On the Placement of Web Server Replicas

Lili Qiu, Cornell University
Venkata N. Padmanabhan, Microsoft Research
Geoffrey M. Voelker, U.C. San Diego

Abstract—**Recently there has been an increasing deployment of content distribution networks (CDNs) that offer hosting services to Web content providers. CDNs deploy a set of servers distributed throughout the Internet and replicate provider content across these servers for better performance and availability than centralized provider servers. Existing work on CDNs has primarily focused on techniques for efficiently redirecting user requests to appropriate CDN servers to reduce request latency and balance load. However, little attention has been given to the development of placement strategies for Web server replicas to further improve CDN performance.**

In this paper, we explore the problem of Web server replica placement in detail. We develop several placement algorithms that use workload information, such as client latency and request rates, to make informed placement decisions. We then evaluate the placement algorithms using both synthetic and real network topologies, as well as Web server traces, and show that the placement of Web replicas is crucial to CDN performance. We also address a number of practical issues when using these algorithms, such as their sensitivity to imperfect knowledge about client workload and network topology, the stability of the input data, and methods for obtaining the input.

Keywords—**World Wide Web, replication, replica placement algorithm, content distribution network (CDN).**

I. Introduction

With the explosive growth of the World Wide Web, popular Web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. To address the need, there has been an increasing trend toward outsourcing content distribution to commercial hosting services such as Akamai [2], Exodus [12], Digital Island [11], etc. Existing work on CDNs has primarily focused on techniques for efficiently redirecting user requests to appropriate CDN servers to reduce request latency and balance load. However, little attention has been given to the development of placement strategies for Web server replicas to further improve CDN performance. Placement strategies are important because appropriate placement of server replicas benefits content providers by reducing latency for their clients, and benefits ISPs by reducing bandwidth consumption.

In this paper, we propose several algorithms that can automate the server placement decision. More specifically, we consider the following scenario. A popular Web site aims to improve its performance (e.g., reducing its clients' perceived latency) by pushing its content to some hosting services. The problem is to choose M replicas (or hosting services) among N potential sites ($N > M$) such that some objective function is optimized under a given traffic pattern. The objective function can be minimizing either its clients' latency, or its total bandwidth consumption, or an overall cost function if each link is associated with a cost.

In our study, we assume that each client uses a single replica (of course, multiple clients can use the same replica). In other words, a client gets all of its content from the same replica. So our analysis of replica placement focuses on the *traffic load* generated by the clients while ignoring what content is actually downloaded by clients. While our assumption is not entirely realistic – for example, a CDN such as Akamai would, in general, have *partial* replicas and direct clients to different replicas depending on what content is accessed – it enables us to project into the future when falling storage costs might make it feasible for each replica to be a *complete* replica. In such a setting, a client may well be directed to a single replica for most or all of its accesses.

We evaluate the performance of the various placement algorithms by simulating their behavior on synthetic and real network topologies and several access traces from large commercial and government Web servers. As far as we know, this is the first experimental study on this subject. We also address a number of practical issues when using these algorithms online in a content distribution network, and study the sensitivity of the placement algorithms to imperfect information about client workload characteristics. Based upon our results, we conclude that a greedy algorithm for solving the Web server replica placement problem can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and therefore is a viable algorithm for use in the general Internet environment where workload information will always be imperfect.

The rest of the paper is organized as follows. In Section II, we survey previous work. We describe graph theoretic formulations of the replica placement problem in Section III, and present a number of placement algorithms in Section IV. Then in Section V and Section VI, we describe our simulation methodology and performance results. In Section VII, we discuss a number of practical issues when using the algorithms. We then conclude in Section VIII.

II. Previous Work

There has been a considerable amount of research on Web performance, ranging from Web workload characterization [3], [4], [23] to developing techniques to enhance Web performance. Two primary techniques for enhancing Web performance are caching and replication. Previous work has studied many aspects of caching and replication, such as object routing, object distribution, object selection, inter-replica or inter-proxy communication, and policy management [24]. However, less attention has been given to the placement of Web proxies or Web replicas. The only prior work on the placement problem that we know of is [16] by Li *et al.* They approached the proxy placement problem with the assumption that the underlying network topologies are trees, and modeled it as a dynamic programming problem. Although an interesting first step, this approach has a number of limitations. First, the Internet topology is not a tree, and the paper does not evaluate how well the dynamic programming algorithm based on tree-topologies works for Internet-like topologies. Our evaluation using real traces and topologies (in Section VI) shows that, although the assumption of a tree topology makes it possible to obtain an optimal solution to the placement problem when the constraints are satisfied (i.e., the topology is actually a tree, and the clients can only direct requests to proxies on its path toward the Web server, but not to sibling proxies), in a more general setting it does not perform as well as the heuristics that work in general graph topologies. Moreover, its high computational complexity ($O(N^3M^2)$ for choosing M proxies among N potential sites) prevents its practical use in topologies with thousands of nodes.

Jamin *et al.* examined the placement problem for Internet instrumentation in [14]. They investigated both graph theoretic methods and heuristics for instrumenting the Internet to obtain distance maps. They showed that an Internet distance map service based on their placement techniques (including the placement heuristics that do not require full topological knowledge) can offer useful hints for server selection by clients.

III. Graph Theoretic Approaches

In this section we review two graph theoretic approaches that can help us determine the number and the placement of Web replicas given the network topology and the users' demands. In the following, we use the terms facilities, centers, and replicas synonymously. We study two variants of the center placement problem: one is the facility location problem, and the other is the minimum K-median problem. Both problems are NP-hard [22]. However, there are constant-factor approximation algorithms for the metric variants of both problems, where the metric variants require that the distance function c is non-negative, symmetric, and satisfies the triangle inequality.

A. Facility Location Problem

The facility location problem is defined as follows. Given a set of locations i at which facilities may be built, building a facility at location i incurs a cost of f_i. Each client j must be assigned to one facility, incurring a cost of $d_j c_{ij}$ where d_j denotes the demand of the node j, and c_{ij} denotes the distance between i and j. The objective is to find a solution (i.e., both the number of facilities and the locations of the facilities) of the minimum total cost.

There have been a number of approximation algorithms developed for this NP-hard problem in the metric space. Throughout the paper, a *ρ-approximation algorithm* is a polynomial-time algorithm that always finds a feasible solution with an objective function value within a factor of ρ of optimal. The best approximation algorithm known today was developed by Charikar & Guha [6], who gave a 1.728-approximation algorithm.

B. Minimum K-Median Problem

The minimum K-median problem is stated as follows. Given n points, we must select K of these to be centers (facilities), and then assign each input point j to the selected center that is closest to it. If location j is assigned to a center i, we incur a cost $d_j c_{ij}$. The goal is to select the K centers so as to minimize the sum of the assignment costs. The main difference between the K-median and facility location problems is that, in K-median, there are no costs for opening centers. Instead, a number K is specified as an input that is an upper bound on the number centers that can be opened. Recently, Charikar and Guha [6] gave a 4-approximation algorithm for this problem in the metric space.

C. Capacitated Versions

The formulations of the facility location problem and minimum K-median problem given above do not constrain the amount of service that can be provided at any center. There are capacitated variants that do constrain the service capacity at centers, requiring that each facility serve no more requests than the capacity defined at that location. The worst-case performance bound for the capacitated variants are considerably worse than for the non-capacitated versions [8], [7].

Depending on different constraints and cost functions to be optimized, replica placement can be formulated as either an uncapacitated/capacitated facility location problem, or an uncapacitated/capacitated minimum K-median problem.

D. Summary

In the rest of this paper, we consider the formulation of the uncapacitated minimum K-median problem. That is, we restrict the maximum number of replicas, but do not restrict the number of requests served by each replica. We believe that this is a reasonable formulation because increasing the number of replica sites is significantly more difficult than increasing the capacity of a site. The maximum number of replicas is usually given a priori for cost and administrative reasons, whereas the capacity constraint on the replica can be overcome by adding more machines. [1]

We also ignore the cost of placing replicas for the following reasons. If our objective function is to minimize network bandwidth consumption, we can ignore the replication traffic (i.e., the traffic associated with managing the replicas, distributing content to the replicas, etc.) since it is typically orders of magnitude smaller than the traffic generated by users' requests. Furthermore, since most content distribution networks (e.g., Akamai) that replicate content have their own private high-speed networks, the bandwidth consumption incurred during the replication is usually not a major concern. On the other hand, if our objective function is to optimize another performance metric, such as users' perceived latency, then it is unclear how to incorporate the replication cost (in the unit of network bandwidth) into the objective function in a different unit (such as time in the case of client latency).

As in [16], we fix the origin server to be one of the replica sites. However, including or excluding the original server is not a fundamental choice and has little impact on our results.

IV. Placement Algorithms

In this section, we present a number of algorithms for solving the minimum K-median problem. The objective is to minimize the total cost of all the requests. We define the cost of a request from node i to node j as the distance between the two nodes,

[1] If the capacity of replicas needs to be taken into account, we can use the capacitated minimum K-median problem formulation, and the algorithms described below will still apply.

where the distance can reflect any performance metric we want to optimize, such as latency, hop counts, or the economic cost of the path between two nodes (assuming there is a cost associated with the links on the path). The algorithms work the same regardless of what metric is used.

A. Tree-based Algorithm

Li et al. proposed a placement algorithm in [16] based on the assumption that the underlying topologies are trees, and modeled it as a dynamic programming problem. The algorithm was originally designed for Web proxy cache placement, and it is also applicable for Web replica placement. At a very high level, they divide a tree T into several small trees T_i, and show that the best way of placing $t > 1$ proxies in the tree T is to place t'_i proxies the best way in each small tree T_i, where $\sum_i t'_i = t$. The algorithm is shown to find an optimal placement when the underlying topologies are trees, and clients request from the proxy on the path toward the Web server, that is, clients cannot request from a sibling proxy. However, these two assumptions can prune possibly better placement choices. As shown in Section VI, the optimal solutions under these assumptions are usually not as good as the solutions found by the greedy and hot spot heuristics (without the assumptions), which are described later in this section.

B. Greedy Algorithm

The basic idea of the greedy algorithm is as follows. Suppose we need to choose M replicas among N potential sites. We choose one replica at a time. In the first iteration, we evaluate each of the N potential sites individually to determine its suitability for hosting a replica. We compute the cost associated with each site under the assumption that accesses from all clients converge at that site, and pick the site that yields the lowest cost. In the second iteration, we search for a second replica site which, in conjunction with the site already picked, yields the lowest cost. In general, in computing the cost, we assume that clients direct their accesses to the nearest replica (i.e., one that can be reached with the lowest cost). We iterate until we have chosen M replicas.

C. Random

The random algorithm is oblivious to client workload, and randomly chooses M replicas among N potential sites from a uniform distribution. To improve performance, we execute the algorithm several times – in our simulations, we execute over 10 times, and pick the random assignment that yields the lowest cost.

D. Hot Spot

The hot spot algorithm attempts to place replicas near the clients generating the greatest load. It sorts the N potential sites according to the amount of traffic generated within their vicinity. It places the replicas at the top M sites that generate the largest amount of traffic. We define A's vicinity as the circle centered at A with some radius. In our simulations, we vary the radius from 0 to the maximum distance between any pair of nodes in the graph, and report the best performance over all the radii tested.

E. Super-Optimal Algorithm

As mentioned earlier, the minimum K-median problem is NP-hard. Computing the exact optimal solution is therefore too computationally intensive to be useful in practice. To evaluate how well the algorithms described above perform, we compute a (fairly tight) lower bound on the cost of any feasible solution of our minimization problem. We do so using a *super-optimal* algorithm based on Lagrangian relaxation with subgradient optimization [21]. As its name suggests, the solution produced by this algorithm may be better than optimal because it may not be feasible. Nevertheless, it serves as a useful data point for comparison.

More specifically, the K-median problem can be stated as the following integer program, where the 0-1 variable y_i, $i \in N$, indicates whether the location i is selected as a center, and the 0-1 variable $x_{ij}, i, j \in N$, indicates whether location j is assigned to the center at i:

$$\begin{array}{lll}
\text{minimize } \sum_{i,j \in N} d_j c_{ij} x_{ij} \ (1) & & \\
\text{subject to} & \sum_{i \in N} x_{ij} = 1 \text{ for each } j \in N, & (2) \\
& x_{ij} \leq y_i \text{ for each } i, j \in N, & (3) \\
& \sum_{i \in N} y_i \leq k, & (4) \\
& x_{ij} \in \{0, 1\}, \text{ for each } i, j \in N, & (5) \\
& y_i \in \{0, 1\}, \text{ for each } i \in N. & (6)
\end{array}$$

The set of constraints (2) ensures that each location $j \in N$ is assigned to some center $i \in N$, the set of constraints (3) ensures that, whenever a location j is assigned to a center i, then a center must have been opened at i, and (4) ensures that at most k centers are open.

We then apply Lagrangian relaxation to the integer programming problem, where the constraints (2) are weighted by multipliers and placed in the objective function as follows:

$$L(x, y, u) = \sum_{i,j \in N} d_j c_{ij} x_{ij} - \sum_{j \in N} u_i (\sum_{i \in N} x_{ij} - 1)$$

Then we use the subgradient method given in Held *et al.* [25] to compute the super-optimal bound. It is an iterative procedure that begins with a specified multiplier value u^0 and generates a sequence of multiplier values u^k. On iteration k the Lagrangian problem is solved with multipliers u^{k-1}, and a new value u^k is determined from u^{k-1} and the Lagrangian solution using the rule in [25]. At each iteration an upper bound on the optimal value is available. In order to obtain the lower bound in our minimization problem, we use the subgradient method on $-L$. In our simulations, we use 1000 iterations for 100 nodes topologies, and 200 iterations for all other topologies. To get a tighter bound, for a specific instance of problem we use three random values to initialize the multiplier u^0, and obtain a super-optimal bound from each value of u^0. The maximum of the three is used as the lower bound.

F. Summary

Table I lists the computational time of various algorithms for selecting M replicas among N potential sites. If only a handful of potential hosting sites is available, the cost of the computationally complex algorithms may not be significant. However, in our analysis, we consider clusters defined by address prefixes, which will be explained in Section V-B, as potential

Tree-based [16]	Greedy	Random	Hot Spot
$O(N^3 M^2)$	$O(N^2 M)$	$O(NM)$	$N^2 + min(NlogN + NM)$

TABLE I

COMPARISON OF COMPUTATIONAL TIME OF VARIOUS ALGORITHMS

replica sites. In this case, N is on the order of 100,000 (the number of address prefixes in the Internet), so clearly the computational complexity of the replica placement algorithm becomes very significant. To reduce the computational cost, we consider only the top, in terms of requests generated, few hundreds or few thousands of clusters. Since these top clusters generate most of the traffic, as shown in Section V-B, ignoring the requests from unpopular clusters has little effect on the results.

In the following sections, we will compare the above algorithms with the super-optimal algorithm using the real Web traces and network topologies.

V. SIMULATION METHODOLOGY

To evaluate the performance of the various algorithms presented in this paper, we simulate the behavior of the algorithms on a variety of network topologies and Web workloads. In this section, we discuss the network topologies and Web workloads that we use in our evaluations. We then describe the performance metric that we use as a basis for comparing the algorithms.

A. Network Topology

In our simulations, we use both randomly generated network topologies and the real Internet topologies derived from BGP routing tables.

We generate two types of random network topologies: random trees and random graphs. The primary reason for studying performance on the tree structure is to determine how the optimal tree-based algorithm compares to the other algorithms. To generate random trees, we wrote a simple program that takes 3 parameters: the total number of nodes, the maximum distance between any two nodes, and the maximum degree of a tree node. Starting from the root node, we recursively create random children until the total number of nodes specified is reached. In our simulations, we use 100-node and 300-node trees, and we set the maximum distance to 10 and the maximum node degree to 10, 15, and 20. For each parameter setting, we generate three different trees.

To generate random graphs, we use the GT-ITM internetwork topology generator [5]. In particular, we use three network models: pure random, Waxman, and Transit-Stub. In the pure random model, vertices are distributed at random locations in a plane, and an edge is added between a pair of vertices with probability p. In the Waxman model, the probability of an edge from u to v is given by $P(u, v) = \alpha e^{-d/(\beta L)}$, where $0 < \alpha$ and $\beta \leq 1$ are parameters of the model, d is the Euclidean distance from u to v, and L is the maximum distance between any two nodes. The Transit-Stub model generates hierarchical graphs by composing interconnected transit and stub domains; see [26] for further details.

We use a wide range of parameters for each network model. For each parameter setting, we generate three different topologies. We do not claim that these network models and parameters we use are representative for the Internet topology. Instead, our goal is to make the generated topologies as rich as possible by using multiple models with a wide range of parameters. As we will show in Section VI, the performance of the placement algorithms is similar across different network models and parameters.

Trace ID	Web Site	Period	Duration
1	MSNBC	8/3/99 - 8/5/99	9 am - noon
2	MSNBC	9/27/99 - 10/1/99	All day
3	MSNBC	10/7/99 - 10/14/99	All day
4	ClarkNet	9/4/95 - 9/10/95	All day
5	NASA	7/1/95 - 7/31/95	All day

TABLE II

ACCESS LOGS USED

We also construct a simplified model of the actual Internet topology using BGP routing data from a set of seven geographically-dispersed BGP peers. Each BGP routing table entry specifies an AS path, $AS_1, AS_2, \cdots, AS_n$, to a destination address prefix block (AS_1 corresponds to the BGP peer and AS_n corresponds to the destination address prefix block). We construct an AS-level topology graph of the network using the AS paths. The AS path $AS_1, AS_2, \cdots, AS_n$ yields edges between adjacent nodes (AS's) in the path (e.g., $(AS_1, AS_2), (AS_2, AS_1), (AS_2, AS_3), etc.$). We map individual clients and address prefix blocks to their corresponding AS nodes in the topology graph, and assign the distance between two nodes as the AS hop counts between the two nodes.

While not very detailed, an AS-level topology at least partially reflects the true topology of the Internet. Furthermore, recent study [17] has shown the AS hop count of a path is a decent indicator of the path's proximity, reliability, and stability.

B. Web Workload

To evaluate the algorithms on realistic traffic patterns, we use the access logs collected at the MSNBC server site [19], during three periods, as shown in Table II. MSNBC is a large and popular commercial news site in the same category as CNN [9] and ABCNews [1], and is consistently ranked among the busiest sites in the Web [18]. For diversity, we also use the traces collected at ClarkNet [10] and NASA Kennedy Space Center in Florida [20] during 1995. Table II shows the detailed trace information. We use the workload in one day or 3 hours (for the August 1999 traces) to parameterize one simulation setup.

We use the access logs in the following way. First, we use the approach proposed by Krishnamurthy *et al.* in [15] to cluster the Web clients that are topologically close together. Their method is based on the information available from BGP routing table snapshots, and they show that it significantly outperforms a heuristic that assumes a fixed-length, 24-bit network prefix.

To use their method for clustering clients, we obtained the complete BGP routing tables from seven geographically and topologically diverse ISPs [13]. For each client IP address in the access logs, we find its best matching prefix in the union of the routing tables. All the clients whose IP addresses have the same best prefix match belong to the same cluster. Figure 1 plots the number of requests generated by each cluster. As we can see, in the 8/3/99 MSNBC trace, the top 10, 100, 1000, and 3000 clusters account for about 23.55%, 44.86%, 77.96%, and 93.97% requests, respectively. The other server traces have similar results, though the NASA traces are a little more concentrated: the top 10, 100, 1000, and 3000 clusters account for about 29.67%, 51.96%, 85.39%, and 97.37% requests, respectively.

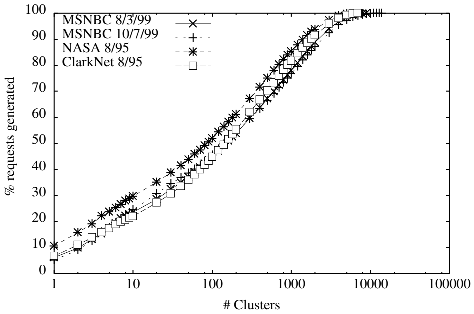


Fig. 1. The CDF of the number of requests generated by the Web clusters defined by address prefixes.

For a network topology of a specific size, say 100 nodes, we choose the top 100 clusters in the traces and map them randomly to the nodes in the graph. Assigning a cluster 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 cluster C_i. For each network topology and access log, we make three different random assignments from the clusters to the nodes in the graphs.

C. Performance Metric

To compare the performance of the algorithms on the various network topologies and access logs, we use the ***relative performance*** of the algorithms as a metric. We define the relative performance as the ratio between the cost of the feasible solution found by the algorithm to the cost determined by the super-optimal algorithm. The relative performance is an appropriate metric, since it reflects the cost we want to minimize. The normalization step is to show how far away we are from the super-optimal, and does not alter the metric to be minimized. The smaller the value of the relative performance, the better the algorithm performs. A relative performance of 1 implies the algorithm finds an optimal solution, but the optimal solution need not necessarily have a relative performance of 1, since the super-optimal solution may not be achievable.

VI. Simulation Results

In this section, we evaluate the performance of the various placement algorithms on a variety of synthetic and realistic network topologies using the Web server traces.

A. Random tree topologies

First, we evaluate the performance of the placement algorithms on the tree topologies. More specifically, we run each placement algorithm in hundreds of simulation runs and examine the performance of the algorithm across all simulation runs. Each simulation run is parameterized by (i) the Web server trace, (ii) the network topology, (iii) the mapping of clusters to nodes in the simulation topology, and (iv) the number of replicas to pick. We evaluate the algorithms on 100-node and 300-node trees using Web traces 1 and 3 listed in Table II. We use three different random assignments from the clusters defined by address prefix to the nodes in the trees. We then vary the number of replicas to place from 1 to 80 for the 100-node trees, and from 1 to 100 for the 300-node trees.

Figure 2 shows the cumulative distribution (CDF) of the relative performance of the algorithms on tree topologies. As we can see, the greedy algorithm and the tree-based algorithm perform the best, with the greedy algorithm slightly better. The hot spot algorithm has a performance in between these two and the random algorithm, which clearly has the worst performance. We quantify the differences in relative performance of the algorithms in the next set of graphs.

Figure 3 shows the minimum (best case), maximum (worst case), and median values of the relative performance of these algorithms over all simulation runs, where the tree-based, greedy, random, and hot spot algorithms are numbered algorithm 1, 2, 3, and 4, respectively. On average, both the greedy and tree-based algorithms are within 5% worse than the super-optimal algorithm for 100-node trees, and within 20%–30% worse than the super-optimal algorithm for 300-node trees. The hot spot algorithm has a relative performance that is about 30% worse than the super-optimal algorithm. The random algorithm performs considerably worse than the others. This is also evident from its CDF curve shown in Figure 2, which has a very gradual slope. For the three graph sizes, 50% of the simulation runs for the random algorithm have a relative performance of at least 2.5. Note also that the relative ranking of these algorithms is consistent across all the tree topologies and Web traces tested.

The reason that even in tree topologies the tree-based algorithm is not the best performer is that it assumes clients can only direct requests to replicas on the path toward the Web server. This assumption eliminates some potentially better placement choices.

B. Random graph topologies

We also evaluate the performance of the placement algorithms on random graphs generated by the GT-ITM topology generator. As with the tree topologies, we run each algorithm in hundreds of simulation runs and examine the performance of the algorithms across all simulation runs. We vary the number of replicas to place from 1 to 80 for the 100-node graphs, from 1 to 100 for the 300-node graphs, and from 1 to 200 for the 1000 and 3000-node graphs. For every graph size, we use three network models with different parameters, as described in Section V-A. We plot the CDF of the relative performance of the different placement algorithms across all simulation runs in Figure 4. We show the minimum, maximum, and median of the relative performance across all simulation runs using errorbars in Figure 5, where the algorithms are numbered as in Figure 3.

Before describing the results in the graphs, we make the following observations. First, the tree-based algorithm requires the underlying topology to be a tree. For our evaluation of the tree-based algorithm on general graphs, we generate three random spanning trees for a given graph, where all the spanning trees are rooted at the original server node. Then we run the algorithm on each of the trees. The three adjacent errorbars for $Algorithm\ ID = 1$ in Figure 5 correspond to the performance obtained using the three different spanning trees. Second, we only report the performance of the tree-based algorithm for 100-node and 300-node topologies since it takes too long to run on topologies with 1000 or more nodes. For example, it takes over 11 hours to place 5 replicas among 1000 potential sites on an

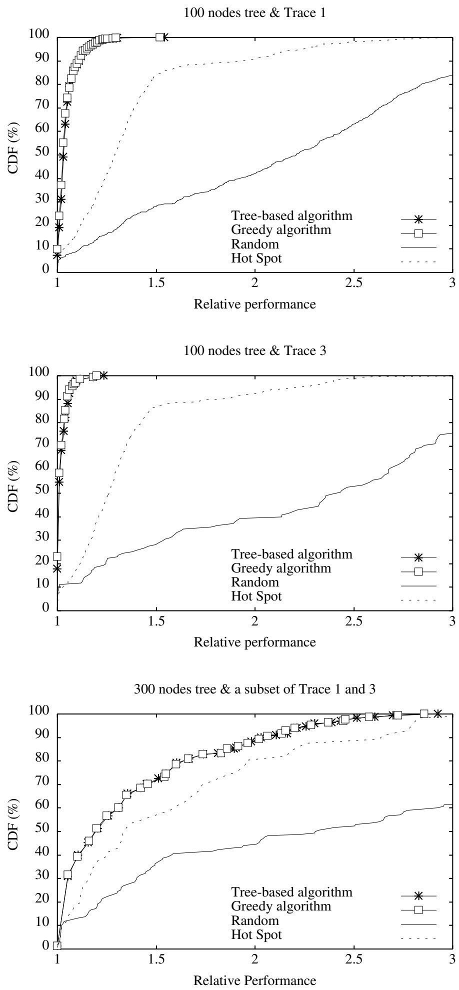


Fig. 2. The CDF of relative performance across all simulation runs of the placement algorithms on tree topologies.

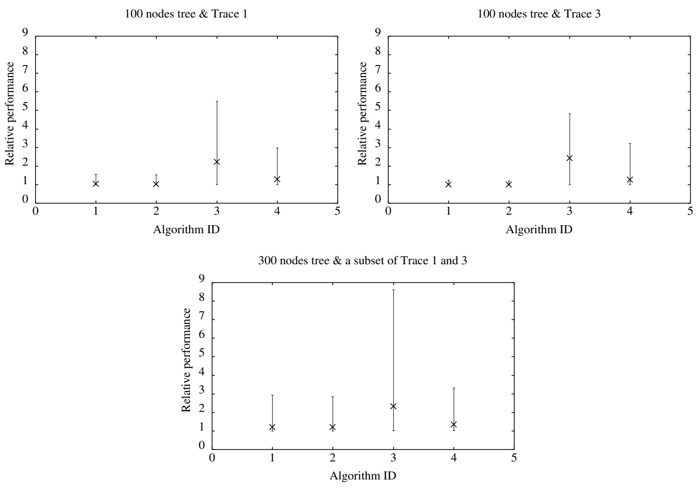


Fig. 3. A summary of the performance of the placement algorithms on tree topologies using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

UltraSparc machine with a 500 MHz CPU and 4 GB of memory. As a result, we conclude that the tree-based algorithm is not practical for making real-time placement decisions when the network size grows to thousands of nodes. In comparison, for the same scenario, the greedy, hot spot, and random algorithms take less than 1 minute to run.

Compared to the super-optimal algorithm, the greedy algorithm performs within a factor of 1.5 in the median cases, and around a factor of 4 in the maximum cases. These results are significantly better than all of the other algorithms, including the tree-based algorithm. Another interesting observation is that the hot spot algorithm is often better than the tree-based algorithm on the general graphs. The random algorithm, as before, performs the worst: its median performance is around 2.5 and its maximum relative performance is as high as 11–13.

C. Internet topology

We also evaluate the performance of the placement algorithms using a model of Internet topology derived from BGP routing tables. In this case, we use AS hop counts as the distance metric between two connected nodes. As shown in Figure 6 and Figure 7, the ranking of the various algorithms stays the same as in the randomly generated graphs. From the best to the worst in order are the greedy, hot spot, tree-based, and random algorithms. However, the performance difference between the algorithms is smaller than that in the randomly generated graphs. This is because the number of AS hops between any two nodes is not as widely distributed as the distance in the generated topologies. The number of AS hops varies from 0 to 6 for the 100 top AS's (in terms of the number of requests generated to the MSNBC Web server during the periods under study), and from 0 to 9 for the 1000 top AS's. In contrast, the distance between any two nodes in the generated topologies can be different by orders of magnitude.

D. Effects of imperfect knowledge about input data

The above simulation results are based on the assumption that we have perfect knowledge of the underlying topologies and the number of requests generated from each node. In practice, we do not have perfect knowledge about these inputs, but only have rough estimates. In this section, we examine how imperfect knowledge about the input data affects the placement decision. In particular, we want to find out if the placement decision based on inaccurate information will still be useful, and how far its performance deviates from that obtained using perfect knowledge.

Our approach is to salt the input data with random noise of uniform distribution, and vary the amount of noise added to the input data. This is done in two ways: (1) we perturb the volume of requests from a client by up to a factor of 2 (i.e., if the true number of requests is d, the perturbed value ranges between $\frac{d}{2}$ and $2d$), and (2) we perturb the distance, c_{ij}, between two nodes i and j by up to a factor of 4 (i.e., the corrupted distance ranges between $\frac{c_{ij}}{4}$ and $4c_{ij}$). We feed the salted inputs to the placement algorithms, and compute the cost after applying the placement decision to the actual input data. As before, we use relative performance as the metric, defined as the ratio between the cost of the feasible solution found by the algorithms using the salted

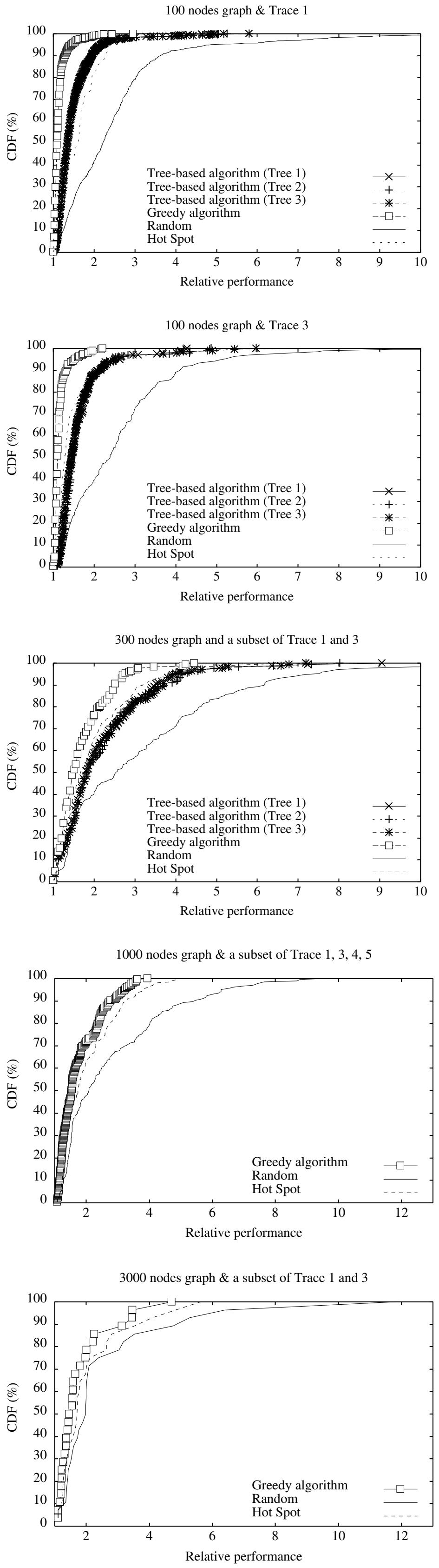


Fig. 4. The CDF of the relative performance across all simulation runs of the placement algorithms on general graphs.

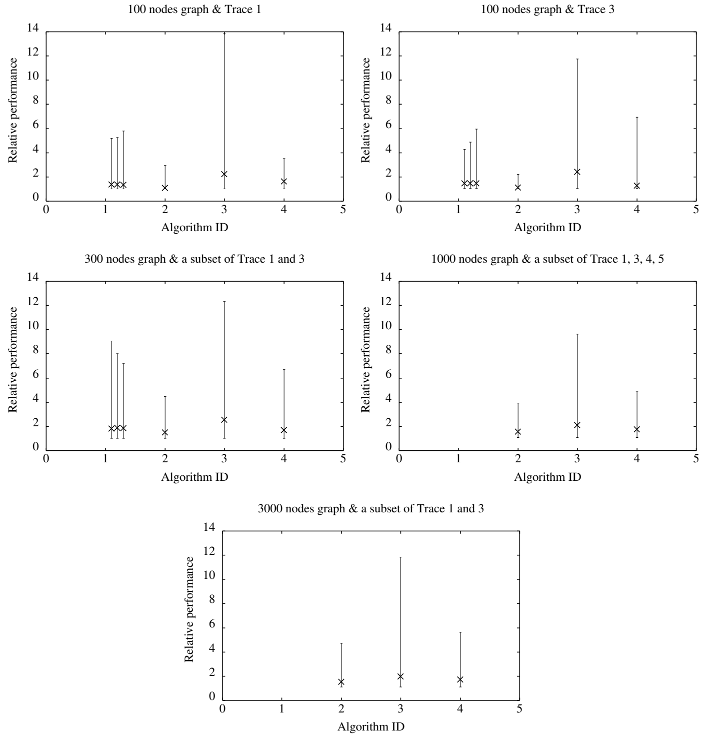


Fig. 5. A summary of the performance of the placement algorithms on graph topologies using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

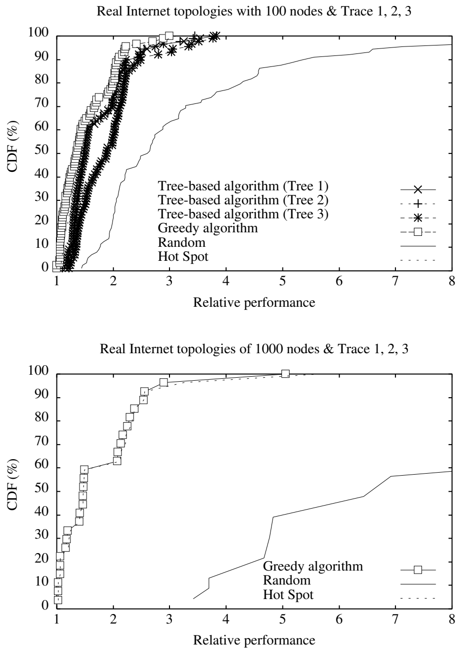


Fig. 6. The CDF of relative performance across all simulation runs of the placement algorithms on the model of the real Internet topology derived from the BGP routing tables.

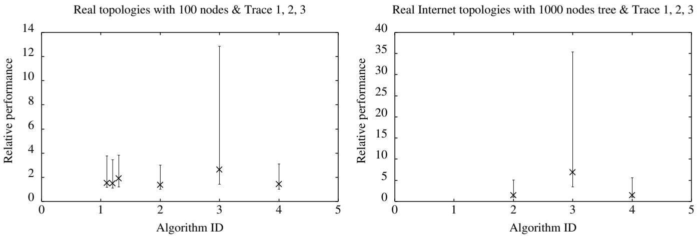


Fig. 7. A summary of the relative performance of the placement algorithms on the model of the real Internet topology using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the corresponding algorithm. The tree-based, greedy, random, and hot spot algorithms are respectively numbered 1, 2, 3, and 4 in the graph.

inputs to the cost determined by the super-optimal algorithm using the actual inputs.

Figure 8 shows the minimum, maximum, and median of the relative performance over all the values of the error rates in the distance and load. As we can see, the performance deviation is small. In particular, even with the salted error as high as a factor of 4, the cost of the greedy algorithm is in most cases within a factor of 2 of the super-optimal algorithm when using perfect knowledge. This is also evident from Figure 9, which plots the relative performance of the greedy algorithm versus the errors in the input. As we can see, as the error increases, the performance degrades only slightly.

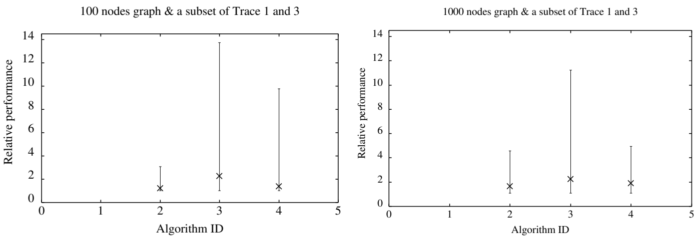


Fig. 8. The relative performance of the placement algorithms on the graph topologies using errorbars, with both the load and distance information salted with random noise.

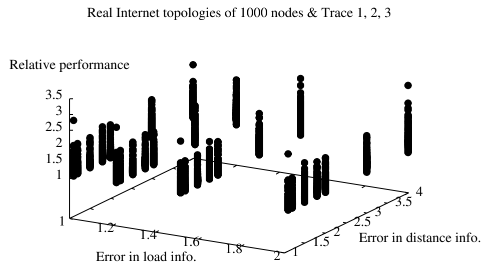


Fig. 9. The relative performance of the greedy algorithm on the input salted with random noise.

The above perturbation technique on the topology is most useful when our performance metric is the propagation delay or the economic cost of the paths. If our performance metric is AS hop count, we can infer the distance between two nodes by using BGP routing tables as illustrated in Section V-A. However, when the number of BGP peers providing routing information is very limited, we may not have a very accurate AS-level topology map (for example, we do not see all the links).

To study the effect of overlooking some network links on the placement algorithms, we randomly remove 0–50% of the edges in the AS-level Internet topology derived from the BGP routing tables, and use the perturbed topology information in the placement algorithm. Figure 10 shows the performance results for the greedy algorithm normalized by the performance of the super-optimal algorithm using perfect topology information for the 10/8/99 MSNBC server trace. As we can see, the performance of the greedy algorithm hardly changes as more edges are removed. In particular, even when the edge removal probability is as high as 50%, the relative performance of the greedy algorithm stays within 2.6. The insensitivity of the greedy algorithm to the edge removal partly comes from the fact that the only topology information that the greedy algorithm (and all other algorithms except the tree-based algorithm) requires is the distance matrix. When testing the distance matrix in more detail, we find that the distance matrix is not sensitive to edge removal. In particular, removing up to 5% of the edges in the graph does not change the distance matrix in our experiments.

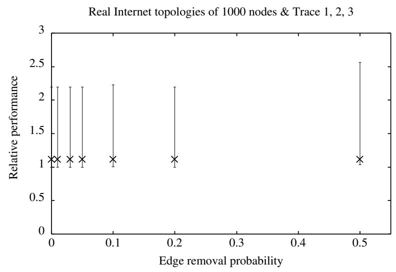


Fig. 10. A summary of the relative performance of the greedy algorithm during edge removal using errorbars. The lower and upper bounds, and the X mark on each errorbar correspond to the minimum, maximum, and median, respectively, of the relative performance of the greedy algorithm.

E. Stability of input data

The above section studies the effect of imperfect knowledge on the placement decision. One of the major reasons that we do not have perfect knowledge about the input data is that the input data is changing over time. When making the placement decision for the next 24 hours, ideally we would like to give the placement algorithms the load and network information for the next 24 hours. However, in practice, we can only use the past information to predict the future load and network information. How good such a prediction is can significantly affect the performance of the placement algorithm. In this section, we investigate this issue in detail.

Our evaluation is done in two parts. In the first part, we assume the topology information is accurate but the load information is based on the prediction. In particular, we consider the scenario where we want to make a placement decision for 10/1/99 by using the workload for the previous few days. We predict the load generated from a cluster by averaging its load during the previous n days, where n varies from 1 to 4.

To perform this evaluation, we use Trace 2 listed in Table II, which contains the access logs from 5 consecutive working days (from Monday to Friday). We pick the top 1000 clusters from 10/1/99. (The top 1000 clusters on 9/27/99 - 9/30/99 have more than 90% overlap with those on 10/1/99.) As before, we randomly assign the clusters to the nodes in the randomly generated topologies of various network models and parameters. For each topology and cluster assignment, we simulate the placement algorithms on the actual workload on 10/1/99 and five predicted workloads: (i) the workload of 9/30/99, (ii) the averages of 9/29/99 and 9/30/99, (iii) the averages of 9/28/99 – 9/30/99, and (iv) the averages of 9/27/99 – 9/30/99.

Figure 11 shows the CDF of the greedy algorithm's performance across all simulation runs. Here we normalize the performance of the greedy algorithm using the predicted load by its performance using the actual workload on 10/1/99. The lower the normalized performance, the better the prediction is. A normalized performance of 1 means the performance is exactly the same as that obtained using the actual workload. As we can see, the performance using the predicted workload closely matches the performance using the actual workload, within 5% over all cases. Note that, in some cases, the performance using the predictions is slightly better than using the actual workload. This is because the greedy algorithm does not give the optimal performance even when the input data is completely accurate.

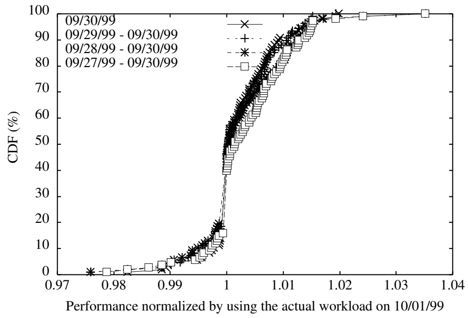


Fig. 11. The CDF of the greedy algorithm's performance using the predicted workload normalized by its performance using the actual workload across all simulation runs.

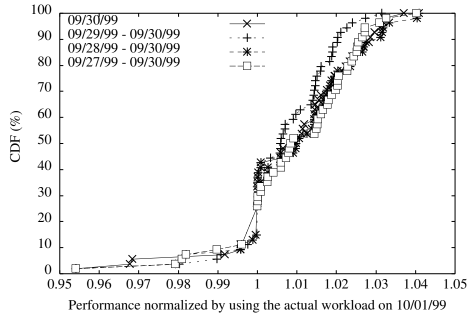
For the second part of the evaluation, we use the same strategy as above, but salt the topology information with random noise as described in Section VI-D. Figure 12 shows the performance results when we perturb the distance between any two nodes by up to a factor of 1.2 and 2. As we can see, the performance deviation from using the accurate load and network information is small: when the perturbation in distance is up to a factor of 1.2 and 2, the deviation is only within 5% and 17%, respectively. Moreover, the performance results are similar across all the prediction windows tested.

Finally, we have observed significant variation between weekday workloads and weekend workloads, even though they are consecutive in time. This is not surprising, and suggests that we should use past weekday workload information to predict future weekday workloads, and likewise use past weekend workload data to predict future weekend workloads.

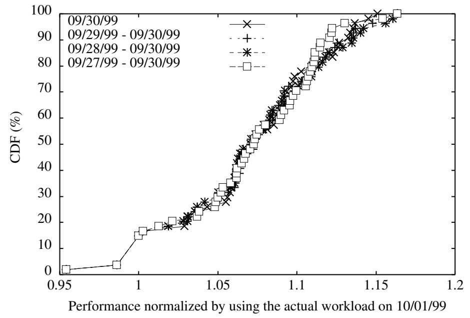
VII. Practical Considerations

In this section, we discuss ways to obtain the input data for the placement algorithms. As mentioned earlier, the input to the placement algorithms is a graph with weighted nodes and edges. A node's weight represents the amount of traffic initiated by the node, and an edge's weight represents latency, or link cost, or hop count, etc. In order to apply the placement algorithms in practice, we need to be able to obtain both traffic pattern and network topology information in real-time.

Obtaining node weights is relatively straightforward. During re-provisioning, the Web server communicates with all the active replicas (i.e., the replicas that serve the requests, as opposed



(a) Perturbation factor is 1.2



(b) Perturbation factor is 2

Fig. 12. The CDF of the greedy algorithm's performance using the predicted workload normalized by its performance using the actual workload when we also perturb the distance between any two nodes.

to potential replica sites) about the number of requests generated by all the popular clusters, where clusters are identified using the approach proposed in [15].

The method for obtaining edge weights depends on the performance metric that we want to optimize. Since replication placement is a relatively long-term provision, we believe it is desirable to use the performance metrics that are stable on the order of hours, such as propagation delay, hop count, or economic cost of the path between two nodes.

To approximate the distance between each pair of nodes, we can use BGP routing tables to infer the hop counts between each pair of nodes as described in Section V. An interesting question is how many BGP peers we would need routing information from in order to construct a fairly accurate AS-level topology map. The answer clearly depends on the richness of the connectivity, i.e., the (average) degree of nodes in the topology graph. The greater the degree, the greater the number of BGP peers from which we will need routing information. The worst case is a completely connected graph (which, however, is far from the reality). However, as we show in Section VI, the performance of the greedy algorithm is not sensitive to overlooking some network links – its relative performance stays within 2.6 of the super-optimal algorithm even when the edge removal probability is as high as 50%.

A separate question is whether knowing the topology is sufficient for solving the placement problem. In general, we would need some notion of Internet "weather", that is, the network performance between two points, say a client location and a potential replica site. There are several research efforts (e.g., IDMaps [14]) focusing on the problem of constructing such an Internet weather map. If desired, we could, in a straightforward manner, substitute cost metrics derived from an Internet weather map in place of those derived from topology information in our algorithms. Before such a service is widely available, we can also have the Web sites periodically *ping* or *traceroute* a representative client in each identified popular cluster. Since the number of popular clusters is not large, usually around 1 - 3 thousand as is the case with MSNBC Web site, such probing is affordable especially when the provisioning timescale is on the order of hours or longer.

VIII. Conclusion

In this paper, we study the online problem of placing Web server replicas in content distribution networks (CDNs) to minimize the cost for clients to access data replicated on the servers. We approach the placement problem by formulating it as a minimum K-median graph theoretic problem. We present various algorithms for solving the minimum K-median problem, and evaluate the performance of the algorithms by simulating their behavior on synthetic and real network topologies and several Web traces. We also address a number of practical issues when using these algorithms online in a content distribution network. As far as we know, this is the first experimental study on this subject.

Our main results and conclusions are:

- Placement algorithms should incorporate client workload information, such as client distance and request rate, in their placement decisions. Such algorithms consistently perform a factor of 2 – 5 better than a workload-oblivious random algorithm.
- A greedy algorithm that places replicas based upon both a distance metric and request load performs the best (i.e., its median performance is within a factor of 1.1 – 1.5 of optimal). A hot spot algorithm based upon request load only performs nearly as well (its median performance is within a factor of 1.6 - 2 of optimal). A tree-based algorithm developed for proxy cache hierarchies [16] performs better than random placement, but not as well as the algorithms for general graph topologies.
- The placement algorithms are not very sensitive to noise in the estimates of distance and load used as inputs to the algorithms. Even with rough estimates of client distance and request load salted with random noise, the algorithms perform nearly as well as when they used perfect knowledge. For example, when the salted error is as high as a factor of 4, the greedy algorithm stays within a factor of 2 of the super-optimal in most cases.
- When deployed, the placement algorithms must predict future request load based upon past information. We show that the algorithms can use a simple moving window average for predicting load with negligible impact on performance.
- The relative performance of the placement algorithms is consistent across network topologies (tree, random graph, AS hopcount), topology parameters (the number of nodes, inter-node distance), trace workloads, and noise in the inputs.

Based upon our results, we conclude that a greedy algorithm for Web server replica placement can provide content distribution networks with performance that is close to optimal. Although the greedy algorithm depends upon estimates of client distance and load predictions, we find that it is relatively insensitive to errors in these estimates and is therefore a viable and practical algorithm for use in the general Internet environment where workload information will likely be imperfect.

As for future work, we are interested in exploring incremental versions of the placement algorithms that also take into account the cost of changing the set of replica sites. Ideally, for placement strategies with similar performance, we prefer the one that incurs the least amount of perturbation to the system. For example, if site A is already hosting a Web service, then we prefer not to replace it with another replica site unless the performance degradation of continuing to use A is significant. We are also interested in studying distributed versions of the placement algorithms to further improve the scalability of the system. One possible distributed algorithm would be to make the algorithm hierarchical.

IX. Acknowledgements

Thanks to Craig Labovitz for providing us with the complete BGP routing tables from seven different ISPs. Thanks to Yin Zhang and Ravi Kumar for helpful discussions.

References

[1] ABC News. http://www.abcnews.com.
[2] Akamai. http://www.akamai.com.
[3] M. F. Arlitt, and C. L. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. IEEE Transactions on Networking, Vol. 5, No. 5, October 1997.
[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications.
[5] K. Calvert, and E. Zegura. GT Internetwork Topology Models (GT-ITM). http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm.
[6] M. Charikar, and S. Guha. Improved Combinatorial Algorithms for the Facility Location and K-Median Problems. In *Proc. of the 40th Annual IEEE Conference on Foundations of Computer Science*, 1999.
[7] M. Charikar, S. Guha, E. Tardos, and D.B. Shmoys. "A constant-factor approximation algorithm for the k-median problem". Proceedings of the 31st Annual ACM Symposium on Theory of Computing (1999).
[8] F.A. Chudak and D.B. Shmoys. "Improved approximation algorithms for the capacitated facility location problem". In *Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (1999)*, S875-S876, 1999.
[9] Cable News Network. http://www.cnn.com.
[10] ClarkNet Server Traces. http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html.
[11] Digital Island. http://www.digitalisland.com/.
[12] Exodus. http://www.exodus.com.
[13] IPMA Project. http://www.merit.edu/ipma/.
[14] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang. On the Placement of Internet Instrumentation. In *Proc. of INFOCOM'2000*, March 2000.
[15] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *Proc. of SIGCOMM'2000*, August - September 2000.
[16] B. Li, M. J. Golin, G. F. Ialiano, and X. Deng. On the Optimal Placement of Web Proxies in the Internet. In *Proc. of INFOCOM'99*, March 1999.
[17] P. R. McManus. A Passive System for Server Selection within Mirrored Resource Environments Using AS Path Length Heuristics. http://proximate.appliedtheory.com/proximate_study.html.
[18] MediaMetrix. http://www.mediametrix.com.
[19] MSNBC. http://www.msnbc.com.
[20] NASA Kennedy Space Center Server Traces. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.
[21] G. Cornuelos, M. L. Fisher, and G. L. Nemhauser. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. Management Sciences, Vol. 23, 1977, pp. 789-810.
[22] A compendium of NP optimization problems. http://www.nada.kth.se/ viggo/problemlist/compendium.html.
[23] V. N. Padmanabhan, and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *Proc. of SIGCOMM'2000*, August - September 2000.
[24] J. Touch. Web Caching and Replication - Research Issues. Internet daft, draft-ietf-wrec-res-00.text. http://www.wrec.org/Drafts/draft-ietf-wrec-res-00.txt.
[25] P. Wolfe, and H. P. Crowder. Validation of Subgradient Optimization. Math Programming. Vol. 6, 1974, pp. 62 - 88.
[26] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an InternetWork. In *INFOCOM'96*, 1996.