On the Clustering of Web Content for Efficient Replication [*]

Yan Chen
UC Berkeley

Lili Qiu
Microsoft Research

Weiyu Chen, Luan Nguyen, Randy H. Katz
UC Berkeley

Abstract

Recently there has been an increasing deployment of content distribution networks (CDNs) that offer hosting services to Web content providers. In this paper, we first compare the un-cooperative pull-based replication of Web contents used by commercial CDNs with the cooperative push-based approach. Our results show that the latter can achieve comparable users' perceived performance with much less replication and update traffic (4 - 5% of those in the former scheme). Motivated by the observation, we explore how to efficiently push content to CDN nodes. Using trace-driven simulation, we show that replicating content in units of URLs can yield 60 - 70% reduction in clients' latency compared to replicating in units of Web sites. On the other hand, it is very expensive to perform such a fine-grained replication.

To address this issue, we propose to replicate content in units of clusters, each containing objects which are likely to be requested by clients that are topologically close. To this end, we describe three clustering techniques, and use various topologies and several real traces from large Web servers to evaluate their performance. Our results show that cluster-based replication achieves 40 - 60% improvement over full Web site replication. In addition, by adjusting the number of clusters, we can smoothly trade off the management and computation cost for better client performance.

To take into account of change in users' access patterns, we also explore incremental clusterings to adaptively add new documents to the content clusters. We examine both offline and online incremental clusterings, where the former assumes access history is available while the latter predicts access pattern based on the hyperlink structure. Our results show that the offline clusterings yield close to the performance of the complete re-clustering while at much lower overhead. The online incremental clustering and replication cut down the retrieval cost by 4.6 - 8 times compared to no replication and random replication, so it is especially useful to improve document availability during flash crowds.

Keywords: Content Distribution Network (CDN), replication, Web content clustering, stability.

1 Introduction

In the past decade, we have seen an astounding growth in the popularity of the World Wide Web. Such growth has created a great demand for efficient Web services. One of the primary techniques to improving Web performance is to replicate content to multiple places in the Internet, and have users get data from the closest data repository. Such replication is very useful and complementary to caching in that (i) it improves document availability during flash crowds as content are pushed out before they are accessed, and (ii) pushing content to strategically selected locations (i.e., cooperative push-based replication) yields significant performance benefit than pulling content and passively caching them solely based on users' request sequence (i.e., un-cooperative pulling).

A number of previous works [15, 25] have studied how to efficiently place Web server replicas on the network, and concluded that a greedy placement strategy, which selects replica locations in a greedy fashion iteratively, can yield close to optimal performance (within a factor of 1.1 - 1.5) while at much lower computa-

[*]This paper is an extended version of the paper that will appear in the Proceedings of the 9th International Conference on Network Protocols (ICNP'2002), November 2002.

tional cost. Built upon the previous works, we also use the greedy placement strategy for replicating content. In our work, we focus on an orthogonal issue in Web replication: what content is to be replicated.

We start by analyzing several access traces from large commercial and government Web servers. Our analysis shows that 10% of hot data can cover over 80% of requests, and this coverage can last for at least a week in all the traces under study. This suggests that it is cost effective to replicate only the hot data and leave the cold data at the origin Web server.

Then we compare the traditional un-cooperative pulling vs. cooperative push-based replication, both of which replicate only hot documents. Our results show that the latter scheme can yield comparable clients' latency while only using about 4-5% of the replication and update cost of the former.

Motivated by the observation, we explore how to efficiently push content to CDN nodes. We compare the performance between per Web site-based replication (all hot data) versus per hot URL-based replication. We find the per URL-based scheme yields a 60-70% reduction in clients' latency. However, it is very expensive to perform such a fine-grained replication: it takes 102 hours to come up with the replication strategy for 10 replicas per URL on a PII-400 low end server. This is clearly not acceptable in practice.

To address the issue, we propose several clustering algorithms that group Web content based on their correlation, and replicate objects in units of content clusters (i.e., all the objects in the same cluster are replicated together). We evaluate the performance of cluster-based replication by simulating their behavior on a variety of network topologies using the real traces. Our results show that the cluster-based replication schemes yield 40 - 60% improvement over the per Web site replication, but only at 1% - 2% of computation and management cost of the URL-based scheme (The management cost include communication overhead and state maintenance for tracking where content has been replicated).

Finally, as the users' access pattern changes over time, it is important to adapt content clusters to such changes. Simulations show that clustering and replicating content based on old access pattern does not work well beyond one week; on the other hand, complete re-clustering and re-distribution, though achieves good performance, has large overhead. To address the issue, we explore incremental clusterings that adaptively add new documents to the existing content clusters. To this end, we examine both offline and online incremental clusterings, where the former assumes access history is available while the latter predicts access pattern based on hyperlink structure. Our results show that the offline clusterings yield close to the performance of the complete re-clustering while at much lower overhead. The online incremental clustering and replication reduce the retrieval cost by 4.6 - 8 times compared to no replication and random replication, so it is very useful to improve document availability during flash crowds.

The rest of the paper is organized as follows. We survey previous work in Section 2. We describe our simulation methodology in Section 3, and study the temporal stability of popular documents in Section 4. In Section 5, we compare the performance of the pull-based vs. push-based replication. We formulate the push-based content placement problem in Section 6, and compare the the Web site-based replication and the URL-based replication using trace-driven simulation in Section 7. We describe content clustering techniques for efficient replication in Section 8, and evaluate their performance in Section 9. In Section 10, we examine offline and online incremental clusterings that take into account of changes in users' access pattern. Finally we conclude in Section 11.

2 Related Work

A number of research efforts have studied placement of Web server replicas or Web caches. Li *et al.* approached the proxy placement problem with the assumption that the underlying network topologies are trees, and modeled it as a dynamic programming problem [18]. While an interesting first step, this approach has an important limitation that the Internet topology is not a tree. More recent studies [15, 25], based on evaluation using real traces and topologies, have independently reported that a greedy placement algorithm can provide content distribution networks with performance that is close to optimal. Furthermore, Qiu *et al.* found although the greedy algorithm depends on estimates of client distance and load predictions, it is relatively insensitive to errors in these estimates [25].

There is considerable work done in data clustering, such as K-means [16], HAC [28], CLANRNS [22], etc. In the Web research community, there have been many interesting research studies on clustering Web content or identifying related Web pages for various purposes, such as pre-fetching, information retrieval, and Web page organization, etc. Cohen *et al.* [9] investigated the effect of content clustering based on temporal access patterns and found it effective in reducing latency, but they considered a single server environment and didn't study the more accurate spatial clustering. Padmanabhan and Mogul [23] proposed a pre-fetching algorithm using a dependency graph. When a page A is accessed, clients will pre-fetch a page B if the arc from A to B has a large weight in the dependency graph. Su *et al.* proposed a recursive density-based clustering algorithm for efficient information retrieval on the Web[26]. As in the previous work, our content clustering algorithms also try to identify groups of pages with similar access pattern. Unlike many previous works, which are based on analysis of individual client access patterns, we are interested in aggregated clients' access patterns, since content is replicated for aggregated clients. Also, we quantify the performance of various cluster-based replications by evaluating their impact on replication.

Moreover, we examine the stability of content clusters using *incremental clustering*. Incremental clustering has been studied in previous work, such as [8] and [32]. However, to the best our knowledge, none of the previous work looks at incremental clustering as a way to facilitate content replication and improve the access performance perceived by clients. We are among the first to examine clustering Web content for efficient replication, and use both replication performance and stability as the metrics for evaluation of content clustering.

3 Simulation Methodology

Throughout the paper, we use trace-driven simulations to evaluate the performance of various schemes. In this section, we describe the network topologies and Web traces we use for evaluation.

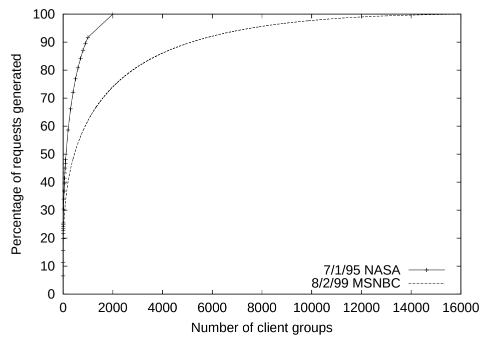


Figure 1. The CDF of the number of requests generated by the Web client groups defined by BGP prefixes for the MSNBC trace, and by domains for the NASA trace.

3.1 Network Topology

In our simulations, we use three random network topologies generated from the GT-ITM internetwork topology generator [31]: pure random, Waxman, and Transit-Stub. In the pure random model, nodes are randomly assigned to locations in a plane, with a uniform probability p of an edge added between a pair of nodes. The Waxman model also places nodes randomly, but creates an edge between a pair of node u and v with probability $P(u, v) = \alpha e^{(-d/\beta L)}$, where $d = |\overset{\Leftrightarrow}{u} \Leftrightarrow \overset{\Leftrightarrow}{v}|$, L is the maximum Euclidean distance between any two vertices, and $\alpha > 0$ and $\beta \leq 1$. The Transit-Stub model generates network topologies composed of interconnected transit and stub domains, and better reflects the hierarchical structure of real networks. We further experiment with various parameters in every topology model.

In addition to using synthetic topologies, we also construct an AS-level Internet topology using BGP routing data collected from a set of seven geographically-dispersed BGP peers in April 2000 [14]. Each BGP routing table entry specifies an AS path, AS_1, AS_2, ..., AS_n, $etc.$, to a destination address prefix block. We construct a graph using the AS paths, where individual clients and address prefix blocks are mapped to their corresponding AS nodes in the graph, and every AS pair has an edge with the weight being the shortest AS hop count between them.

Web Site	Period	Duration	# Requests avg - min - max	# Clients avg - min - max	# Client Groups avg - min - max
MSNBC	8/99 - 10/99	10 am-11 am	1.5M - 642K - 1.7M	129K - 69K - 150K	15.6K - 10K - 17K
NASA	7/95 - 8/95	All day	79K - 61K - 101K	5940 - 4781 - 7671	2378 - 1784 - 3011
WorldCup	5/98 - 7/98	All day	29M - 1M - 73M	103K - 13K - 218K	N/A

Table 1. Access logs used.

3.2 Web Workload

In our evaluation, we use the access logs collected at the MSNBC server site [20], as shown in Table 1. MSNBC is a large and popular commercial news site in the same category as CNN and ABCNews [1]. It is consistently ranked among the busiest sites in the Web [19]. For diversity, we also use the traces collected at NASA Kennedy Space Center in Florida [21] during 1995 and the WorldCup Web site in 1998 [4]. Table 1 shows the detailed trace information. The number of client groups is unavailable in the WorldCup trace because it anonymized all client IP addresses. As a result, we are unable to group clients to study their aggregated behavior for clustering. So we only use it for hot data stability analysis.

We use the access logs in the following way. When using the AS-level topology, we group clients in the traces based on their AS numbers. When using random topologies, we group the Web clients based on BGP prefixes [17] using the BGP tables from a BBNPlanet (Genuity) router [6]. For the NASA traces, since most entries in the traces contain host names, we group the clients based on their domains, which we define as the last two parts of the host names (e.g., a1.b1.com and a2.b1.com belong to the same domain). Figure 1 plots the CDF of the number of requests generated by Web client groups. As we can see, in the 8/2/99 MSNBC trace, the top 10, 100, 1000, 3000 groups account for 18.58%, 33.40%, 62.01%, and 81.26% of requests, respectively; in the 7/1/95 NASA trace, the top 10, 100, 1000 groups account for 25.41%, 48.02%, and 91.73% of requests, respectively.

We choose top 1000 client groups in the traces since they cover most of the requests (62-92%) and map them to 1000 nodes in the random topologies. Assigning a group C_i to a node P_i in the graph means that the weight of the node P_i is equal to the number of requests generated by the group C_i.

In our simulations, we assume that replicas can be placed on any node, where a node represents a popular IP cluster in the MSNBC traces, or a popular domain in the NASA trace. Given the rapid growth of CDN service providers, such as Akamai (which already has more than 11,000 servers in about 500 networks around the world [3]), we believe this is a realistic assumption. Moreover, for any URL, the first replica is always at the origin Web server (a randomly selected node), as in [18, 25]. However, including or excluding the original server as a replica is not a fundamental choice and has little impact on our results.

3.3 Performance Metric

We use the average retrieval cost as our performance metric, where the retrieval cost of a Web request is the sum of the costs of all edges along the path from the source to the replica from which the content is downloaded. In the synthetic topologies, the edge costs are generated by the GT-ITM topology generator. In the AS-level Internet topology, the edge costs are all 1, so the average retrieval cost represents the average number of AS hops a request traverses.

4 Stability of Hot Data

Many studies report that Web accesses follow the Zipf-like distribution [7], which are also exhibited by our traces. This indicates that there is large variation in the number of requests received by different Web pages, and it is important to focus on popular pages when doing replication. In order for replicating popular pages to be an effective approach, the popularity of the pages needs to be stable. In this section, we investigate this issue.

We analyze the stability of Web pages using the following two metrics: (i) the stability of Web page popularity rankings, as used in [24], and (ii) the stability of request coverage from (previous) popular Web pages. The latter is an important metric to quantify the effi-

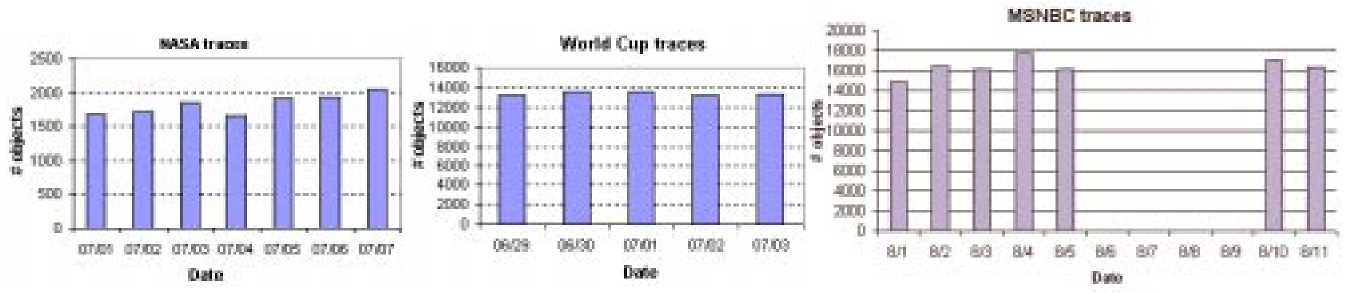
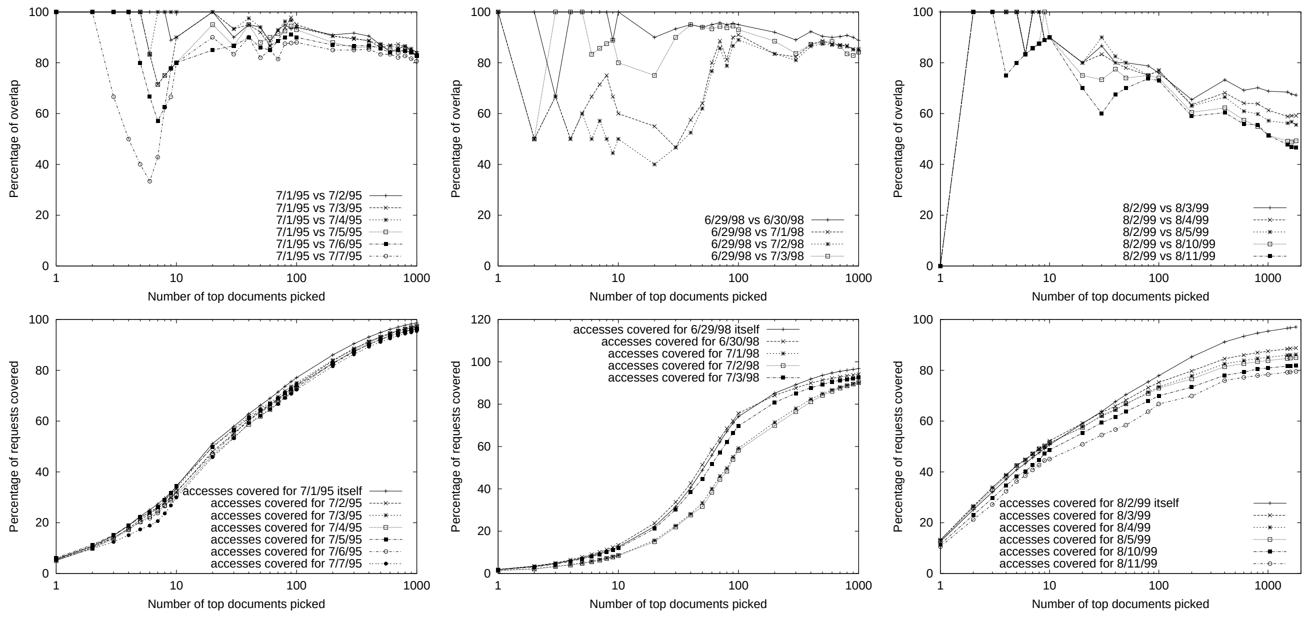


Figure 2. The number of URLs accessed in NASA, WorldCup, and MSNBC traces.



(a) 7 days of NASA traces (b) 5 days of WorldCup Traces (c) 6 days of MSNBC traces

Figure 3. Hot Web page stability of popularity ranking (top), and stability of access request coverage (bottom) with daily intervals.

ciency of pre-fetching/pushing of hot Web data. One of our interesting findings is that while the popularity ranking may fluctuate, the request coverage still remains stable.

We study the stability of both metrics in various time scales: within a month, a day, an hour, a minute, and a second. They show similar patterns, so we only present the results for the daily and monthly scale. Figures 2 and 4 show the number of unique URLs in the traces. Figures 3 and 5 show the stability results for the time gap being a few days and one month, respectively. In all the graphs on the top of Figures 3 and 5, the x-axis is the number of most popular documents picked (e.g., x = 10 means we pick the 10 most popular documents), and y-axis is the percentage of overlap. As we can see, the overlap is mostly over 60%, which indicates many documents are popular on both days. On the other hand, for the WorldCup site, which is event-driven and frequently has new content added, the overlap sometimes drops to 40%.

A natural question arises: whether the old hot documents can continue to cover the majority of requests as time evolves. The graphs on the bottom of Figure 3 and 5 shed light on this. Here we pick the hot documents from the first day, and plot the percentage of requests covered by these documents for the first day itself and for the following days. As we can see, the request coverage remains quite stable. The top 10% of objects on one day can cover over 80% requests for at least the subsequent week. We also find that the stability of content varies across different Web sites. For example, the stability period of WorldCup site is around a week, while the top 10% objects at the NASA site can continue to cover the majority of requests for two months.

Based on the above observations, we conclude that when performing replica placement, we only need to consider the top few URLs (e.g., 10%), as they account for most of the requests. Furthermore, since the request coverage of these top URLs remains stable for a long period (at least a week), it is reasonable to replicate based on previous access pattern, and change the provision infrequently. This helps to reduce the cost of replication, computation, and management. We will examine this issue further in Section 10.

5 Un-cooperative Pulling vs. Cooperative Pushing

Many CDN providers (e.g., Akamai [3] and Digital Island [11]) use un-cooperative pulling. In this case, CDN nodes (a.k.a. CDN (cache) servers or edge servers) serve as caches and pull content from the origin server when a cache miss occurs. There are various mechanisms that direct client requests to CDN nodes, such as DNS-based redirection, URL rewriting, HTTP redirection, etc. Figure 6 shows the CDN architecture using the DNS-based redirection [5], one of the most popular redirection schemes due to its transparency. The CDN name server does not record the location of replicas, thus a request is directed to a CDN node, only based on network connectivity and server load.

Several recent works proposed to proactively push content from the origin Web server to the CDN nodes according to users' access patterns, and have them cooperatively satisfy clients' requests [15, 25, 27]. We can adopt a similar CDN architecture as shown in Figure 6 to support such a cooperative push-based content distribution. First, the Web content server incrementally push contents based on their hyperlink structures and/or some access history collected by CDN name server (Section 10.2). The content server runs a "push" daemon, and advertise the replication to the CDN name server, which maintains the mapping between content and their replica locations. The mapping can be coarse (e.g., at the level of Web sites if replication is done in units of Web sites), or fine-grained (e.g., at the level of URLs if replication is done in units of URLs). The CDN name server redirect client's request to its closest replica.

In both models, the CDN edge servers are allowed to execute their cache replacement algorithms. That is, the mapping in cooperative push-based replication are soft-state. If the client cannot find the content in the redirected CDN edge server, either the client will ask the CDN name server for another replica, or the edge server pulls the content from the Web server, and replies to the client.

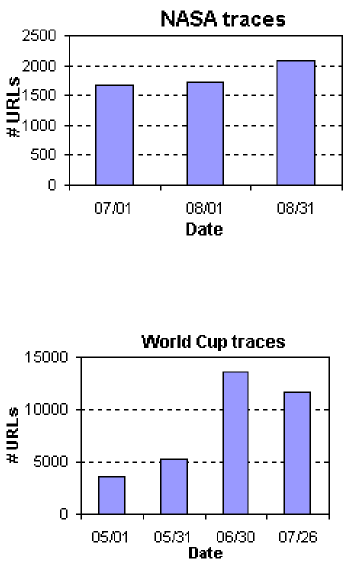


Figure 4. The number of URLs accessed in NASA and WorldCup traces.

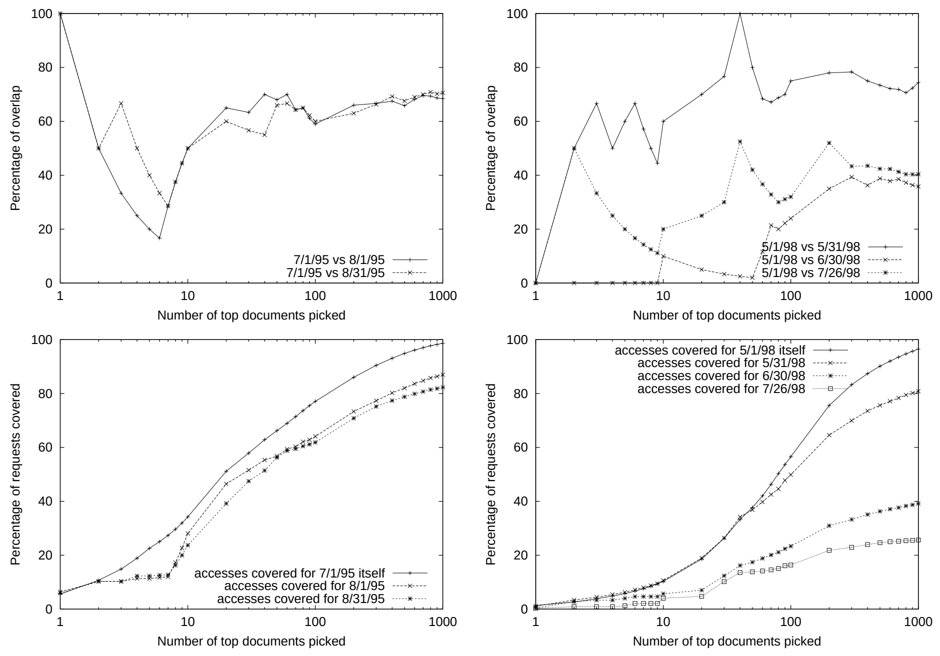


Figure 5. Hot Web page stability of popularity ranking (top), and stability of access request coverage (bottom) for NASA (left column) and WorldCup (right column) with monthly intervals.

5.1 Performance Comparison of Un-cooperative Pulling vs. Cooperative Pushing

Now we compare the performance between the un-cooperative pulling versus cooperative pushing using trace-driven simulation as follows. We apply the MSNBC trace on 10am - 11am of 8/2/1999 to a transit-stub topology. We choose the top 1000 URLs and top 1000 client groups with 964466 requests in total. In our evaluation, we assume that there is a CDN node located in every client group. In the un-cooperative pulling, we assume that a request is always redirected to the client's local CDN node and the latency between the client and its local CDN node is negligible (i.e., latencies incurred at step 5 and 8 shown in Figure 6 are 0), since they both belong to the same client group. In the cooperative push-based scheme, we replicate content in units of URLs to achieve similar clients' latency. Requests are directed to the closest replicas. (If the content is not replicated, the request goes to the origin server.) The details of replication algorithm will be explained in Section 7. As shown in Figure 6, the resolution steps (1-4) to locate a CDN node are the same for both schemes. Therefore we only need to compare the time for the "GET" request (step 5-8 in Figure 6).

Our results show that the un-cooperative pulling needs to out-source 120433 *URL replicas* to achieve an average latency of 79.2ms, where the URL replica is the total number of times URLs being replicated (e.g., URL_1 replicated 3 times, and URL_2 replicated 5 times, then the replication cost is 8 URL replicas). In comparison, the cooperative push-based scheme (per URL) uses only 5000 URL replicas to get a comparable latency (i.e., 77.9ms).

We also use the same access logs along with the corresponding modification logs to compare the cost of pushing updates to the replicas to maintain consistency. In our experiment, whenever a URL is modified, the Web server must notify all the nodes that contain the URL. Because the update size is unavailable in the trace, we use the total number of messages sent as our performance metric. With 11509 update events in 8/2/1999, the un-cooperative pulling uses 1349655 messages (about 1.3GB if we assume average update size is 1KB), while the cooperative per-URL based pushing only uses 54564 messages (53.3MB), about 4% update traffic, to achieve comparable user latency.

The results above show that cooperative pushing

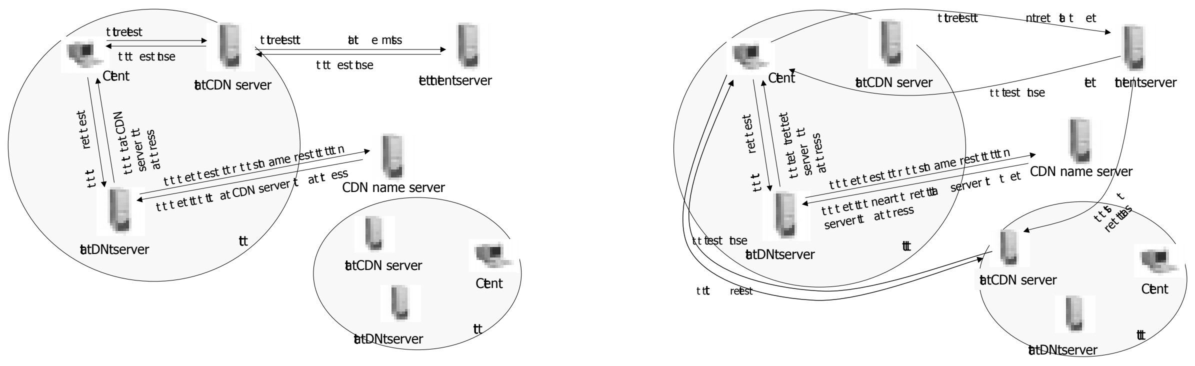


Figure 6. CDN architecture: un-cooperative pull-based (left), cooperative push-based (right)

yields much lower traffic, compared to the traditional un-cooperative pulling, which is currently in commercial use. One main reason for the traffic savings is that in the push-based scheme Web pages are strategically placed at selected locations, and a client's request is directed to the closest replica that contains the requested objects, while in the pulling scheme requests are directed to the closest replica that may or may not contain the requested objects. Of course, the performance benefit of the cooperative push-based scheme comes at the cost of the maintenance and storage overhead for content directory information. We will analyze the cost in Section 8.

In addition to the lower replication and update cost, the traffic cost and the management cost are controllable in the push-based scheme by clustering correlated content as we will show later, whereas both costs are demand-driven in the pulling. Moreover, for newly created content that has not been accessed, cooperative push-based replication is the only way to improve its availability and performance. We will study such performance benefits in Section 10.2.2.

Motivated by the observations, in the remainder of the paper we explore how to effectively push content to CDN nodes.

6 Problem Formulation

We describe the Web content placement problem as follows. Consider a popular Web site or a CDN hosting server, which aims to improve its performance by pushing its content to some hosting server nodes. The problem is to decide what content is to be replicated and where so that some objective function is optimized under a given traffic pattern and a set of resource constraints. The objective function can be to minimize either clients' latency, or loss rate, or total bandwidth consumption, or an overall cost function if each link is associated with a cost.

For Web content delivery, the major constraint in replication cost is the network access bandwidth at each Internet Data Center (IDC) to the backbone network. Current CDNs charge the content providers by the amount of bandwidth provisioned for delivering the contents [10]. Moreover, replication is not a one-time cost. Once a page is replicated, we need to pay additional resource to keep it consistent. Depending on the update and access rate, the cost of keeping things consistent can be high.

Based on the above observations, we formulate the Web content placement problem as follows. Given a set of URLs U and a set of locations L to which the URLs can be replicated, replicating a URL incurs one unit replication cost. A client j fetching a URL u from the ith replica of u located at $l_{u(i)}$ incurs a cost of $C_{j,l_{u(i)}}$, where $C_{j,l_{u(i)}}$ denotes the distance between j and $l_{u(i)}$. Depending on the metric we want to optimize, the distance between two nodes can reflect either the latency, or loss rate, or total bandwidth consumption or link cost. The problem is to find a replication strategy (i.e., for each URL u, we decide the set of locations $l_{u(i)}$ to which u is replicated) such that it minimizes

$$\sum_{j\in CL}(\sum_{u\in U_j}(min_{i\in\ R_u} C_{j,l_{u(i)}}))$$

subject to the constraint that the total replication cost (i.e., either $\sum_{u\in U}|u|$ assuming the replication cost of all URLs is the same, or $\sum_{u \in U} |u| * f(u)$ to take into account of different page sizes) is bounded by R, where $|u|$ is the number of different locations to which u is replicated, $f(u)$ is the size of URL u, CL is the set of clients, U_j is the set of URLs requested by the j-th client, R_u is the set of locations to which URL u has been replicated.

7 Replica Placement Per Web Site vs. Per URL

In this section, we examine if replication at a fine granularity can help to improve the performance for push-based scheme. Our performance metric is the total latency incurred for all the clients to fetch their requested documents, as recorded in the traces. We compare the performance of replicating all the hot data at a Web site as one unit (i.e., per Web site-based replication, see Algorithm 1) versus replicating content in units of individual URLs (i.e., per URL-based replication, see Algorithm 2).

For both approaches below, $totalURL$ is the number of distinct URLs of the Web site to be replicated, $currReplicationCost$ is the number of URL replicas deployed and $maxReplicationCost$ is the total number of URL replica threshold. For simplicity, we assume that all URLs are of the same size. The non-uniform nature of size distribution actually has little effect on the results as shown in Section 9.2.

```
  procedure GreedyPlacement_WebSite($maxReplicationCost$, $totalURL$)
1 Initially, all the URLs reside at the origin Web server, $currReplicationCost = totalURL$
2 while $currReplicationCost < maxReplicationCost$ do
3     foreach node i without the Web site replica do
4         Compute the clients' total latency reduction if the Web site is replicated to $i$ (denoted as $gain_i$)
      end
5     Choose node $j$ with maximal $gain_j$ and replicate the Web site to $j$
6     $currReplicationCost += totalURL$
  end
```

Algorithm 1: Greedy Replica Placement (Per Web site)

When replicating content in units of URLs, not all URLs have the same number of replicas. Given a fixed replication cost, we give a higher priority to URLs that yield more improvement in performance. Algorithm 2 uses greedy approach to achieve it: at each step, we choose the $< object, location >$ pair that gives the largest performance gain.

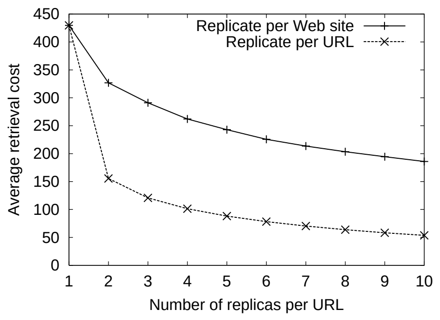
```
  procedure GreedyPlacement_URL($maxReplicationCost$, $totalURL$)
1 Initially, all the URLs reside at the origin Web server, $currReplicationCost = totalURL$
2 foreach URL u do
3     foreach node i do
4         Compute the clients' total latency reduction for accessing $u$ if $u$ is replicated to $i$ (denoted as $gain_{u_i}$)
      end
5     Choose node j with maximal $gain_{u_j}$, $best\_site_u = j$ and $max\_gain_u = gain_{u_j}$
  end
6 while $currReplicationCost < maxReplicationCost$ do
7     Choose URL $v$ with maximal $max\_gain_v$, and replicate $v$ to node $best\_site_v$
8     Repeat steps 3, 4 and 5 for $v$
9     $currReplicationCost++$
  end
```

Algorithm 2: Greedy Replica Placement (Per URL)

We will compare the computational cost of the two algorithms with clustering-based approach in Section 8. Figure 7 shows the performance gap between the per Web site-based replication and the per URL-based replication. The first replica is always at the origin Web server for in both schemes, as described in Section 3. In our simulation, we choose top 1000 URLs from the 08/02/99 MSNBC trace, covering 95% of requests, or top 300 URLs from the 07/01/95 NASA trace, covering 91% of requests. For the MSNBC trace, the per URL-based replication can consistently yield a 60-70% reduction in clients' latency; for the NASA trace, the improvement is 30-40%. The larger improvement in the MSNBC trace is likely due to the fact that requests in the MSNBC trace are more concentrated on a small number of pages, as reported in [24]. As a result, replicating the very hot data to more locations, which is allowed in the per URL-based scheme, is more beneficial.

8 Clustering Web Content

In the previous section, we have shown that a fine-grained replication scheme can reduce clients' latency by up to 60-70%. However since there are thousands of hot objects that need to be replicated, searching over all the possible $< object, location >$ combinations is prohibitive. In our simulations, it takes 102 hours to



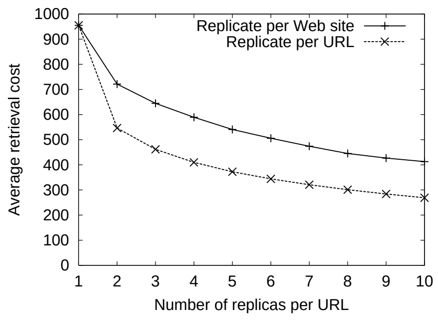


Figure 7. Performance of the per Web site-based replication vs. the per URL-based replication for 8/2/99 MSNBC trace (left) and 7/1/95 NASA trace (right)

come up with a replication strategy for 10 replicas per URL on a PII-400 low end server. Clearly it is too expensive for practical use. To achieve the benefit of the fine-grained replication while at reasonable computation and management cost, in this section, we investigate clustering Web content based on their access pattern, and replicate content in units of clusters.

At a high level, clustering enables us to smoothly tradeoff the computation and management cost for better clients' performance. The URL-based scheme is one extreme clustering: create a cluster for each URL. It can achieve good performance at the cost of high management overhead. In comparison, the Web server based scheme is another extreme: one cluster for each Web server. This approach is simple and has small management overhead, but its performance is 60-70% worse than the former approach, as shown in Section 7. We can smoothly tradeoff between the two by adjusting the number of clusters. This provides more flexibility and choices in CDN replication. Depending on the CDN provider's need, it can choose whichever point it find appropriate.

Below we quantify how clustering helps to reduce computation and management cost. Suppose there are N objects, and M (roughly $N/10$) hot objects to be put into K clusters ($K < M$). Assume on average there are R replicas/URL that can be distributed to S CDN servers (potential sites) to serve C clients. For the per cluster case, we not only record where each cluster is stored, but also keep track of the cluster to which each URL belongs. Note that even with hundreds of R and tens of thousands of M, it is quite trivial to store all the information (even for the per URL scheme).

On the other hand, the computation cost of the replication schemes is much higher, and becomes an important factor that determines the feasibility of the schemes in practice. To use the greedy algorithm for the per Web site placement, the computation complexity is $O(RSC)$ [25]. For the per URL placement using the greedy algorithm, the complexity is $O(MR \times (M + SC))$, given the algorithm in Section 7. Basically, there are MR iterations. And for each iteration, we have to choose the $< object, location >$ pair which will give the most performance gain from M candidates. After that, the next best location for that object and its cost gain need to be updated with $O(SC)$ computation cost. Similarly, the computational time is $O(KR \times (K + SC))$ for the per cluster replication algorithm. In addition, it has clustering cost $O(MK)$ in the spatial clustering discussed later. Assuming the placement adaptation frequency is f_p and the clustering frequency is f_c, Table 2 summarizes the management cost for the various replication schemes. As we will show in Section 10, the content clusters remain stable over time. Thus f_c is small, and the computational cost of clustering is negligible when compared with the cost of the replication.

One possible clustering scheme is to group URLs based on their directories. While simple, this approach may not capture correlations between URLs' access patterns for the following reasons. First, deciding where to place a URL is a manual and complicated process, since it is difficult to predict without consulting the actual traces whether two URLs are likely to be accessed together. Even with full knowledge about the contents of URLs, the decision is still heuristic since people have different interpretations of the same data. Also, most Web sites are organized with convenience

Replication Scheme	States to Maintain	Computation Cost
Per Web Site	$O(R)$	$f_p \times O(RSC)$
Per Cluster	$O(RK + M)$	$f_p \times O(KR \times (K + SC)) + f_c \times O(MK)$
Per URL	$O(RM)$	$f_p \times O(MR \times (M + SC))$

Table 2. Management cost for various scales of data replication, where $K < M$.

of maintenance in mind, and such organization does not correspond well to the actual correlation of URLs. For example, a Web site may place its HTML files in one directory and images in another, even though a HTML file is always accessed together with its linked images. Finally, it can be difficult to determine the appropriate directory level to separate the URLs.

We tested our hypothesis for the MSNBC trace on 8/1/1999: we cluster the top 1000 URLs using the 21 top level directories, and then use the greedy algorithms to place on average 3 replicas/URL (i.e., 3000 replicas in total, the same scenario we used evaluating other clustering schemes). Compared with the per Web site-based replication, it reduces latency only by 12% for pure random graph topology, and by 3.5% for transit-stub topology. Therefore the directory-based clustering only yields a marginal benefit over the Web site based replication for the MSNBC site; in comparison, as we will show later, clustering content based on access pattern can yield more significant performance improvement: 40% - 50% for the above configuration.

In the remaining of this section, we examine content clustering based on access patterns. We start by introducing our general clustering framework, and then describe the correlation distance metrics we use for clustering.

8.1 General Clustering Framework

Clustering data involves two steps. First, we define the correlation distance between every pair of URLs based on a certain correlation metric. Then given n URLs and their correlation distance, we apply standard clustering schemes to group them. We will describe our distance metrics in Section 8.2. Regardless of how the distance is defined, we can use the following clustering algorithms to group the data.

We explore two generic clustering methods. The first one aims to minimize the maximum diameter of all clusters while limiting the number of clusters. The diameter of cluster i is defined as the maximum distance between any pair of URLs in cluster i. It represents the worst-case correlation within that cluster. The second one aims to minimize the number of clusters while limiting the maximum diameter of all clusters.

1. Limit the number of clusters, then try to minimize the maximal diameter of all clusters. We use the classical K-split algorithm by T. F. Gonzalez [13]. It is a $O(NK)$ approximation algorithm, where N is the number of points and K is the number of clusters. And it guarantees solution within twice the optimal.

2. Limit the diameter of each cluster, and minimize the number of clusters. This can be reduced to the problem of finding cliques in a graph using the following algorithm: Let N denote the set of URLs to be clustered, and d denote the maximum diameter of a cluster. Build graph $G(V, E)$ such that $V = N$ and edge $(u, v) \in E \Leftrightarrow dist(u, v) < d$, $\forall u, v \in V$. We can choose d using some heuristics, e.g., a function of average distance over all URLs. Under this transformation, every cluster corresponds to exactly one clique present in the generated graph. Although the problem of partitioning graphs into cliques is NP-complete, we adopt the best approximation algorithm in [12] with time complexity $O(N^3)$.

We have applied both clustering algorithms, and got similar results. So in the interest of brevity, we present the results obtained from using the first clustering algorithm.

8.2 Correlation Distance

In this section, we describe the correlation distance we use. We explore three orthogonal distance metrics: one based on spatial locality, one based on temporal locality, and another based on popularity locality. The metrics can also be based on semantics, such as the hyperlink structures or XML tags in Web pages. We will examine the hyperlink structure for online incremental clustering in Section 10.2.2, and leave the clustering based on other metadata, such as XML tags, for future work.

8.2.1 Spatial Clustering

First, we look at clustering content based on their spatial locality in the access patterns. At a high level, we would like to cluster URLs that share similar access distribution across different regions. For example, two URLs that both receive the largest number of accesses from New York and Texas and both receive the least number of accesses from California may be clustered together.

We use BGP prefixes or domain names to partition the Internet into different regions as described in Section 3. We represent the access distribution of a URL using a spatial access vector, where the ith field denotes the number of accesses to the URL from the client group i. When there are L client groups, each URL is uniquely identified as a point in L-dimensional space. In our simulation, we use the top 1000 clusters (i.e., $L = 1000$), covering 70% - 92% of requests.

We define the correlation distance between URLs A and B in two ways: either (i) the Euclidean distance between the points in the L-dimension space that represent the access distribution of URL A and B, or (ii) the complement of *cosine vector similarity* of spatial access vector A and B.

$$\begin{aligned} correl\_dist(A,B) &= 1 \Leftrightarrow vector\_similarity(A,B) \\ &= 1 \Leftrightarrow \frac{\sum_{i=1}^{k} A_i \times B_i}{\sqrt{\sum_{i=1}^{k}(A_i)^2 \times \sum_{i=1}^{k}(B_i)^2}} \end{aligned} \quad (1)$$

Essentially, if we view each spatial access vector as an arrow in high-dimension space, the vector similarity gives the cosine of the angle formed by the two arrows.

8.2.2 Temporal Clustering

In this section, we examine temporal clustering. As its name suggests, the temporal clustering tries to cluster Web content based on temporal locality of the access pattern.

There are many ways to define temporal locality. One possibility is to divide the traces into n time slots, and assign a temporal access vector to each URL, where the element i is the number of accesses to that URL from the time slot i. Then we can use similar methods in spatial clustering to define the correlation distance. However, in our experiments we found that many URLs share similar temporal access vectors because of specific events, and they do not necessarily tend to be accessed together by the same client. One typical example is in the event-driven World-Cup trace, where the corresponding URLs in English and French have very similar temporal access patterns during game time, but as expected are almost never fetched together by any client groups.

To address this issue, we consider URLs are correlated only if they are generated in a short period by the same client. In particular, we extend the co-occurrence based clustering by Su *et al.* [26]. At a high-level, the algorithm divides requests from a client into variable length sessions, and only considers URLs requested together during a client's session to be related. We make the following enhancements: (i) we empirically determine the session boundary rather than choose an arbitrary time interval; (ii) we quantify the similarity in documents' temporal locality using the co-occurrence frequency.

Determine session boundaries: First, we need to determine user sessions, where a session refers to a sequence of requests initiated by a user without prolonged pauses in between. We apply the heuristic described in [2] to detect the session boundary: we consider a session has ended if it is idle for sufficiently long time (called *session-inactivity period*); and we empirically determine the session inactivity period as the knee point where the change in its value does not yield a significant change in the total number of sessions [2]. Both the MSNBC and NASA traces have the session-inactivity period as 10 - 15 minutes, and we choose 12 minutes in our simulations.

Correlation in temporal locality: We compute the correlation distance between any two URLs based on the co-occurrence frequency (see Algorithm 3). This reflects the similarity in their temporal locality and thus the likelihood of being retrieved together by a client. Assume that we partition the traces into p sessions. The number of co-occurrences of A and B in the session i is denoted as $f_i(A, B)$, which is calculated by counting the number of interleaving access pairs (not necessarily adjacent) for A and B.

```
    procedure TemporalCorrelationDistance()
1   foreach session with access sequence ($s_1, s_2, \ldots s_n$) do
2       for i = 1; i ≤ n-1; i++ do
3           for j = i+1; j ≤ n; j++ do
4               if $s_i \neq s_j$ then $f_i(s_i, s_j)$++; $f_i(s_i, s_j)$++;
5               else exit the inner for loop to avoid counting duplicate pairs
            end
        end
    end
6   foreach URL $A$ do compute the number of occurrences $o(A)$
7   foreach pair of URLs (A, B) do
8       Co-occurrence values $f(A, B) = \sum_{i=1}^{p} f_i(A, B)$
9       Co-occurrence frequency $c(A, B) = \frac{f(A,B)}{o(A)+o(B)}$
10      Correlation distance $correl\_dist(A, B) = 1 - c(A, B)$
    end
```

Algorithm 3: Temporal Correlation Distance Computation

Steps 2 to 5 of Algorithm 3 computes $f_i(A, B)$. For example, if the access sequence is "*ABCCA*" in session i. The interleaving access pairs for A and B are *AB* and *BA*, so $f_i(A, B) = f_i(B, A) = 2$. Similarly, $f_i(A, C) = f_i(C, A) = 3$, $f_i(B, C) = f_i(C, B) = 2$. Note that in Step 8 and 9, since $f(A, B)$ is symmetric, so is $c(A, B)$. Moreover, $0 \leq c(A, B) \leq 1$ and $c(A, A) = 1$. The larger the $c(A, B)$, the more closely correlated the two URLs are, and the more likely they are to be accessed together. Step 10 reflects the property that distance decreases with the increase in the correlation.

8.2.3 Popularity-based Clustering

Finally, we consider the approach of clustering URLs by their access frequency to examine whether the document popularity can capture the important locality information. We consider two metrics. The first correlation distance metric is defined as

$$correl\_dist(A, B) = |access\_freq(A) \Leftrightarrow access\_freq(B)|$$

The second distance metric is even simpler. If N URLs are to be clustered into K clusters, we sort them according to the total number of accesses, and place objects 1 .. $\lfloor \frac{N}{K} \rfloor$ into cluster 1, $\lfloor \frac{N}{K} \rfloor$+1 .. $\lfloor \frac{2N}{K} \rfloor$ into cluster 2, and so on.

We tested both metrics on MSNBC traces and they yield very similar results. Therefore we only use the simpler approach for evaluation in the rest of the paper. The popularity-based clustering only takes into account of the hit count, but does not consider either the time or space correlation between accesses.

8.2.4 Traces Collection for Clustering

The three clustering techniques all require access statistics, which can be collected at CDN name servers or CDN servers. The popularity-based clustering needs the least amount of information: only the hit counts of the popular Web objects are sufficient. In comparison, the temporal clustering requires the most fine-grained information – the number of co-occurrences of popular Web objects, which can be calculated based on the access time and source IP address for each request. The spatial clustering is in between the two: for each popular Web object, it needs to know how many requests are generated from each popular client group, where the client groups are determined using BGP prefixes collected over widely dispersed routers [17].

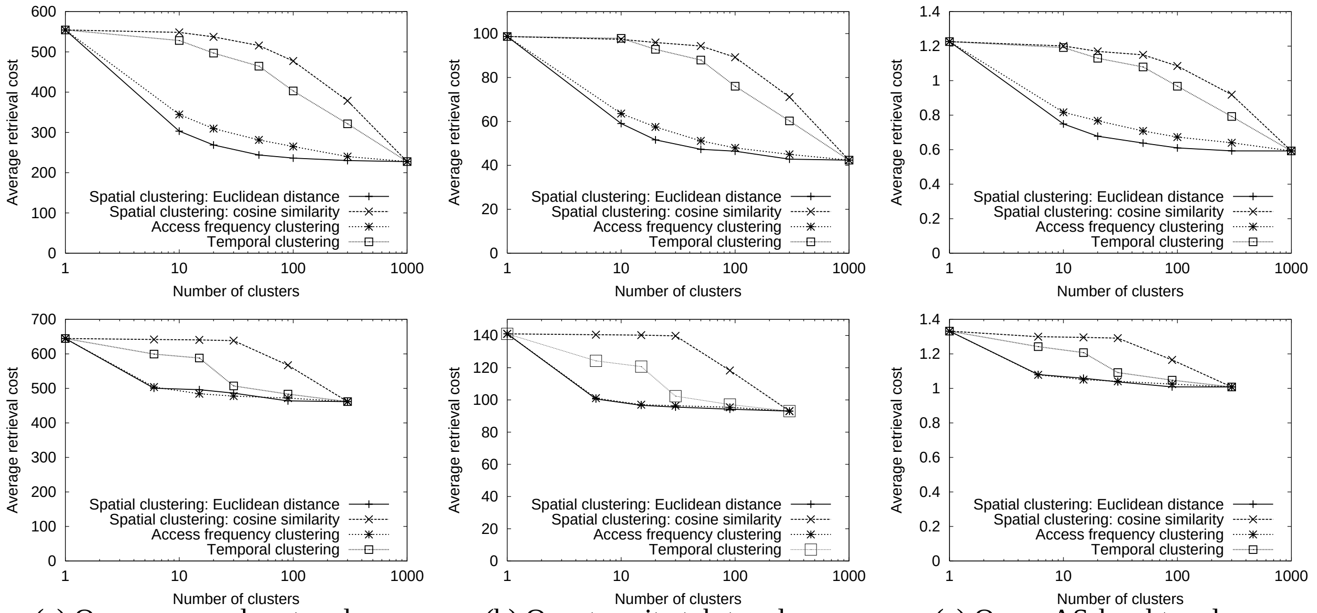
9 Performance of Cluster-based Replication

In this section, we evaluate the performance of different clustering algorithms on a variety of network topologies using the real Web server traces.

9.1 Replication Performance Comparison of Various Clustering Schemes

First we compare the performance of various cluster-based algorithms. In our simulations, we use the top 1000 URLs from the MSNBC traces covering 95% of requests, and the top 300 URLs from the NASA trace covering 91% of requests. The replication algorithm we use is similar to Algorithm 2 in Section 7. For iteration step 7, we choose the $< cluster, location >$ pair that gives the largest performance gain per URL.

Figure 8 compares the performance of various clustering schemes for the 8/2/1999 MSNBC trace and 7/1/1995 NASA trace. The starting points of all the clustering performance curves represent the single cluster case, which corresponds to the per Web



(a) On a pure random topology (b) On a transit-stub topology (c) On an AS-level topology

Figure 8. Performance of various clustering approaches for MSNBC 8/2/1999 trace with averagely 5 replicas/URL (top) and for NASA 7/1/1995 trace with averagely 3 replicas/URL (bottom) on various topologies

site-based replication. The end points represent the per URL-based replication, another extreme scenario where each URL is a cluster.

As we can see, the clustering schemes are efficient. Even with the constraint of a small number of clusters (i.e., 1% - 2% of the number of Web pages), spatial clustering based on Euclidean distance between access vectors and popularity-based clustering achieve performance close to that of the per URL-based replication, with much lower management cost (see Section 8). Spatial clustering with cosine similarity and temporal clustering do not perform as well. It is interesting that although the popularity-based clustering does not capture variations in individual clients' access patterns, it achieves comparable and sometimes better performance than the more fine-grained approaches. A possible reason is that many popular documents are globally popular [30], and access frequency becomes the most important metric that captures different documents' access pattern.

The relative rankings of various schemes are consistent across different network topologies. The performance difference is smaller in the AS topology, because the distance between pairs of nodes is not as widely distributed as in the other topologies.

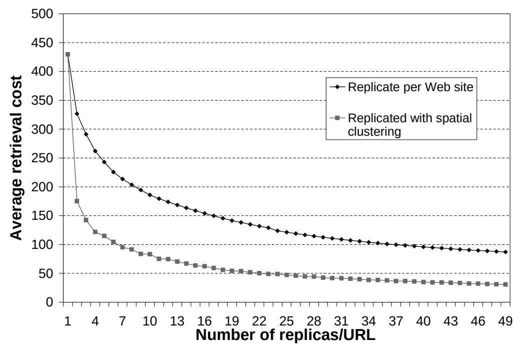


Figure 9. Performance of cluster-based replication for MSNBC 8/2/1999 trace (in 20 clusters) with up to 50 replica/URL on pure random topology

We also evaluate the performance of the cluster-based replication by varying the replication cost (i.e., the average number of replicas/URL). Figure 9 shows the performance results when we use the spatial access vector based clustering scheme and 20 content clusters. As before, the cluster-based scheme out-performs the per Web site scheme by over 50%. As expected the performance gap between per Web site and per cluster replication decreases as the number of replicas per URL increases. Compared to the per URL-based replication, the cluster-based replication is more scalable:

Per site	10 clusters	50 clusters	300 clusters	Per URL
132.7	108.1	84.7	81.3	80.4

Table 3. Average retrieval cost with non-uniform file size

it reduces running time by over 20 times, and reduces the amount of state by orders of magnitude.

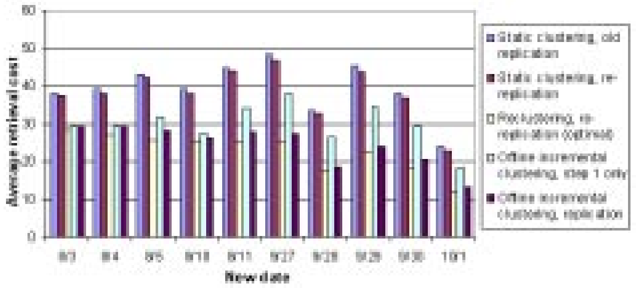
9.2 Effects of Non-Uniform File size

So far, we assume each replicated URL consumes one unit of replication cost. In this section, we compute the replication cost by taking into account of different URL sizes. The cost of replicating a URL is its file size. We modify Algorithm 2 in Section 7 so that in iteration step 7, we choose the $< cluster, location >$ pair that gives the largest performance gain per byte.

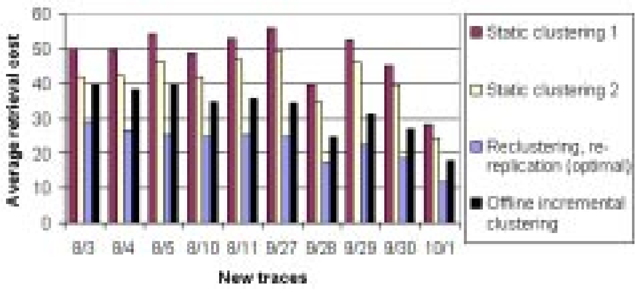
We ran the experiments using the top 1000 URLs of the MSNBC trace on 8/2/1999. Table 3 shows the performance of the Euclidean distance based spatial clustering with an average of 3 replicas/URL on a transit stub topology. The results exhibit a similar trend as those obtained under the assumption of uniform URL size: the per URL-based replication out-performs the per Web site-based replication by 40%, and the cluster-based schemes (50 clusters) achieve similar performance as the per URL-based replication (1000 clusters) with only about 5% management cost if we ignore the cost for clustering.

10 Incremental Clustering

In the previous sections, we have presented cluster-based replication, and showed it is flexible and can smoothly tradeoff replication cost for users' perceived performance. To adapt to changes in user access pattern, in this section we examine incremental clusterings. We start by studying the performance of static clustering, which re-distributes the existing content clusters (without changing the clusters). Then we look at incremental clustering, which gradually puts new popular URLs to existing clusters and replicates them. We compare both static and incremental clusterings with the complete re-clustering and redistribution, which is regarded as the optimal case.



(a) Cluster based on the Euclidean distance of spatial access vector.



(b) Cluster based on the access frequency.

Figure 10. Stability analysis of the per cluster replication for MSNBC 1999 traces with 8/2/99 as birth trace (averagely 5 replicas/URL).

10.1 Static Clustering

It is important to determine the frequency of cluster perturbation and redistribution. If the clients' interested URLs and access patterns change very fast, a fine-grained replication scheme that takes into account of how a client retrieves multiple URLs together may require frequent adaptation. The extra maintenance and clustering cost may dictate that the per Web site replication approach be used instead. To investigate whether this would be a serious concern, we perform the following experiments on the MSNBC traces: *birth trace* and *new trace*, where the birth trace and new trace are access traces for Day 1 and Day 2, respectively (Day 2 follows Day 1 either immediately or a few days apart).

1. We use the birth trace to determine content clusters and the placement of the clusters. According to the content placement, we compute the total latency incurred for the requests in the new trace. Note that accesses to URLs that are not included in the birth trace have to go to the origin

Web server, potentially incurring a higher cost. It is likely to be infrequent since the set of popular content is stable over time as reported in Section 4. This scheme simulates one with non-adaptation in either content clustering or content placement.

2. We use the birth trace to cluster Web content, and use the new trace to determine the placement of clusters and evaluate performance. As before, accesses to the URLs that are not included in the birth trace have to go to the origin Web server. This simulates a scheme that adapts the placement decision to the new access pattern, while not changing the way in which content are clustered.

3. We use new trace to determine content clusters and the placement of clusters, and evaluate performance. This simulates a scheme with adaptation in both content clustering and placement decision. It basically approximates the best performance we can achieve using the up-to-date information of the access pattern by completely re-clustering and re-replicating the content.

We study the stability for both spatial clustering with Euclidean distance (referred as SC) and access frequency (popularity) based clustering (referred as AFC), both of which yield the best performance as shown in Section 9. In our experiments, we use 8/2/99 trace as the birth trace, and use 8/3/99, 8/4/99, 8/5/99, 8/10/99, 8/11/99, 9/27/99, 9/28/99, 9/29/99, 9/30/99 and 10/1/99 traces as the new traces. We simulate on pure-random, Waxman, transit-stub, and AS topologies. The results for different topologies are similar, and below we only present the results from transit-stub topologies in the interest of space.

We use the following simulation configuration throughout this section unless otherwise specified. We use the top 1000 client groups from the 8/2/99 MSNBC traces, and the top 1000 URLs in our simulation. We cluster URLs into 20 groups using SC or AFC. The top 1000 client groups during 8/3/99 - 10/1/99 have over 70% overlap with those on 8/2/99.

As shown in Figure 10, using past workload information performs significantly worse than using the actual workload. The performance of AFC is slightly worse than that of SC. In addition, as we would expect, the difference in the performance gap increases with the time gap. The redistribution of old clusters based on the new trace does not help for SC, and helps slightly for AFC. The increase in the clients' latency is largely due to the creation of new contents, which have to be fetched from the origin site according to our assumption. (The numbers of new URLs are shown in row 1 of Table 4.) This is a pessimistic assumption, and gives an upper bound on the cost of users' requests. In the next section, we will use various incremental clustering schemes to address this issue.

10.2 Incremental Clustering

In this section, we examine how to incrementally add new documents to existing clusters without much perturbation. First, we formulate the problem, and set up framework for generic incremental clustering. Then we investigate both online and offline incremental clustering. The former predicts access patterns of new objects based on hyperlink structures, while the latter assumes such access information is available. Finally, we compare their performance and management cost with the complete re-clustering and re-distribution.

10.2.1 Problem Formulation

We define the problem of *incremental clustering* for distributed replication system as follows. Given N URLs, initially they are partitioned into K clusters and replicated to various locations to minimize the total distance cost of all clients' requests. The total number of URL replicas created is T. After some time, V of the original objects become cold when the number of requests to them drops below certain threshold, while W new popular objects emerge, and need to be clustered and replicated to achieve good performance. To prevent the number of object replicas T from increasing dramatically, we can either explicitly reclaim the cold object replicas or implicitly replace them through replacement policies such as LRU and LFU. For simplicity, we adopt the latter approach. The replication cost is defined as the total number of replicas distributed for new popular objects.

One possible approach is to completely re-cluster and re-replicate the new ($N \Leftrightarrow V + W$) objects as the

Row	Date of new trace in 1999	8/3	8/4	8/5	8/10	8/11	9/27	9/28	9/29	9/30	10/1
1	# of new popular URLs	315	389	431	489	483	539	538	530	526	523
2	# of cold URL replicas freed	948	1205	1391	1606	1582	1772	1805	1748	1761	1739
3	# of orphan URLs when using $\lvert\overleftrightarrow{v}_{new} \Leftrightarrow \overleftrightarrow{v}_c\rvert > r$	0	0	2	1	1	6	4	6	8	6
4	# of orphan URLs when using $\lvert\overleftrightarrow{v}_{new} \Leftrightarrow \overleftrightarrow{v}_c\rvert > r_{max}$	0	0	2	0	1	6	4	6	7	5
5	# of new URL replicas deployed for non-orphan URLs ($\lvert\overleftrightarrow{v}_{new} \Leftrightarrow \overleftrightarrow{v}_c\rvert \leq r$)	983	1091	1050	1521	1503	1618	1621	1610	1592	1551
6	# of new clusters generated for orphan URLs ($\lvert\overleftrightarrow{v}_{new} \Leftrightarrow \overleftrightarrow{v}_c\rvert > r$)	0	0	2	1	1	3	3	3	3	3
7	# of URL replicas deployed for orphan URLs ($\lvert\overleftrightarrow{v}_{new} \Leftrightarrow \overleftrightarrow{v}_c\rvert > r$): row 2 - row 5 if row 2 $>$ row 5	0	0	341	85	79	154	184	138	169	188
8	# of new URL replicas deployed (Access frequency clustering)	1329	1492	1499	1742	1574	2087	1774	1973	1936	2133

Table 4. Statistics and cost evaluation for offline incremental clustering. Using MSNBC traces with 8/2/99 as birth trace, 20 clusters, and averagely 5 replicas/URL. Results for clustering based on SC (row 3 - 7) and AFC (row 8).

third scheme described in Section 10.1. However this approach is undesirable in practice, because it requires reshuffling the replicated objects and re-building the content directory, which incurs extra replication traffic and management cost. Therefore our goal is to find a replication strategy that balances the tradeoff between replication cost and clients' performance while with little disturbance to the existing replicas.

We can use the following two steps for incremental clustering:

1. STEP 1: If the correlation between the new URL and an existing content cluster exceeds a threshold, add the new URL to the cluster that has the highest correlation.

2. STEP 2: If there are still new URLs left (referred as *orphan URLs*), create new clusters and find replica locations for them.

10.2.2 Online Incremental Clustering

Pushing newly created documents are useful during unexpected flash crowds events, such as disaster. Without clients' access information, we predict access pattern of new documents using the following two methods based on hyperlink structures.

1. Cluster URLs based on their parent URLs, where we say URL a is URL b's parent if a has a hyperlink pointing to b. However, since many URLs point back to the root index page, the root page is not included in any children cluster since its popularity differs significantly.

2. Cluster URLs based on their *hyperlink depth*. The hyperlink depth of URL o is defined as the *smallest* number of hyperlinks needed to traverse before reaching o, starting from the root page of the Web server.

In our evaluation, we use WebReaper 9.0 [29] to crawl http://www.msnbc.com/ at 8am, 10am and 1pm (PDT time) respectively on 05/03/2002. (The reason for using different data set here is because we do not have the crawled MSNBC content in 1999.) Given a URL, WebReaper downloads and parses the page to look for links to other pages and objects. It will then extract this list of sub-links and download them.

This process continues recursively until a pre-defined depth is reached. We set the crawled hyperlink depth to 3, and ignore any URLs outside www.msnbc.com except the outsourced images. Since we also consider the URLs pointed by all the crawled documents, our analysis includes all pages within 4 hyperlink distance away from the root page. Clustering based on the hyperlink depth generates 4 clusters, e.g., depth = 1, 2, 3, and 4 (exclusive of the root page). The access logs do not record accesses to image files, such as .gif and .jpg. We have the access information for the remaining URLs, whose statistics are shown in Table 5. In general, about 60% of these URLs are accessed in two-hour period after crawling.

To measure the popularity correlation within a cluster, we define *access frequency span* (in short, af_span) as follows

$$af\_span = \frac{standard\ deviation\ of\ access\ frequency}{average\ access\ frequency}$$

We have MSNBC access logs from 8am - 12pm and 1pm - 3pm on 5/3/2002. For every hour during 8am - 12pm and 1pm - 3pm, we use the most recently crawled files to cluster content, and then use the access frequency recorded in the corresponding access logs to compute af_span for each cluster. We also compute the average, 10 percentile and 90 percentile of af_span in all clusters, and compare them with af_span when treating all URLs (except the root) as a single cluster.

The results are shown in Figure 11. As we can see, both clustering methods show much better popularity correlation (i.e., smaller af_span) than the non-clustered case, and method 1 consistently outperforms method 2. Based on the observation, we design an online incremental clustering algorithm as follows. For each new URL o, assign it to the existing cluster that has the largest number of URLs sharing the same parent URL as o (i.e., the largest number of sibling URLs). If there are ties, we are conservative, and pick the cluster that has the largest number of replicas. Note that o may have multiple parents, so we consider all the children of its parents as its siblings except the root page. When a new URL o is assigned to cluster c, then we replicate o to all the replicas to which cluster c has been replicated.

We simulate them on a 1000-node transit-stub topology as follows. First, among all the URLs crawled at 8am, 2496 of them were accessed during 8am - 10am. We use AFC to cluster and replicate them based on the 8am - 10am access logs, with 5 replicas per URL on average. Among those new URLs that appear in the 10am crawling, but not in the 8am crawling, 16 of them were accessed during 10am - 12pm. Some of them were quite popular, receiving 33262 requests in total during 10am - 12pm. We use the online incremental clustering algorithms above to cluster and replicate the 16 new URLs with a replication cost of 406 URL replicas. This yields an average retrieval cost of 56. We also apply the static AFC by using 10am - 12pm access logs, and completely re-clustering and re-replicating all these 2512 (2496 + 16) URLs, with 5 replicas per URL on average. As it requires information about future workload and completely re-clustering and re-replicating content, it serves as the optimal case, and yields an average retrieval cost of 26.2 for the 16 new URLs. However, if the new URLs are not pushed but only cached after it is accessed, the average retrieval cost becomes 457; and if we replicate the 16 new URLs to random places using the same replication cost as in the online incremental clustering (406 URL replicas), the average retrieval cost becomes 259.

These results show that the online incremental clustering and replication cuts the retrieval cost by 4.6 times compared to random pushing, and by 8 times compared to no push. Compared to the optimal case, the retrieval cost doubles, but since it requires no access history nor complete re-clustering or replication, such performance is quite good.

10.2.3 Offline Incremental Clustering

Now we study offline incremental clusterings, which use access history as input.

STEP 1: In the SC clustering, when creating clusters for the birth trace, we record the center and diameter of each cluster. Given a cluster U with p URLs, each URL u_i is identified by its spatial access vector $\overleftrightarrow{v_i}$ and $correlation\_distance(u_i, u_j) = |\overleftrightarrow{v_i} \Leftrightarrow \overleftrightarrow{v_j}|$. We define the center c as $\frac{\sum_{i=1}^{p} \overrightarrow{v_i}}{p}$. The radius r is $max_i(|\overleftrightarrow{v_c} \Leftrightarrow \overleftrightarrow{v_i}|)$, which is the maximum Euclidean distance between the center and any URL in U. For

Crawled time on 5/3/2002	8am	10am	1pm
# of crawled URLs (non image files)	4016	4019	4082
# of URL clusters (clustering with the same parent URL)	531	535	633

Table 5. Statistics and clustering of crawled MSNBC traces

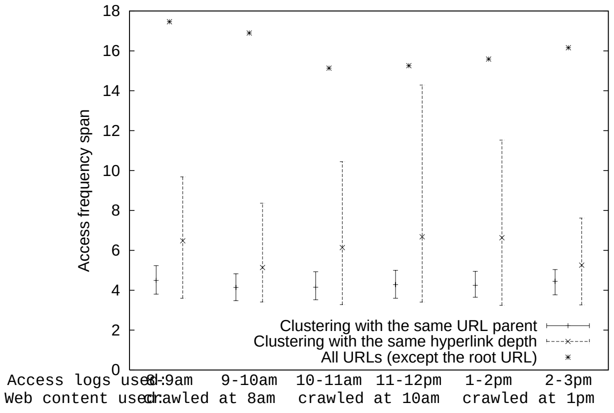


Figure 11. Popularity correlation analysis for semantics-based clustering. The error bar shows the average, 10 and 90 percentile of af_span.

each new URL $\overleftrightarrow{v_{new}}$, we add it to an existing cluster U whose center c is closest to $\overleftrightarrow{v_{new}}$, if either $|\overleftrightarrow{v_{new}} \Leftrightarrow \overleftrightarrow{v_c}| < r$ or $|\overleftrightarrow{v_{new}} \Leftrightarrow \overleftrightarrow{v_c}| < r_{max}$ is satisfied, where r is the radius of cluster U, and r_{max} is the maximum radius of all clusters.

Our analysis of MSNBC traces shows that most of the new URLs can find their homes in old clusters (as shown in rows 3 and 4 of Table 4); this implies the spatial access vector of most URLs are quite stable, even after about two months. Furthermore, the difference between using $|\overleftrightarrow{v_{new}} \Leftrightarrow \overleftrightarrow{v_c}| < r$ and $|\overleftrightarrow{v_{new}} \Leftrightarrow \overleftrightarrow{v_c}| < r_{max}$ is insignificant. So we consider the former in the remaining of this section. Once a new URL is assigned to a cluster, the URL is replicated to all the replicas to which the cluster has been replicated. Row 5 of Table 4 shows the number of new URL replicas.

In the AFC clustering, the correlation between URLs is computed using their ranks in access frequency. Given K clusters sorted in decreasing order of popularity, a new URL of rank i (in the new trace) is assigned to $\lceil \frac{i}{K} \rceil$th cluster. In this case, all new URLs can be assigned to one of existing clusters, and step 2 is unnecessary.

Figure 10 shows the performance after the completion of Step 1. As we can see, incremental clustering has improvement over static clustering by 20% for

```
   procedure IncrementalClusteringReplication_OrphanURLs()
1  Compute and record the biggest diameter $d$ of original clusters
   from the birth trace
2  Use limit diameter clustering (Section 8) to cluster the orphan
   URLs into $K'$ clusters with diameter $d$
3  $l$ = number of cold URL replicas freed - number of URL replicas
   deployed for non-orphan URLs in Step 1.
4  if $l > 0$ then replicate the $K'$ clusters with $l$ replicas
5  else Replicate the $K'$ clusters with $l'$ replicas, where $l'$ = number
   of orphan URLs $\times$ average number of replicas per URL
```

Algorithm 4: Incremental Clustering and Replication for Orphan URLs (Spatial Clustering)

SC, and 30-40% for AFC. At this stage, SC and AFC have similar performance. But notice that AFC has replicated all the new URLs while SC still has orphan URLs for the next step. In addition, AFC deploys more new URL replicas (row 8 of Table 4) than SC (row 5 of Table 4) at this stage.

STEP 2: We further improve the performance by clustering and replicating the orphan URLs. Our goal is (1) to maintain the worst-case correlation of existing clusters after adding new ones, and (2) to prevent the total number of URL replicas from increasing dramatically due to replication of new URLs. Step 2 only applies to SC, and we use Algorithm 4.

Row 6 and 7 in Table 4 show the number of new clusters generated by orphan URLs and the number of URL replicas deployed for the orphan URLs. Figure 10 (top) shows the performance of SC after Step 2 is completed. As the figure shows, SC out-performs AFC by about 20% after step 2, and achieves comparable performance to complete re-clustering and re-replication, while using only 30 - 40% of the replication cost as in the complete re-clustering and re-replication (which is 4000 URL replicas, the total number of replicas (1000 URLs $\times$ 5 replicas/URL) except those on the Web server).

To summarize, in this section, we study online and offline incremental clusterings, and show they are very effective in improving users' perceived performance with small replication cost.

11 Conclusion

In this paper, we explore how to efficiently push content to CDN nodes for cooperative access. Using trace-driven simulations, we show that replicating content in units of URLs out-performs replicating in units of Web sites by 60 - 70%. However such a fine-grained replication is too expensive to be practical for large Web sites. To address the issue, we propose three clustering schemes based on spatial, temporal, and popularity locality to group the Web documents and replicate them in units of clusters. Our evaluations based on various topologies and Web server traces show that we can achieve performance comparable to the per URL-based replication at only 1% - 2% of the management cost. To adapt to changes in users' access patterns, we explore both offline and online incremental clusterings, and show that the offline clusterings yield close to the performance of the complete re-clustering while at much lower overhead. The online incremental clustering and replication reduce the retrieval cost by 4.6 - 8 times compared to no replication and the random replication, so it is very useful to avoid flash crowd and improve document availability.

Based on our results, we recommend CDN operators to use the cooperative clustering-based replication. More specifically, for the content with access histories, we can group them through either spatial clustering or popularity-based clustering, and replicate them in units of clusters. To reduce replication cost and management overhead, incremental clustering is preferred. For the content without access histories (e.g., newly created ones), we can incrementally add them to existing content clusters based on hyperlink structures, and push them to the locations to which the cluster has been replicated. This online incremental cluster-based replication is very useful to improve document availability during flash crowds.

In conclusion, our main contributions include (i) cluster-based replication schemes to smoothly trade off management and computation cost for better clients' performance in a CDN environment, (ii) an incremental clustering framework to adapt to changes in users' access patterns, and (iii) an online popularity prediction scheme based on hyperlink structures.

References

[1] ABCNEWS. http://www.abcnews.com.

[2] ADYA, A., BAHL, P., AND QIU, L. Analyzing browse patterns of mobile clients. In *Proceedings of SIGCOMM Internet Measurement Workshop 2001* (Nov. 2001).

[3] AKAMAI. http://www.akamai.com.

[4] ARLITT, M., AND JIN, T. Workload characterization of the 1998 world cup web site. HP Tech Report HPL-1999-35(R.1).

[5] BARBIR, A., CAIN, B., DOUGLIS, F., GREEN, M., HOFMANN, M., NAIR, R., POTTER, D., AND SPATSCHECK, O. Known CN request-routing mechanisms. IETF draft, http://www.ietf.org/internet-drafts/draft-cain-cdnp-known-request-routing-04.txt.

[6] BBNPLANET. telnet://ner-routes.bbnplanet.net.

[7] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., AND SHENKER, S. Web caching and zipf-like distributions: Evidence and implications. In *Proc. of INFOCOMM '99* (Mar 1999).

[8] CHARIKAR, M., CHEKURI, C., FEDER, T., AND MOTWANI, R. Incremental clustering and dynamic information retrieval. In *Proceedings of STOC* (May 1997).

[9] COHEN, E., KRISHNAMURTHY, B., AND REXFORD, J. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of ACM SIGCOMM* (Sep 1998).

[10] DANZIG, P. Former V. P. Techonology of Akamai, Personal communication, 2001.

[11] DIGITALISLAND. http://www.digitalisland.com.

[12] EDACHERY, J., SEN, A., AND BRANDENBURG, F. J. Graph clustering using distance-k cliques. In *Proc. of Graph Drawing* (Sep 1999).

[13] GONZALEZ, T. F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science 38* (1985), 293–306.

[14] IPMA project. http://www.merit.edu/ipma.

[15] JAMIN, S., JIN, C., KURC, A., RAZ, D., AND SHAVITT, Y. Constrained mirror placement on the Internet. In *Proceedings of IEEE INFOCOM'2001* (April 2001).

[16] KAUFMAN, L., AND ROUSSEEUW, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[17] KRISHNAMURTHY, B., AND WANG, J. On network-aware clustering of web clients. In *Proc. of SIGCOMM '2000* (Aug. 2000).

[18] LI, B., GOLIN, M. J., ITALIANO, G. F., DENG, X., AND SOHRABY, K. On the optimal placement of Web proxies in the Internet. In *Proceedings of IEEE INFOCOM'99* (Mar. 1999).

[19] MEDIAMETRIX. http://www.mediametrix.com.

[20] MSNBC. http://www.msnbc.com.

[21] NASA kennedy space center server traces. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[22] NG, R., AND HAN, J. Efficient and effective clustering methods for data mining. In *Proc. of Intl. Conf. on VLDB* (1994).

[23] PADMANABHAN, V. N., AND MOGUL, J. C. Using predictive prefetching to improve world wide web latency. In *ACM SIGCOMM Computer Communication Review* (July 1996).

[24] PADMANABHAN, V. N., AND QIU, L. Content and access dynamics of a busy Web site: Findings and implications. In *Proc. of SIGCOMM '2000* (Aug 2000).

[25] QIU, L., PADMANABHAN, V. N., AND VOELKER, G. M. On the placement of Web server replica. In *Proceedings of IEEE INFOCOM'2001* (April 2001).

[26] SU, Z., YANG, Q., ZHANG, H., XU, X., AND HU, Y. Correlation-based document clustering using web. In *Proceedings of the 34th HAWAII International conference on System Sciences* (January 2001).

[27] VENKATARAMANI, A., YALAGANDULA, P., KOKKU, R., SHARIF, S., AND DAHLIN, M. The potential costs and benefits of long term prefetching for content distribution. In *Proc. of Web Content Caching and Distribution Workshop 2001* (2001).

[28] VOORHEES, E. M. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing & Management, 22* (1986), 465–476.

[29] WEBREAPER. http://www.webreaper.net.

[30] WOLMAN, A., VOELKER, G. M., SHARMA, N., CARDWELL, N., BROWN, M., LANDRAY, T., PINNEL, D., KARLIN, A. R., AND LEVY, H. M. Organization-based analysis of web-object sharing and caching. In *USENIX Symposium on Internet Technologies and Systems* (1999).

[31] ZEGURA, E., CALVERT, K., AND BHATTACHARJEE, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (1996).

[32] ZHANG, T., RAMAKRISHNAN, R., AND LIVNY, M. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of SIGMOD* (1996).