Clustering 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 pulling of Web contents used by commercial CDNs with the cooperative pushing. Our results show that the latter can achieve comparable users' perceived performance with only 4 - 5% of replication and update traffic compared to the former scheme. Therefore 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. However, 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 large Web server traces to evaluate their performance. Our results show that the cluster-based replication achieves 40 - 60% improvement over the per Web site based replication. In addition, by adjusting the number of clusters, we can smoothly trade off the management and computation cost for better client performance.

To adapt to changes in users' access patterns, we also explore incremental clusterings that adaptively add new documents to the existing 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 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.

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 pushing) yields significant performance benefit than pulling content and passively caching them solely driven by users' request sequence (i.e., un-cooperative pulling).

A number of previous works [1, 2] 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) at a low computational 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.

First, we compare the traditional un-cooperative pulling vs. cooperative pushing. Simulations on a variety of network topologies using real Web traces show that the latter scheme can yield comparable clients' latency while only using about 4-5% of the replication and update cost compared to the former scheme.

Motivated by the observation, we explore how to efficiently push content to CDN nodes. We compare the performance between the per Web site-based replication (all hot data) versus the per (hot) URL-based replication, and show 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. To address the issue, we propose several clustering algorithms to group Web content based on their correlation, and replicate objects in units of content clusters. Simulation 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. 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 with 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 Sec. 2, and describe our simulation methodology in Sec. 3. We compare the un-cooperative pulling vs. the cooperative pushing in Sec. 4. Then we formulate the push-based content placement problem in Sec. 5, and compare the per Web site-based replication with the per URL-based replication in Sec. 6. We describe content clustering techniques for efficient replication in Sec. 7, and evaluate their performance in Sec. 8. In Sec. 9, we examine offline and online incremental clusterings. Finally we conclude in Sec. 10.

2 Related Work

A number of research efforts have studied the problem of placing Web server replicas (or 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 [3]. While an interesting first step, it has an important limitation that the Internet topology is not a tree. More recent studies [1, 2], based on evaluation using real traces and topologies, have independently reported that a greedy placement algorithm can provide content distribution networks with close-to-optimal performance.

There is considerable work done in data clustering, such as K-means [4], HAC [5], CLANRNS [6], etc. In the Web research community, there have been many interesting research studies on clustering Web content or identifying related Web pages for pre-fetching, information retrieval, and Web page organization, etc. Cohen *et al.* [7] 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/frequency clustering. Su *et al.* proposed a recursive density-based clustering algorithm for efficient information retrieval on the Web [8]. As in the previous work, our content clustering algorithms also try to identify groups of pages with similar access patterns. 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. In addition, 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 [9] and [10]. However, to the best of our knowledge, none of the previous work looks at incremental clustering as a way to facilitate content replication and improve clients' perceived performance. 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.

3.1 Network Topology

In our simulations, we use three random network topology models in the GT-ITM internetwork topology generator [11]: pure random, Waxman, and Transit-Stub. We further experiment with various parameters for each topology model. Refer to [12] for details.

In addition to using synthetic topologies, we also construct an AS-level Internet topology using BGP routing data collected from seven geographically-dispersed BGP peers in April 2000 [13]. 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, where individual clients 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.

3.2 Web Workload

In our evaluation, we use the access logs collected at the MSNBC server site [14], which is consistently ranked among the busiest sites in the Web [15]. For diversity, we also use the traces collected at NASA Kennedy Space Center [16]. Table 1 shows the detailed trace information.

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]. 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).

In [12], we show that hot data remain stable to cover the majority of requests as time evolves. For instance, the top 10% of objects on one day can cover over 80% requests for at least the subsequent week. Therefore it is cost-effective to only replicate *hot* content, which is used throughput this paper.

We choose top 1000 client groups in the traces since they cover most of the requests (62-92%) [12], and map them to 1000 nodes in the topologies. Assigning a group C_i to a node P_i in the graph means that the weight of P_i is equal to the number of requests generated by the group C_i.

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

Table 1. Access logs used.

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 traces. Given the rapid growth of CDN service providers, e.g., Akamai (which has more than 11,000 servers in about 500 worldwide networks [18]), 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 [3, 2]. 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 Un-cooperative Pull vs. Cooperative Push

Many CDN providers (e.g., Akamai [18] and Digital Island [19]) use un-cooperative pulling and do not replicate content until there is an access. In this case, the CDN name server does not record the location of replicas, and a request is directed to a CDN node only based on network connectivity and server load. So the CDN nodes serve as caches and pull content from the origin server when a cache miss occurs, regardless of how the content has been replicated.

On the other hand, 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 [1, 2, 20, 12]. Our trace-driven simulation shows that the cooperative pushing can yield comparable clients' latency while only using about 4-5% of the replication and update cost compared to un-cooperative pulling. The lower cost in the pushing scheme is mainly due to the strategic placement of replicas and cooperation among the replicas. Refer to [12] for further details.

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 pull-based scheme. Moreover, for newly created content that has not been accessed, the cooperative pushing is the only way to improve document availability and performance. We will study such performance benefits in Section 9.2.2.

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

5 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. Since network access bandwidth is scarce resource, we want to optimize our objective function while bounding the replication cost.

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 a 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

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

subject to the constraint that the total replication cost is bounded by R, where 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. (The total replication cost is 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 URL sizes, where $|u|$ is the number of different locations to which u is replicated, $f(u)$ is the size of URL u.)

6 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. 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 simplicity, we assume the replication costs of all URLs are the same. We can easily incorporate dif-

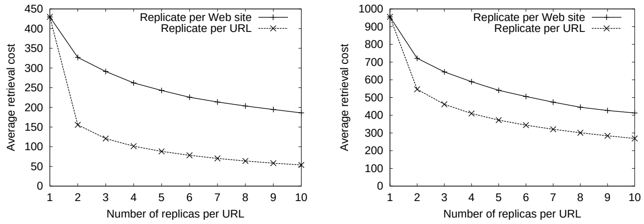


Figure 1. Performance of per Web site-based replication vs. per URL-based scheme for 8/2/99 MSNBC trace (left) and 7/1/95 NASA trace (right)

ferent URL sizes by modifying the Algorithm 2 to give preference to the URL that gives the largest performance gain per byte. In [12], we show the performance results of different URL sizes are similar.

In both algorithms, $totalURL$ is the number of distinct URLs of the Web site to be replicated, $currRepCost$ is the current number of URL replicas deployed, and $maxRepCost$ is the total number of URL replicas that can be deployed.

When replicating content in units of URLs, different URLs have different 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_WebSite($maxRepCost, totalURL$)
1 Initially, all the URLs reside at the origin Web server, $currRepCost = totalURL$
2 **while** $currRepCost < maxRepCost$ **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     $currRepCost += totalURL$
end

Algorithm 1: Greedy Replica Placement (Per site)

Figure 1 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 in both schemes, as described in Sec. 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 [21]. As a result, replicating the very hot data to more locations, which is allowed in the per URL-based scheme, is more beneficial.

procedure GreedyPlacement_URL($maxRepCost, totalURL$)
1 Initially, all the URLs reside at the origin Web server, $currRepCost = totalURL$
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** $currRepCost < maxRepCost$ **do**
7   Choose URL v with maximal max_gain_v, replicate v to $best\_site_v$
8   Repeat steps 3, 4 and 5 for v
9     $currRepCost++$
end

Algorithm 2: Greedy Replica Placement (Per URL)

7 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 come up with a replication strategy for 10 replicas per URL on a PII-400 low end server. This approach is too expensive for practical use even using high end servers. To achieve the benefit of the fine-grained replication 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 per URL-based replication 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 per Web site-based replication is another extreme: one cluster for each Web site. While it is easy to manage, its performance is much worse than the former approach, as shown in Section 6. 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 operating 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 to serve C clients. In the per cluster-based replication, 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. The storage cost of the per URL based replication is also manageable.

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. The computational complexities of Algorithm 1 and Algorithm 2 are $O(RSC)$ [2] and $O(MR \times (M + SC))$ [12], respectively. Similarly, the complexity of the cluster-based replication algorithm is $O(KR \times (K + SC))$. There is an additional clus-

Rep Scheme	Manage States	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. Overhead comparison ($K < M$)

tering cost, which varies with the clustering algorithm that is used. 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 Sec.9, the content clusters remain stable for at least one week. Therefore f_c is small, and the computational cost of clustering is negligible compared to the cost of the replication.

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 metrics we use for clustering.

7.1 General Clustering Framework

Clustering data involves two steps. First, we define 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 7.2. 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 method 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. We use the classical K-split algorithm [22]. It is a $O(NK)$ approximation algorithm, where N is the number of points and K is the number of clusters. It guarantees a solution within twice the optimal. The second method aims to minimize the number of clusters while limiting the maximum diameter of all clusters. This problem is NP-complete, and we adopt the best approximation algorithm in [23] 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.

7.2 Correlation Distance

In this section, we explore three orthogonal correlation distance metrics, which are based on spatial locality, temporal locality, and popularity, respectively. We can also use correlation metrics based on semantics, such as hyperlink structures or XML tags in Web pages. We will examine hyperlink structures for online incremental clustering in Sec. 9.2.2, and leave the clustering based on other metadata, such as XML tags, for future work. Another possibility is to group URLs by the directories they reside. But we find that the performance is quite poor [12], because the directory structures do not correlate well with the access patterns.

7.2.1 Spatial Clustering First, we look at clustering content based on the spatial locality in the access patterns. 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 i-th client group. Given 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 distributions of URL A and B, or (ii) the complement of *cosine vector similarity* of spatial access vector A and B.

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

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

7.2.2 Temporal Clustering In this section, we examine temporal clustering, which clusters Web content based on temporal locality of the access pattern. We try various ways to define the temporal locality [12], and only show the one that yields the best results.

Basically, URLs are considered to be correlated only if they are requested in a short period by the same client. In particular, we extend the co-occurrence based clustering by Su *et al.* [8]. 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 pro-longed pauses in between. We apply the heuristic described in [24] to detect the session boundary. Both the MSNBC and NASA traces have the session-inactivity period of 10 - 15 minutes, so 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. 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.

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 the distance decreases as the correlation increases.

procedure TemporalCorrelationDistance()
1 **foreach** *session with access sequence* $(s_1, s_2, \ldots s_n)$ **do**
2   **for** *i = 1; i* $\leq$ *n-1; i++* **do**
3   **for** *j = i+1; j* $\leq$ *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

7.2.3 Popularity-based Clustering

Finally, we consider the approach of clustering URLs by their access frequency. We consider two metrics. The first correlation distance metric is defined as

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

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

7.3 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. In comparison, the temporal clustering requires the most fine-grained information – the number of co-occurrences of popular 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].

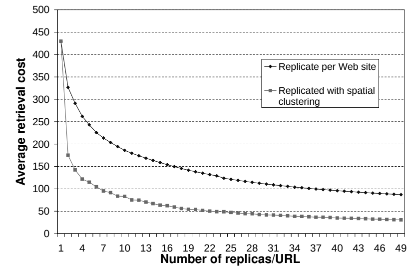


Figure 3. Performance of cluster-based replication for MSNBC 8/2/1999 trace (20 clusters) with up to 50 replicas/URL on pure random topology

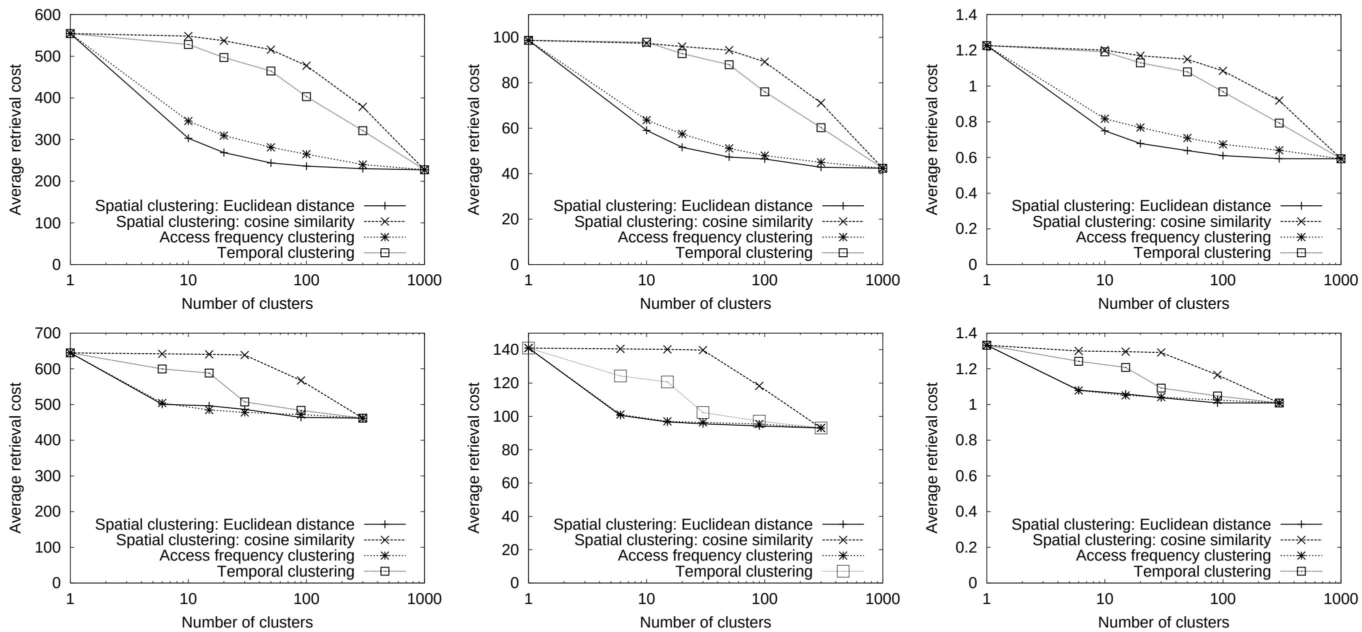
8 Performance of Cluster-based Replication

In this section, we evaluate the performance of different clustering algorithms on a variety of network topologies using the real Web server traces. 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 6. In the iteration step 7, we choose the $< cluster, location >$ pair that gives the largest performance gain per URL.

Figure 2 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 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, at much lower management cost (see Sec. 7). 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 [25], 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.

We also evaluate the performance of the cluster-based replication by varying the replication cost (i.e., the average number of replicas/URL). Figure 3 shows the performance results when we use the spatial access vector based clustering scheme and 20 content clusters. As before, the cluster-



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

Figure 2. 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

based scheme out-performs the per Web site scheme by over 50%. As expected, the performance gap between the per Web site and the 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: it reduces running time by over 20 times, and reduces the amount of state by orders of magnitude.

9 Incremental Clusterings

To adapt to changes in users' 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 optimal case, i.e., the complete re-clustering and redistribution.

9.1 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 considers 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 evaluate three methods, as shown in Table 3 using 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).

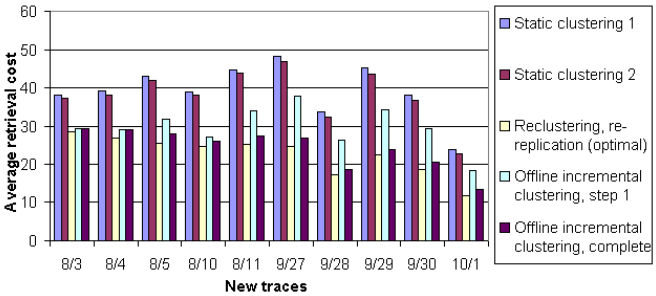
Methods	Static 1	Static 2	Optimal
Traces used for clustering	birth	birth	new
Traces used for replication	birth	new	new

Table 3. Static and optimal clustering schemes

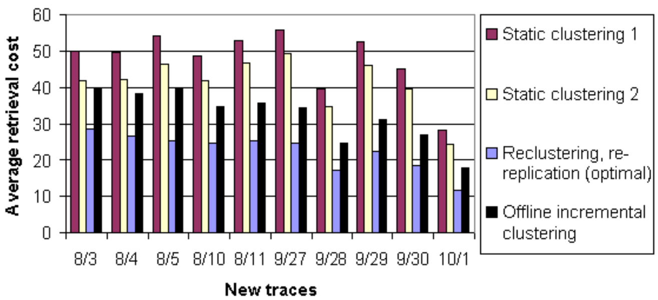
Note that in the static 1 and static 2 methods, accesses to the URLs that are not included in the birth trace have to go to the origin Web server, potentially incurring a higher cost. All three methods are evaluated using new traces. We consider the spatial clustering based on Euclidean distance (referred as SC) and popularity (i.e., access frequency) based clustering (referred as AFC), the two with the best performance in Sec. 8. 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.

We use the following simulation configuration throughout this section unless otherwise specified. We choose 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 - 10/1 have over 70% overlap with those on 8/2.

As shown in Figure 4, using the 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



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



(b) Cluster based on the access frequency.

Figure 4. Stability analysis of the per cluster replication for MSNBC 1999 traces with 8/2/99 as birth trace (averagely 5 replicas/URL).

fetched from the origin site according to our assumption. (The numbers of new URLs are shown in row 1 of Table 4.) In the next section, we will use various incremental clusterings to address this issue.

9.2 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 clusterings. 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.

9.2.1 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 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 a certain threshold, while W new popular objects emerge, and need to be clustered and replicated to achieve good performance. To prevent the number of replicas T from increasing dramatically, we can either explicitly reclaim the cold object replicas or implicitly replace them through policies such as

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

Table 5. Statistics of crawled MSNBC traces

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 $(N - V + W)$ objects as the third scheme described in Section 9.1. However this approach is undesirable in practice, because it requires re-shuffling 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 and management cost versus clients' performance.

Incremental clustering takes the following two steps:

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 with it.

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

9.2.2 Online Incremental Clustering Pushing newly created documents are useful during unexpected flash crowds and disasters. 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.
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 [26] to crawl http://www.msnbc.com/ at 8am, 10am and 1pm (PDT time) respectively on 05/03/2002. Given a URL, the WebReaper downloads and parses the page. Then it recursively downloads the pages pointed by the URL until a pre-defined hyperlink depth is reached. We set the depth to be 3 in our experiment. We 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.

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 ($\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert > r$)	0	0	2	1	1	6	4	6	8	6
4	# of new URL replicas, non-orphan	983	1091	1050	1521	1503	1618	1621	1610	1592	1551
5	# of new clusters, orphan URLs	0	0	2	1	1	3	3	3	3	3
6	# of URL replicas placed for orphan URLs: row 2 - row 4 if row 2 bigger	0	0	341	85	79	154	184	138	169	188
7	# of new URL replicas placed	1329	1492	1499	1742	1574	2087	1774	1973	1936	2133

Table 4. Statistics 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 (rows 3 - 6) and AFC (row 7).

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

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

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.

The results are shown in Figure 5. As we can see, both clustering methods show much better popularity correlation (i.e., smaller af_span) than treating all URLs (except the root) as a single cluster. Method 1 consistently out-performs 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 with 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, we replicate o to all the replicas to which cluster c has been replicated.

We simulate the approach 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.

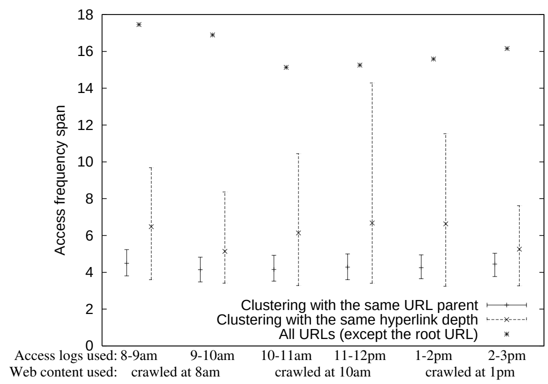


Figure 5. Popularity correlation analysis for semantics-based clustering. The error bar shows the average, 10 and 90 percentile of af_span.

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.

9.2.3 Offline Incremental Clustering

Now we study offline incremental clusterings, with 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 $\overrightarrow{v_i}$ and $correlation\_distance(u_i, u_j) = \lvert\overrightarrow{v_i} - \overrightarrow{v_j}\rvert$. We define the center c as $\frac{\sum_{i=1}^{p}\overrightarrow{v_i}}{p}$. The radius r is $max_i(\lvert\overrightarrow{v_c} - \overrightarrow{v_i}\rvert)$, which is the maximum Euclidean distance between the center and any URL in U. For each new URL $\overrightarrow{v_{new}}$, we add it to an existing cluster U whose center c is closest to $\overrightarrow{v_{new}}$ if $\lvert\overrightarrow{v_{new}} - \overrightarrow{v_c}\rvert < r$, where r is the radius of cluster U.

Our analysis of MSNBC traces shows that most of the new URLs can find homes in the existing clusters (row 3 of Table 4); this implies the spatial access vector of most URLs are quite stable, even after about two months. 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 4 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 4 shows the performance after the completion of Step 1. As we can see, incremental clustering has improvement over static clustering by 20% for 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 7 of Table 4) than SC (row 4 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 (see detailed algorithms in [12]).

Rows 5 and 6 in Table 4 show the number of new clusters generated by orphan URLs and the number of URL replicas deployed for the orphan URLs. As Figure 4 (top) 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 compared to the complete re-clustering and re-replication. (The total replication cost of the latter scheme is 4000 URL replicas: 1000 URLs $\times$ 5 replicas/URL, except 1000 URL replicas residing at the origin Web server.)

10 Conclusion

In this paper, we explore how to efficiently push content to CDN nodes for efficient 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%. To address the scalability issue of such a fine-grained replication, we explore several clustering schemes 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. Furthermore, we examine incremental clusterings to adapt to changes in user access patterns.

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 where the cluster has been replicated. This online incremental clustering 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] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *Proc. of INFOCOM*, 2001.

[2] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of Web server replica," in *Proc. of IEEE INFOCOM*, 2001.

[3] B. Li et al., "On the optimal placement of Web proxies in the Internet," in *Proc. of IEEE INFOCOM*, 1999.

[4] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.

[5] E. M. Voorhees, "Implementing agglomerative hierarchical clustering algorithms for use in document retrieval," *Information Processing & Management*, , no. 22, pp. 465–476, 1986.

[6] R. Ng and J. Han, "Efficient and effective clustering methods for data mining," in *Proc. of VLDB*, 1994.

[7] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving end-to-end performance of the Web using server volumes and proxy filters," in *Proc. of ACM SIGCOMM*, 1998.

[8] Z. Su et al., "Correlation-based document clustering using Web," in *Proc. of the Int. conf. on System Sciences*, 2001.

[9] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," in *Proceedings of STOC*, May 1997.

[10] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proceedings of SIGMOD*, 1996.

[11] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. of IEEE INFOCOM*, 1996.

[12] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. Katz, "Clustering web content for efficient replication," in *UCB/CSD Tech Report No. CSD-02-1193*, 2002.

[13] IPMA Project, "http://www.merit.edu/ipma".

[14] MSNBC, "http://www.msnbc.com".

[15] MediaMetrix, "http://www.mediametrix.com".

[16] NASA Kennedy Space Center Server Traces, "http://ita.ee.lbl.gov/html/contrib/nasa-http.html".

[17] B. Krishnamurthy and J. Wang, "On network-aware clustering of Web clients," in *Proc. of ACM SIGCOMM*, 2000.

[18] Akamai, "http://www.akamai.com".

[19] DigitalIsland, "http://www.digitalisland.com".

[20] A. Venkataramani et al., "The potential costs and benefits of long term prefetching for content distribution," in *Proc. of Web Content Caching and Distribution Workshop*, 2001.

[21] V. N. Padmanabhan and L. Qiu, "Content and access dynamics of a busy Web site: Findings and implications," in *Proc. of ACM SIGCOMM*, 2000.

[22] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[23] J. Edachery, A. Sen, and F. Brandenburg, "Graph clustering using distance-k cliques," in *Proc. of Graph Drawing*, 1999.

[24] A. Adya, P. Bahl, and L. Qiu, "Analyzing browse patterns of mobile clients," in *Proc. of SIGCOMM IMW*, 2001.

[25] Alec Wolman et al., "Organization-based analysis of web-object sharing and caching," in *Proc. of USITS*, 1999.

[26] WebReaper, "http://www.webreaper.net".