hand, always choosing the closest replica [9], [19] could result in poor load distribution. We have explored two approaches to request distribution, both of which try to increase client-server proximity and distribute load at the same time.

A. Feedback based Approach

In the feedback approach, the redirector maintains load information about all the hosts in the system. It sends a client request to the *closest replica whose load is below a threshold*. This simple and intuitive approach takes care of proximity and at the same time distributes load. Also, there is a feedback loop in which the load affects request distribution, which in turn has a bearing on the load of the hosts.

In the feedback case, the redirector module is physically co-located with the distributor. Load information is piggybacked on to responses from the hosts. This information is aged and periodically invalidated. When the load of a host

[4] For simplicity, we assume that replicating involves simply transferring the document to the receiver node. In case it involves transfer of an underlying environment, e.g., for computing dynamic documents, then an explicit cost metric has to be used.

Loop through the replicas in order of increasing distance from the distributor
For each replica x_i /* Replica of x on host i */
  if $load(i) < LW$
    Forward request to host i
  if $(load(i) > LW) \wedge (load(i) < HW)$
    Forward request to host i with probability
    proportional to $\frac{HW - load(i)}{HW - LW}$
In case request has not been forwarded, send it to the least loaded replica.

Fig. 3. Feedback based algorithm for selecting a replica for document x

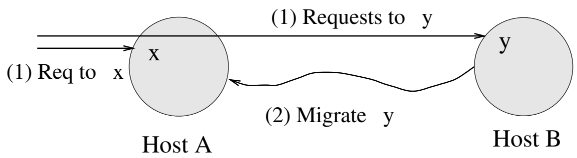

Fig. 4. The figure describes a scenario illustrating the importance of coupling between the request distribution and replica placement algorithms.

rises above HW, the redirector stops sending requests to it. However, after the load information expires, the redirector again starts sending requests to the host.

A major problem with the above algorithm is global synchronization – a situation in which all the distributors send requests to the same host until its load rises above the threshold, and then all of them withdraw simultaneously. In order to prevent the oscillation of loads, we use a probabilistic algorithm (figure 3). If the load of the host is below LW, we send the request to it. However, if it is between LW and HW, we send the request to the host with a probability that is a function of the host load. Our simulations show that the modified algorithm successfully prevents loads from oscillating.

Discussion

The feedback based scheme aggressively sends requests to the closest replica until the host saturates. This implies that when the load in the system is low, all requests would be distributed to their closest replicas. Even at high load, the algorithm tries to send requests to the closest non-overloaded replica.

However, a drawback of the scheme is that the replica placement and request distribution algorithms are decoupled. In particular, it is very difficult to predict the effect of a migration or replication on request distribution. For instance, it is not possible to a-priori determine the number of requests that would be sent to a new replica and hence, the potential benefit of creating it cannot be estimated. This makes taking dynamic replica placement decisions difficult. As another example, consider the scenario in figure 4. (1) A replica of document x resides on host A while document y resides on host B. All requests to y pass through host A. (2) Document y is migrated to host A in order to improve proximity. (3) The replica of y on host A becomes closer to a lot of requests; the load on A increases; requests to x start getting diverted to alternate (far-off) replicas and the replica of x on A is deleted. Therefore, it is not obvious that the migration of y was advantageous for the overall system performance. Such a scenario becomes more likely at high loads.

There are two properties of our scheme and feedback schemes in general that lead to such situations:

1. The effect of a migration/replication on request distribution cannot be accurately predicted. For instance, in the above example, we could not predict that migrating document y to host A would overload the host. Therefore, taking dynamic placement decisions is difficult.
2. Since aggregate (instead of per-document) information is used for distributing requests, the movement of one document might have an impact on the way requests are distributed to another one. This makes it all the more difficult to assess the benefit of a particular replica placement decision. In the given example, change in the replica set of document y affected document x in an unpredictable manner.

B. Non-Feedback Based Approach

Predicting the effect of replica set changes on request distribution is important for any system that takes dynamic replication decisions. Making such estimates is difficult in schemes which involve a feedback loop. In the non-feedback approach, we explore a simple deterministic algorithm without feedback that allows us to derive bounds on the load of hosts and predict the effect of object migrations and replications. In this case, there is a centralized redirector which maintains a count of the total number of request sent to each replica of a document. All requests for a document are directed to this centralized redirector; the redirector chooses the best host for the request and updates the count information corresponding to the chosen replica. We denote the redirector's count for replica x_i as $RD\_count(x_i)$, to distinguish it from the access counts maintained by the hosts.

The algorithm sends a request to the closest replica such that the following invariant is maintained: *the count of the most heavily loaded replica is at most a constant times the count of the most lightly loaded replica.* Figure 5 illustrates the algorithm used by the redirector to distribute requests. It computes the rank for each replica and distributes requests in such a manner that each replica has a count of at most rank times the count of the least loaded replica. This ensures that the invariant is satisfied if the maximum rank is bounded. The rank of a replica depends

```
min_RD_count = min_i{RD_count(x_i)}
Loop through the replicas in order of increasing distance
from the client
For each replica x_i
  if (RD_count(x_i) <= min_RD_count × rank(x_i))
    Send request to i
    RD_count(x_i) = RD_count(x_i) + 1
  /* Note: The least loaded replica always trivially satisfies
  the above condition */
```

Fig. 5. Non-Feedback request distribution algorithm for selecting a replica of document x

on where the particular request originated - a replica that is close to the client has a higher rank. To get an intuition into the algorithm, consider a scenario in which an object has two replicas, x_A and x_B. Assume that we assign the closer replica a rank of 2 and the farther one a rank of 1. Then, even if all clients are closer to host A, x_A would receive 66% of the requests. However, if half of the requests are closer to x_A, they would be directed to x_A, while the other half (which are closer to x_B) would be directed to x_B.

A centralized redirector raises issues about the scalability and fault-tolerance of the system. These issues, however, are easily addressed as follows:

- Since the redirector requires count information only for the replicas of the requested document, it can be partitioned and requests can be mapped on to a redirector by hashing the requested URL. For instance, there can be a redirector for each hosted Web site, that handles all the documents contained on that site. Further, the redirector handles only UDP messages and the request distribution algorithm is light weight.
- The redirector only has soft-state. If the redirector is down, requests can be forwarded to a default replica. On being restarted, the redirector builds up state automatically.

Load Estimates

The property that the count of the most heavily loaded replica is bounded leads to nice theoretical results about the load estimates of a replica. [26] presents these results in greater detail, including the proofs and the formal definition of *steady demand.*

Theorem

If the total number of requests issued for a document x over any given interval is L, the number of requests directed to a particular replica x_i is l, the total number of replicas of x is n and the maximum rank of a replica is r_{max}, then, under steady demand,

$$l + \frac{l \times (n-1)}{r_{max}} \leq L \leq l + l \times r_{max} \times (n-1);$$

$$\frac{L}{1 + (n-1) \times r_{max}} \leq l \leq \frac{L \times r_{max}}{r_{max} + n - 1}$$

Theorem
Let host i replicate or migrate document x on host j. Let l_s be the load on x_i (replica of x on host i) before replication or migration and l_r be the load on host x_j after replication. Then, under steady demand,

$$\frac{l_s}{r_{max}^2} \leq l_r \leq l_s \times r_{max}^2$$

Discussion

The non-feedback algorithm distributes requests to all the replicas of an object. Therefore, even if the closest replica is lightly loaded, some of the requests would be served by far-away hosts. In contrast, the feedback scheme would direct all requests to the closest replicas. Thus, at low loads the feedback scheme would perform better.

However, the ability to predict the effect of a replication or migration has a number of desirable effects on the responsiveness, stability and performance of the system:

- It is possible to estimate whether a replica placement is beneficial for the system or not. For instance, on a replication, the load of the newly created replica is at least l_s/r_{max}^2. In order to prevent a cycle of replications and deletions, the load on the new replica should be greater than the deletion threshold, U (section III). We can ensure this by setting the replication threshold M to $r_{max}^2.U$, thus adding stability to the system. In addition, since the request distribution depends on per-document statistics and not aggregate information, there is isolation between documents *i.e.* change in the replica set of one document does not affect how requests are distributed to another one.
- The responsiveness of the system increases as the load estimates allow us to replicate or migrate enmasse, without waiting for new load statistics after each move.
- A host uses the upper bound on load to determine if a new object can be accepted. This keeps the load of the host below the threshold and prevents a situation in which load rises above the threshold due to overly aggressive creation of replicas. The upper-bound estimate is replaced with actual load once no replica creations occur in a time window over which the load metric is averaged.

V. Synthetic Workload Simulation

In this section, we present simulation results using synthetic workloads. We have made two assumptions in our simulation model that deviate it from a real world scenario: (i) Document size is assumed to be constant while numerous studies have shown that it is highly variable. (ii) The request rate is constant while in reality, requests are very bursty. Therefore, the results in this section cannot be extrapolated to the performance of the algorithm in the real world. However, they develop insights into the behaviour of the system and are useful for studying the sensitivity of the algorithm to various parameters. The next section describes the results of a trace driven simulation, which is more realistic and takes care of both the above caveats.

A. Simulation Model

We have simulated the algorithm on the UUNET [30] topology, which consists of 53 nodes in North America, Europe, Pacific and Australia (or so it did at the time of this study) as well as the Worldnet topology which has 14 nodes. Each node in the network can service requests for documents. Initially, all the 10000 documents are distributed among the nodes in round robin fashion. The nodes run the replica placement algorithm at intervals of 100 seconds. All documents are assumed to be of a fixed size ($12KB$) and the request size is negligible compared to the document size. A request incurs latency due to queuing and processing at the server and network delays. The servers follow a FIFO model for request processing. For the non-feedback scheme, the redirector is co-located with a node that has the smallest average distance to other nodes in the topology. In the future, we intend to explore the issue of optimal placement of the redirector for each document in order to minimize latency.

The nodes emit requests at a constant rate. We have used the following workloads for our simulation. In reality, however, a workload would be some combination of the workloads we have considered.

- **Zipf**: Previous studies [11] have observed that the popularity of pages requested by a client follows Zipf's law. According to Zipf law, if pages are ranked according to their access frequency, then the popularity of i-th most popular page is proportional ot $1/i$.
- **Hot Pages**: All pages are divided randomly into hot and cold pages, with p fraction of the pages going to the cold bucket while the remaining to the hot bucket. We choose $p = 0.9$. A client chooses a random page from the hot bucket with probability p and from the cold bucket with probability $1 - p$.
- **Hot Sites**: All sites are divided into hot and cold sites in the ratio $1 : 9$. A document from a hot site is requested with a high probability of 0.9 and from a cold site with probability 0.1.
- **Regional**: All nodes in the network are divided into four regions and each region is assigned a preferred set of documents. Each node requests a document from its region's preferred set with a high probability p and a random object from all the documents with probability $1 - p$ ($p = 0.9$). The regional workload models the situation when object popularity varies

Parameter	Value
Number of documents	10000
Size of document	$12KB$
Link latency	$10ms$
Link bandwidth	$350KBps$
Server Capacity	$200reqs/sec$
High&Low Watermark	$80\&70reqs/sec$
Inter-request time	$0.025sec$
Replication threshold M	$8 \times U$
Deletion threshold U	0.0125

Fig. 6. Parameters used for the simulation

with location.

The parameters used for the simulation are shown in table 6. The values of the link bandwidth, latency and size of documents are based on our experiments on traces of accesses to pages hosted by AT&T's EasyWWW hosting service and on the Worldnet topology. The link bandwidth reflects the bandwidth that is available to an average document transfer and not the raw link capacity. This is more approapriate than computing per-transfer bandwidth by sharing the link capacity among concurrent transfers because there is an unknown amount of unrelated traffic being handled by the backbone at the same time.

B. Performance Results

Figures 7- 9 plot the bandwidth consumption (in bytes per request), average latency of a request and maximum load in the system for the Zipf workload using the UUNET topology (results for the Worldnet topology and other workloads show similar trends). The bandwidth consumption is computed by summing the number of bytes transmitted on each hop. Therefore, the path taken by the response has a greater impact on the bandwidth consumed. The graphs compare the performance of both the feedback and the non-feedback schemes with the case when there is no dynamic replication.

- Dynamic replication considerably reduces bandwidth consumption (figure 7) and client latency (figure 8). This shows that the algorithm successfully improves client-server proximity by replicating and migrating documents. This performance trend is demonstrated for all the workloads, implying that the algorithm can adjust to different access patterns. Further, we note that the latency for the non-feedback case is higher since it involves a detour to the redirector.
- The maximum load in the system remains below the threshold of 80 requests/sec (figure 9). Initially, the maximum load is very high, which is a consequence of a few documents being very popular. Our algorithm manages to eliminate hot spots by replicating and migrating popular documents.

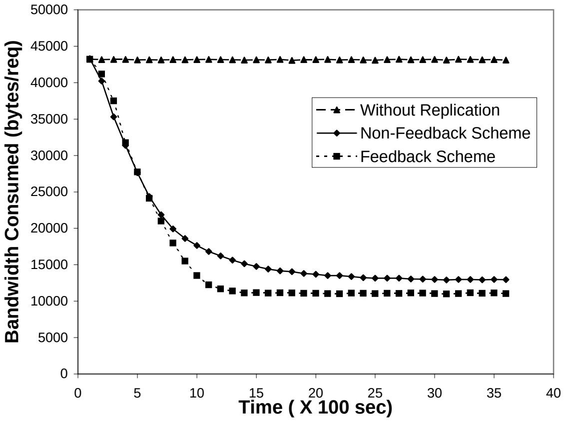

Fig. 7. Bandwidth Consumption (zipf workload)

- The overhead traffic remains below 5% of the total traffic for both the schemes. This overhead is computed as a percentage of the already reduced total traffic. When compared to the original traffic (without replication), the overhead is as low as 1.3%. Thus, the algorithm attains a stable configuration under steady demand.
- The average number of replicas created is around 2.7. Considering that the number of hosts in the topology (and hence the potential number of replicas) is 53, this shows that the algorithm is successful in selectively identifying objects that are most useful to mirror.
- We define the adjustment time as the time it take to reach a bandwidth consumption that is 10% above the average equilibrium bandwidth consumption. The adjustment time for the feedback scheme is 18 minutes while that for the non-feedback scheme is 26 minutes. In fact, significant bandwidth savings come even much earlier than that.

All the remaining workloads also demonstrate similar performance trends.

Summing up, our replica placement algorithm successfully replicates and migrates documents according to the workload in order to increase client-server proximity, balances load and stabilizes under steady demand.

C. Effect of Replication Threshold

Choosing the replication threshold presents a tradeoff between the resources and performance. A low threshold improves performance but also increases the number of replicas. Figure 10 plots the equilibrium bandwidth against the average number of replicas. The different points were generated using different values of the replication threshold. As the threshold decreases, more replicas are created leading to improved performance. For the non-feedback scheme, however, a higher number of replicas can be detrimental to performance, since requests are distributed to all replicas, even if they are not close to the

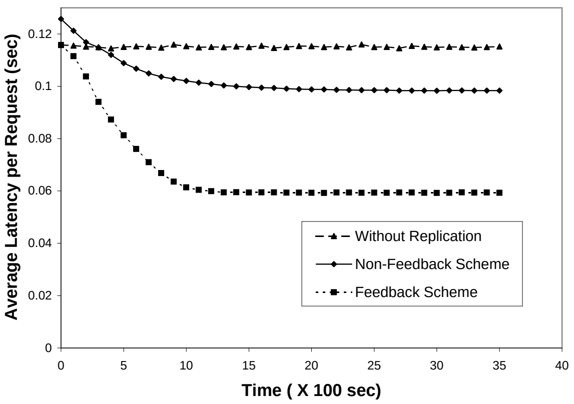

Fig. 8. Average Response Latency (zipf workload)

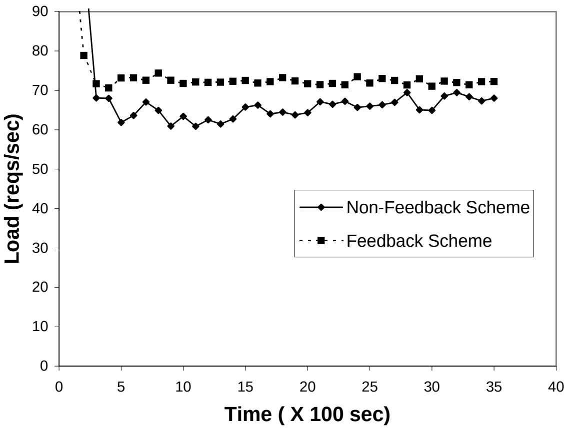

Fig. 9. Maximum Load in the System (zipf workload)

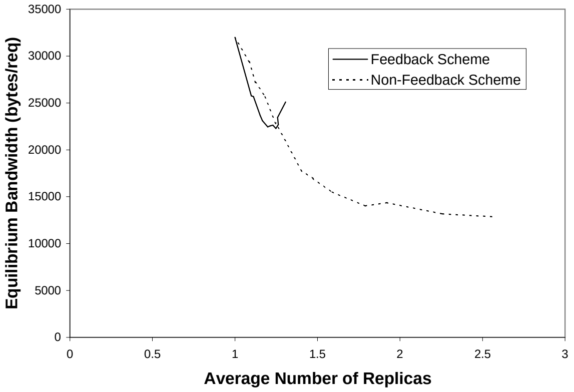

Fig. 10. Effect of Replication Threshold (hot sites)

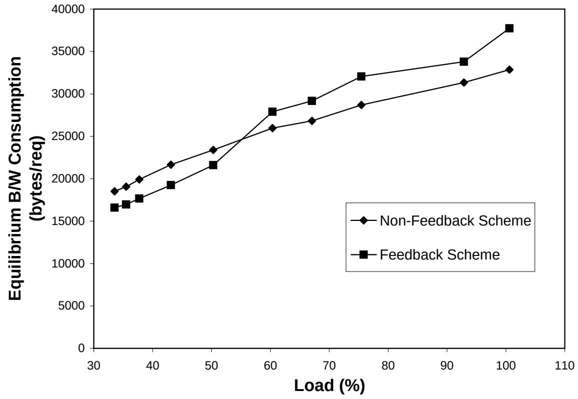

Fig. 11. Effect of Load on Performance (zipf workload)

clients. Further, the non-feedback scheme is not as sensitive to the threshold as the feedback scheme.

D. *Effect of Load*

We define load as the ratio of offered load and the total capacity of the system. The load is varied by changing the two watermarks, LW and HW, which determine the maximum allowable load on a host. Lowering the watermarks reduces the total capacity of the system and increases the load if the offered load remains the same.

Figure 11 plots the equilibrium bandwidth against the load on the system for the Zipf workload. As the load increases, performance of both the schemes deteriorates. This is because nodes which are potential recipients of replicas become over-loaded and hence are unable to accept migrations or replications. Thus, the algorithm is unable to move documents aggressively. Moreover, the algorithm has no mechanism of exchanging objects between overloaded nodes. For instance, if node i has an object accessed mostly from node j and vice-versa, it would make sense for these nodes to swap the objects - the load would remain the same but proximity would improve. Yet, the high load prevents either node from accepting any request for object placement.

Figure 11 also shows that the feedback scheme performs well at low loads. When nodes are not over-loaded, the feedback based request distribution algorithm can direct client requests to the closest replica, while the non-feedback scheme distributes requests to all the replicas. At high loads, however, the ability to predict the effect of a migration or replication becomes crucial in order to dynamically take placement decisions. Since, in the feedback scheme, it is not feasible to make such predictions, it might result in placement decisions that diminish performance (section IV-A). This effect results in the non-feedback scheme outperforming the feedback scheme at high loads. The crossover occurs when the load is around 55%.

VI. Trace Driven Simulation

We have simulated the algorithm on the EasyWWW trace, which is a log of HTTP requests to servers hosted by AT&T's EasyWWW hosting service. The trace covers a three month period, contains around 111 million requests to more than 100 sites and 0.2 million documents. In order to make the simulation efficient, we pruned uninteresting documents whose peak access rate was below 0.01 requests per second. This led to a 72% reduction in the number of documents while only a 1.3% reduction in the

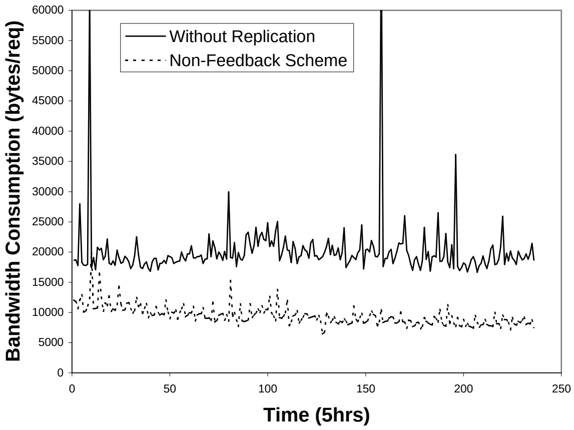

Fig. 12. Bandwidth Consumption (trace)

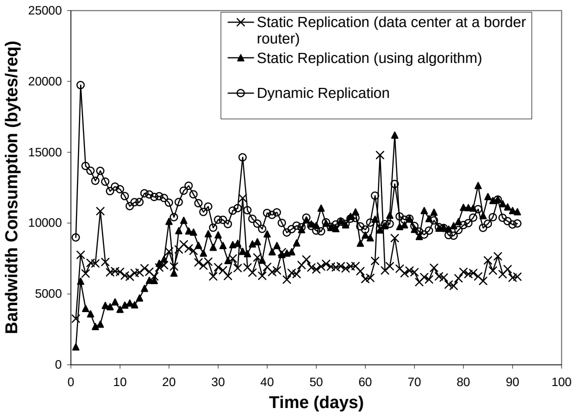

Fig. 13. Comparison between Static and Dynamic Replication

number of requests and 1.1% reduction in total number of bytes requested. Therefore, the performance of the algorithm remains practically unaffected by this pruning. For simulation, we used the 14-node Worldnet topology. Initially, all the documents were placed at a host that corresponds to the backbone node where the centralized Easy-WWW data server is currently located. The sizes of the various documents as also the time of initiating each request were obtained from the trace. Network distances, as well as the closest backbone exit routers to clients were obtained using actual routing tables and the Worldnet customer database. In cases when multiple exit routers were equidistant to a client, we used geographical distances from the Merit database [24] to break the ties. Failing that, the best exit router was chosen at random among the equidistant routers.

A. Performance

Figures 12 plots the bandwidth consumption for the trace simulation. With dynamic replication (non-feedback scheme), the bandwidth requirements reduce by up to 52% as compared to the case when there is no replication. Moreover, our algorithm is able to smooth out bursts in bandwidth demand. The traffic overhead for object relocations is low (less than 5% of already reduced total traffic). Further, the algorithm creates very small number of replicas. The performance of the feedback scheme is similar – the number of replicas created is so small that the request distribution algorithm is irrelevant.

An interesting issue is the cost of this improvement in performance *i.e.* how much extra resources in terms of server capacity and memory are required to achieve the lower bandwidth consumption. In order to estimate the resources used by our algorithm, we computed the maximum amount of data placed at each host (over the duration of the trace) as well as the maximum load on it. The total storage requirement, obtained as the sum of these highest-reached space watermarks, turns out to be about 20% more than the case when documents are not replicated. Similarly, the total server processing capacity requirement is only about 9.7% more than the capacity required if all the documents were placed at the same host. This implies that the reduction in bandwidth can be achieved simply by tearing apart resources from the centralized server and distributing them in the backbone.

B. Comparison with Static Replication

A system using dynamic replication has a number of advantages: administrative scalability, convenience and adaptability to network characteristics and demand. However, an important question is whether dynamic replication can achieve performance comparable to a brute force static replication approach. In other words, does the trade-off between performance, convenience and adaptability favour dynamic replication or not.

Finding an optimal static replica set entails solving a global optimization problem that can be stated as follows: *Given the capacity of each node in the network and the number of requests to each document entering the network through each of the nodes, what is the number and location of replicas of the documents that optimizes bandwidth consumption*. The problem is computationally hard and practically unsolvable, considering the large number of documents. Therefore, we use an approximation algorithm that greedily places replicas based on the unit profit of an object replica.

In order to compare static and dynamic replication, we arbitrarily fixed the resources at each node in the network. Then, we used the initial 30-day subset of the trace (called the training trace) to determine the best static replica set, using our greedy algorithm. Finally, we simulated the whole trace, using our dynamic replication algorithm as well as the static replica set. We also explored a number of other ad-hoc static replication schemes, which led to the observation that shifting the centralized data server

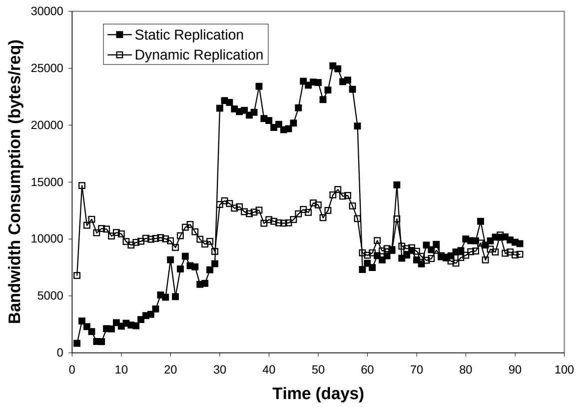

Fig. 14. Adaptability of Static and Dynamic Replication

to one of the border routers performs quite well[5]. Figure 13 illustrates the performance of the various schemes. Dynamic replication achieves performance comparable to static replication (except for the training trace period). We believe that the comparable performance, coupled with a number of other favourable factors factors, makes dynamic replication a win:

- It is very difficult to statically determine the replica set that performs best. In our case, it turned out that moving the data center to one of the border routers performs well. However, in general, such an ad-hoc solution is practically infeasible. Even using an algorithm to determine the replica set does not scale administratively.
- Dynamic replication can adapt to changes in access patterns. In order to illustrate this, we conducted an experiment in which we randomly re-mapped the requested document ID (using a simple hash function) for each request in the middle part of the trace. Figure 14 shows the performance of static and dynamic schemes in this rather contrived scenario. The performance of static replication degrades significantly when the access patterns change as a result of the remapping. On the other hand, dynamic replication adjusts to this change in demand and maintains its performance.
- Dynamic replication can deliver better performance to the end client by adjusting to transient changes in demand of individual documents. For example, static replication cannot adjust to a sudden burst in the demand for a particular demand. However, dynamic replication can replicate the document to cater to the increased demand. While such short-term, localized changes do not reflect in the overall system performance, they have a significant bearing on end-user perceived performance.

[5]While this was a useful side-result for the current Worldnet, in a true global-scale system, keeping all the data in one centralized location would not be feasible. More importantly, the result happens to be true for a particular access pattern unique to Worldnet during a particular time span. It would not hold, in general, for a global scale ISP with large number of customers and external clients all over the globe.

VII. Related Work

Several commercial products offer transparent load balancing among Web servers, using a front-end IP multiplexor (see [14] for a now-incomplete survey) or DNS (e.g., [21], [10]) as request distributors. A few products allow request distribution over a wide-area network [9], [20], [31]. Several research proposals for server selection in a wide-area network have also been described (e.g., [15], [28], [7], [29]). All these approaches focus on replica selection and do not consider the replica placement aspect.

Two start-ups, Sandpiper and Akamai, are offering dynamic content replication. Neither provides any details on the algorithms they use. These companies provide only hosting services while our focus is on a platform providing both Internet and hosting services. This has various implications, including the availability of information to drive the algorithms (such as routing information).

Guyton and Schwartz describe various methods that can be used to determine proximity of nodes in the network [17]. We use routers databases to extract proximity information. One reason for using the routing-based proximity metric is our network-centric view, in which reducing the backbone bandwidth is an overriding concern. A similar proximity measure is used by the CISCO Distributed Director [9].

Heddaya and Mirdad have proposed a *WebWave* protocol for replicating Web objects from the content server up the router tree towards the clients [19]. The protocol achieves almost perfect load balancing among replicas. However, it burdens the Internet routers with the task of maintaining replica locations for web objects and intercepting and interpreting requests for Web objects. It also assumes that each request arrives in a single packet. As the authors note, this protocol cannot be deployed in today's networks.

Baentsch et al [2] propose an infrastructure for performing replication on the Web, without describing the underlying algorithms. Gwertzman and Seltzer [18] motivate the need for geography-based object replication. They propose to base replication decisions on the geographical distance (in miles) between clients and servers. This measure may not correctly reflect bandwidth consumption for fetching an object, since the network topology often does not correspond to the geographical distances.

The problem of placing objects in the proximity of requesting clients has also been addressed in research on file allocation. Early work in this area assumes a central point where decisions on object placement are made by solving an integer programming problem. Even when the search space is heuristically pruned, the scale of our application would make such an approach impractical. More recently, Awerbuch, Bartal, and Fiat have designed a distributed file allocation protocol [1]. However, their protocol does not address the load balancing issue.

VIII. Conclusion

The paper presents an algorithm for dynamic replication and migration of web objects. Simulations of the algorithm on a trace show that dynamic replication successfully reduces bandwidth consumption by as much as 52% compared to the current system and smooths out bursts in bandwidth demand. The overhead traffic is negligible and the number of replicas created is small. Our experiments show that the performance gains can be achieved simply by distributing resources in the network instead of concentrating them at a centralized server. Further, our algorithm achieves performance comparable to any static replication scheme. We have also explored two classes of request distribution algorithms - feedback and non-feedback based. Simulations on a synthetic workload show that feedback algorithms work well at low loads while non-feedback schemes are better at high loads. This leads to the idea of a hybrid approach, which uses feedback at low loads and shifts to the non-feedback approach as the load increases.

We are very grateful to Balachander Krishnamurthy for providing us with EasyWWW traces, which were also used for our earlier work [25], and to Euthimios Panagos for his simulation toolkit used in our experiments.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed Paging for General Networks. In *Proc. of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pp. 574-583, January 1996.

[2] M. Baentsch, L. Baum, G. Molter. S. Rothkugel, and P. Sturm. Enhancing the Web's Infrastructure: From Caching to Replication. *IEEE Internet Computing*, Vol 1, No. 2, pp. 18-27, March/April, 1997.

[3] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proc. of the IEEE Symp. on Parallel and Distr. Processing*, pp. 338-345, 1995.

[4] A. Bestavros and C. Cunha. Server-initiated document dissemination for the WWW. *Bulletin of the Computer Society technical Committee on Data Engineering*, pp. 3-11. Vol. 19, No. 3, September 1996.

[5] H. W. Braun and K. C. Claffy. An Experimental Means of Providing Geographically Oriented Responses Relative to the Source of Domain Name Server Queries Technical Report, San Diego Supercomputing Center, April 1994.

[6] T. Brisco. DNS Support for Load Balancing. Network Working Group RFC 1794, April 1995.

[7] R. Carter and M. Crovella. Server selection using dynamic path characterization in Wide-Area Networks. In *IEEE Infocom'97*, 1997.

[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz and K. Worrell. A Hierarchical Internet Object Cache In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[9] Cisco Distributed Director. White paper available at `http://www.cisco.com/warp/public/751/distdir/dd_wp.htm`

[10] M. Colajanni, Ph. S. Yu, and D. M. Dias. Scheduling Algorithms for Distributed Web Servers. In *Proc. 17th IEEE Intl. Conf. on Distributed Computing Systems*, May 1997.

[11] C. A. Cunha, A. Bestavros and M. Crovella. Characteristics of WWW Client-based Traces. Technical Report TR-95-010, Department of Computer Science, Boston University, April 1995.

[12] F. Douglis, A. Feldmann, B. Krishnamurthy and J. Mogul Rate of Change and other Metrics: a Live Study of the World Wide Web *Proceedings of the Symposium on Internet Technologies and Systems*, December 1997.

[13] B. M. Duska, D. Marwood and M. J. Feeley. The Measured Access Characteristics of World Wide Web Client Proxy Caches. In *Proc. of USENIX Symposium on Internet Technologies and Systems*, December 1997.

[14] R. Farrell. Distributing the Web load. *Network World*, September 22, 1997.

[15] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of IEEE INFOCOM'98*, 1998.

[16] S. D. Gribble and E. A. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 207–218. December 1997.

[17] J. Guyton and M. Schwartz. Locating nearby copies of replicated Internet sarvers. In *Proceedings of SIGCOMM'95*, 1995.

[18] J. Gwertzman and M. Seltzer. The Case for Geographical Push Caching. In *Proc. of the HotOS Workshop*, 1994. Also available as ftp://das-ftp.harvard.edu/techreports/tr-34-94.ps.gz.

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

[20] IBM Interactive Network Dispatcher. `http://www.ics.raleigh.ibm.com/netdispatch/`

[21] E. Katz, M. Butler, and R. McGrath. A scalable Web server: the NCSA Prototype. *Computer Networks and ISDN Systems*, 27, pp. 155-164, September 1994.

[22] Q. Kure. Optimization of File Migration in Distributed Systems. PhD Dissertation, University of California (Berkeley), 1988. Also available as Technical Report UCB/CSD 88/413, Computer Science Division (EECS), University of California (Berkeley), April 1988.

[23] C. Labovitz, G. Robert Malan and F. Jahanian Internet Routing Instability In *Proceedings of SIGCOMM 97*, Cannes, France, September 1997.

[24] Merit Database. Available at ftp://nic.merit.edu/nsfnet/announced.networks/nets.unl.950428

[25] M. Rabinovich and A. Aggarwal. RaDaR: A Scalable Architecture for a Global Web Hosting Service *To appear, WWW8, May 1999.*

[26] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic Replication on the Internet. AT&T Labs Technical Report HA6177000-980305-01-TM, March 1998. Available at `http://www.research.att.com/~misha/radar/tm.ps.gz`.

[27] M. Rabinovich, I. Rabinovich, R. Rajaram and A. Aggarwal. A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service. *To appear, ICDCS 99, May 1999.*

[28] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. Workshop on Internet Server Performance, June 1998. Available at

[29] S. Seshan, M. Stemm, and R. Katz. SPAND: shared passive network performance discovery. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 135-146, 1997

[30] UUNET Global Backbone Available at `http://www.uu.net/lang.en/network/`

[31] Web Challenger. White paper, WindDance Network Corporation. Available at `http://www.winddancenet.com/newhitepap`

[32] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla and E. A. Fox Removal Policies in Network Caches for World-Wide Web Documents In *Proceedings of ACM SIGCOMM'96 Conference*, August 1996.

[33] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems (TODS)*, Vol. 22(4), June 1997, pp. 255-314.