Replica Placement Algorithms with Latency Constraints in Content Distribution Networks

Jie Su
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207
jsu@unity.ncsu.edu

Douglas Reeves
Departments of Computer Science and Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-8207
reeves@eos.ncsu.edu

ABSTRACT

Content distribution networks (CDNs) are a recent development to improve the performance of networked applications. The design of replica placement algorithms is one of the foremost problems in CDNs. Current replica placement methods do not guarantee a bound on the maximum latency to any client. In this paper, we propose several algorithms for replica placement with latency constraints. The algorithms are compared with respect to scalability and performance under a variety of conditions, including the case where clients' request patterns are unknown.

Categories and Subject Descriptors

G.2 [**Mathematics of Computing**]: Discrete Mathematics; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems*

General Terms

Algorithms

Keywords

Content Distribution Network, Replica placement, Algorithm

1. INTRODUCTION

The growth in the World Wide Web has moved us toward distributed, highly interconnected information systems. Unfortunately, this growth has also resulted in serious performance problems, and degraded user experiences, for some applications. Techniques that ensure satisfactory performance of networked applications are important in such an environment. Content distribution network *(CDN)* services, such as Akamai and Digital Island, are a useful technique to improve the performance of networked applications. They work by locating many sites with comparable functionality at different points in the network. This improves the available bandwidth and fault tolerance, and reduces the average latency for obtaining data from servers.

The design of *replica placement* algorithms is one of the foremost problems in CDNs. These algorithms decide what data to replicate at what server locations, in order to achieve the best performance at the lowest cost.

In this paper, we propose several server placement algorithms that distribute content to clients efficiently, with *latency constraints*. More specifically, we consider the following scenario. A popular Web site aims to improve end-user experience by pushing its content to some hosting servers so that each individual user will not experience any latency larger than a latency constraint. The problem is to choose a smallest set of replicas among potential sites such that the latency constraint is met for each of the clients. We evaluate the performance of the various placement algorithms by simulating their behavior on generated network topologies.

The rest of the paper is organized as follows. In Section 2, we review related research work. In Section 3, we give a formulation of replica placement with latency constraint problem, and prove the problem is NP-complete. In Section 4, we present three algorithms for placing replicas with latency constraints. Section 5 describes and analyzes the results of simulating these algorithms under realistic conditions. We conclude the paper in Section 6.

Copyright is held by the author/owner(s).
WWW2004, May 17–22, 2004, New York, NY USA.
ACM xxx.xxx.

2. PREVIOUS WORK

Several papers concerning replica placement for CDNs have been published recently. Research on replica placement has focused on the *minimum K-median problem*, which is useful for determining the placement of replicas given the network topology and the users' demands. In the solution for the minimum K-median problem, the objective of the replica placement algorithm is to minimize the total latency (for all clients) of reaching a target server, subject to the system resources and traffic patterns. The minimum K-median problem is NP-complete for general-graph network topologies. However, there are constant-factor approximation algorithms. These algorithms also work for other objective functions besides total latency. For example, the same methods can minimize the total number of intermediate hops (hop count) between clients and servers, or the total cost, if the use of each network link is associated with a cost.

Our application of the minimum K-median problem is defined more formally as follows. Given are N nodes in the network. Each node is eligible to be a server hosting a replica of the content. Every node also acts as an aggregator of requests and can therefore be regarded as a client, with an aggregated traffic demand of r_j for node j. K of the N nodes are to be selected as servers. Let the distance between two nodes i and j be represented as d_{ij}. A client j which is assigned to its closest server i (ties broken arbitrarily) incurs a cost $r_j d_{ij}$. The objective is to choose the K servers, so that the total cost is minimized. A recent work gave a 4-approximation algorithm for this problem [3].

The minimum K-median problem given above may be constrained by the service capacity at any server, i.e., there is a *capacity constraint*. The capacity constraint may be a storage space constraint, a bandwidth constraint, and/or a processing capacity constraint. The

worst-case performance bounds for capacitated versions of the minimum K-median problem are considerably worse than for the non-capacitated versions [4] [6].

The minimum K-median problem was studied for networks with both tree topologies [11] [9], and general-graph topologies [13] [10]. In a tree topology model, the root of the tree is the origin server, which is always a replication site too. Each CDN site is represented by a node in the tree. The replicas can be placed on any node in the tree. The optimal placement problem for general-graph topology is strictly NP-complete, whereas for tree topology the solution can be obtained in polynomial time when taking into consideration the read, write, and storage costs [11]. Although the Internet is generally not a tree topology, placement in tree topology networks is still worth studying. Some network services such as DNS indeed exhibit a tree topology. In addition, a general-graph topology may be approximated by a tree topology; the optimal replica placement solution for the tree topology can be then applied toward the general-graph topology as an approximation.

Replica placement can be done for the entire data [13] [9], i.e. each replication site has a full copy of the data. The placement can also be done with data object granularity [11] [10], i.e., each replication site may contain a subset of the data objects. The data object replication approach is supposed to be more efficient and more flexible. For example, when a client needs to access a DVD movie, only that particular movie object needs to be replicated by its assigned server. With falling storage costs, in many cases it may be possible to replicate the entire content at all replicas in the future. If storage capacity constraints are to be considered, they only apply to the data object replication model. Bandwidth constraints and processing power constraints apply to both the full replication and data object replication models.

When the replica placement problem is solved using algorithms for the minimum K-median, some clients may suffer a considerably longer latency to their assigned servers than other clients. The major four contributors to network latency are:

- Propagation: The time it takes for a packet to travel between one place and another.
- Serialization: The medium itself (whether optical fiber, wireless, or some other) introduces some delay. The size of the packet introduces delay in a round trip since a larger packet will take longer to receive and return than a short one.
- Router processing and queuing delay: Each gateway node takes time to examine and possibly change the header in a packet (for example, changing the hop count in the time-to-live field).
- Other computer and storage delays: Within networks at each end of the journey, a packet may be subject to storage and hard disk access delays at intermediate devices such as switches and bridges.

For interactive applications, or any application where response time needs to be bounded to meet user expectations, this kind of latency variation may be unacceptable. Examples of such applications include streaming audio and video, e-commerce, game playing, and others. In these situations, it is desirable to bound the latency experienced by any individual client to a fixed value. Figure 1 illustrates such a problem for a trivial network topology. For the network in Figure 1, a solution for the minimum K-median placement problem with $K = 2$ would choose sites 1 and 2 as servers, to achieve a minimum total cost of 30. However, the latency of the client connected to site 3 to either server would be 10. An algorithm placing replicas with a latency constraint of 2 might choose sites 1 and 3 as servers. This guarantees the latency of any client to its closest server is no greater than 2. Finding solutions such as this calls for algorithms that place replicas with latency constraints. This particular problem has both non-capacitated and capacitated versions. The problem can also be studied for full replication, or per-object granularity.

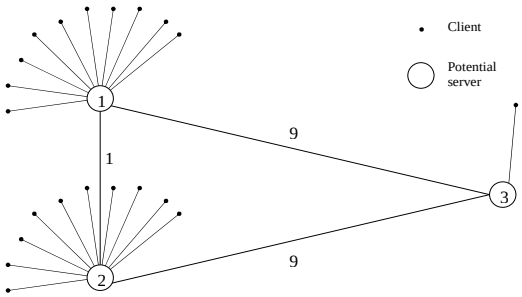

Figure 1: A network topology demonstrating the need for replica placement with latency constraints. The number on each link denotes the length (also the delay) of that link. Where there is no number, the length is 1. The traffic demands of all clients are assumed to be equal, and normalized to 1.

A recent study by Yan Chen et al. [5] partly addressed this problem. These authors proposed algorithms to dynamically place close to the minimum number of replicas, while meeting the clients' latency constraints and servers' capacity constraints, for peer-to-peer networks. The servers are then organized into an application-level multicast tree with small delay and bandwidth consumption for update dissemination. This approach leverages Tapestry [14] to improve the scalability, as Tapestry permits clients to locate nearby replica servers without contacting a root. Two algorithms were proposed, one being the naive placement algorithm, another being the smart placement algorithm. The smart approach attempts to optimize the "best" server selection for a joining client in a larger set of nodes, thus yielding a smaller set of servers in general. Their algorithms are designed specifically for peer-to-peer communication networks, and may be characterized as on-line or dynamic algorithms. It is unclear if and how they would work for general purpose networks where the problem instance was fully and statically specified. In this paper, our main focus is on replica placement algorithms with latency constraints for full content replication on general-graph topology networks. Both capacitated and non-capacitated versions of the problem are studied.

3. PROBLEM FORMULATION

We consider a content distribution network with N nodes that are potential sites for placing replicas. Each potential site in the network is given a site ID. Among the N nodes, there are R nodes that have aggregated requests from their clients, i.e. there are R aggregated client requests. Each client request r_i has a latency constraint l_i. The latency constraint imposed on any node is the minimum of the latency constraints of all its clients' requests. Symbols and their meanings are summarized in Table 1.

Table 1: Symbols and Their Meanings

Symbol	Meaning
N	The number of nodes in the network
s_i	Server node i.
R	The number of aggregated client requests in the network.
r_i	The aggregated client request on node i.
$d_{i,j}$	The distance between node i and node j, the distance can be latency, hop count, link cost etc.
l_i	Latency constraint on node i.
p_i	Server i's capacity constraint, i.e., the maximum number of aggregated requests that can be served by server i.

We say a request r_i can be *covered* by a server node s_j iff $d_{i,j} \leq l_i$. The *popularity* of a potential server node is defined as the number of requests that it covers. The goal of the replica placement with latency constraint is to find a smallest set of servers such that each request is covered by at least one server node. For simplicity, we assume that each request has the same latency constraint l, i.e. $l_i = l$, $1 \leq i \leq N$.

3.1 Proving NP-completeness

In order to prove that our optimization problem is NP-complete, we show that it belongs to NP and then we reduce the minimum dominating set problem to a special case of our problem.

The minimum dominating set problem is defined as follows. Given graph $G(V, E)$, find a dominating set for G, i.e., a subset $V' \subseteq V$ such that for all $u \in V - V'$ there is a $v \in V'$ for which $(u, v) \in E$. The goal is to minimize the cardinality of the dominating set, i.e., $|V'|$.

The replica placement with latency constraint is easily seen to be in NP. Given a placement and the latency constraint l, we can verify in polynomial time whether the placement results in a complete coverage of all requests, i.e. each request is covered by at least one server.

Next, we show the reduction of the minimum dominating set problem to replica placement with latency constraint problem by considering a graph $G(V, E)$ in which we want to find a minimum dominating set V'. We construct a network in the following way: For each node $v \in V$, construct a node denoted v_n in the network, the node is both a potential server and a client with request. For each edge $e \in E$ connecting $v \in V$ and $u \in V$, construct a bi-directional link connecting v_n and u_n in the network, the length (latency) of the link is l. Now solve the minimum replica with latency constraint problem for the network, with latency constraint = l. Clearly the minimum replica selection that meets the latency constraint for all clients in the network is the minimum dominating set for the graph G.

Given our placement problem belongs to NP and the minimum dominating set problem reduces to it, we know our placement problem is NP-complete.

4. PLACEMENT ALGORITHMS

In this section, we present three algorithms for placing replicas with latency constraints, while retaining efficient and balanced resource consumption of the underlying infrastructure. We define the cost of a request from node i to node j as the distance between the two nodes, where the distance can reflect any performance metric we want to bound, 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. Without losing generality, we focus on latency constraints in the following discussion.

For the dominating set problem, the best known approximation is achieved by a greedy algorithm, with a worst-case ratio of $\ln\Delta$ [8] [12], where Δ is the maximum node degree in the graph. There is no polynomial algorithm with better performance unless $P = NP$ [7]. Besides a greedy algorithm, we also present two other algorithms. One, the *popularity algorithm*, is more scalable and may be suitable for online or dynamic execution. The other, the *random* algorithm, is presented for purposes of comparison.

4.1 Greedy Algorithm

We need to choose a replica for each of the aggregated client requests. In the greedy approach, we choose one replica at a time. Initially, all clients are unmarked and no nodes are selected as servers. In each iteration, we evaluate each of the N potential sites to determine their suitability for hosting a replica. For each site not yet selected to host a replica, its popularity among the clients not yet marked (assigned to a server) is computed. The potential site that has the largest popularity is then selected to be a server, and all clients covered by that site are marked. This process continues until all clients are marked as being assigned to a replica server.

In the non-capacitated version of the problem, we proceed as above, assigning to the new server all the clients that it covers. In the capacitated version of the problem server i's capacity constraint p_i is imposed. That is, only the first p_i clients that are covered by server i are assigned to it. For this purpose, requests are ordered in ascending order by their site ID.

4.2 Popularity Algorithm

The greedy algorithm can only be implemented in a centralized manner, with complete knowledge about client requests and latency constraints. In some environments it may be preferable not to have to collect this information centrally. It may also be desirable to support distributed execution for purposes of computing dynamic solutions, i.e., solutions in which the previous assignment of clients to servers is retained as much as possible. We now present such a method.

The basic idea is to allow each client to choose independently who its replica server will be. Each client announces its presence by broadcasting to all potential sites within a distance that meets its latency constraint. We assume a TTL value can be used for this purpose. Once broadcasting has completed, each potential site can compute its popularity from this information. Each potential site then communicates (unicasts) its popularity back to the clients from whom it received such announcements. We will assume sufficient time elapses to receive all responses back from potential sites. Each client then chooses from among the potential sites it reached the one with the greatest popularity. For purposes of this algorithm, ties are broken in order of ascending site ID.

The capacitated version of the problem requires additional complexity, potentially involving multiple rounds of communication between clients and potential sites. This possibility is not discussed further in this paper.

The popularity algorithm does not require centralized collection of client information, or computation of the placement of replicas and assignment of clients to servers. Distributed execution does, however, require more communication, and sufficient time for communication to complete.

4.3 Random Algorithm

For purposes of comparison, we also describe a random algorithm. In this approach, we also choose one replica site at a time. Initially all clients are unmarked and no nodes are selected as servers. In each iteration, one site is chosen at random from among the sites which have not yet been selected. The clients covered by that site are marked. If a randomly chosen site does not cover any clients which are unmarked, the site is discarded as a server and the next site is chosen at random, until one which covers at least one unmarked client is selected. This process continues until all clients are marked as being assigned to a replica server.

In the non-capacitated version of the problem, we proceed as above, assigning to the new server all the clients that it covers. In the capacitated version of the problem only the first p_i clients that are covered by server i are assigned to it.

4.4 Computational Complexity

The trivial computational complexity bounds for non-optimized versions of these algorithms are as follows: $O(N^3)$ for the greedy algorithm; $O(N)$ on each site for the popularity algorithm; and $O(N^2)$ for the random algorithm.

When the network only has a small number of nodes, the computational complexity of the various algorithms may not be significant. However, for a large scale CDN with thousands of nodes, the computational complexity of the placement algorithms becomes a significant factor.

5. SIMULATION

Since the above algorithms are heuristics, it is necessary to evaluate their performance experimentally. In this section, we evaluate the performance of the three algorithms. We used the GT-ITM [2] transit-stub model to generate network topologies containing 5000 nodes. The transit-stub model produces hierarchical graphs from transit and stub domains. It first constructs a connected random graph; each node in that graph represents an entire transit domain. Each node in that graph is then replaced by another connected random graph, representing the backbone topology of one transit domain. Finally, for each node in each transit domain, it generates a number of connected random graphs representing the stub domains attached to that node [2]. Among the 5000 nodes, unless stated otherwise 1000 nodes are chosen randomly as clients, each aggregating a set of requests. Unless specifically noted, the request distribution is generated uniformly across the whole network. Each simulation is done for at least five different request patterns. For simplicity and without loss of generality, we define a server's capacity constraint as the maximum number of aggregated requests that server can serve. Under this definition, the actual number and size of each client request becomes irrelevant. We also assume that each server has the same capacity constraint cap, i.e. $p_i = cap$, $1 \leq i \leq N$.

Our simulation code is written in C++ as an extension of LEDA, the library of efficient data types and algorithms [1].

In the following, we only consider one performance metric: the number of replicas needed to fulfill the latency constraint for all the requests. Figure 2 shows the number of replicas required for various latency constraints for the three algorithms. This experiment did not consider server capacity constraints. The greedy algorithm yields the smallest set of replicas among all three algorithms. The improvement of the greedy algorithm over the random algorithm ranges from 20% to almost 50%, justifying its use. The popularity algorithm is attractive for smaller latency constraints, since it executes in a distributed fashion and has good performance. However, the quality of the solutions produced by the popularity algorithm degrades considerably for larger latency bounds. The reason for the degraded performance of the popularity algorithm at larger latency bounds is because that the popularity algorithm lacks global coordination on server selection, whereas the greedy algorithm coordinates the server selection among all requests.

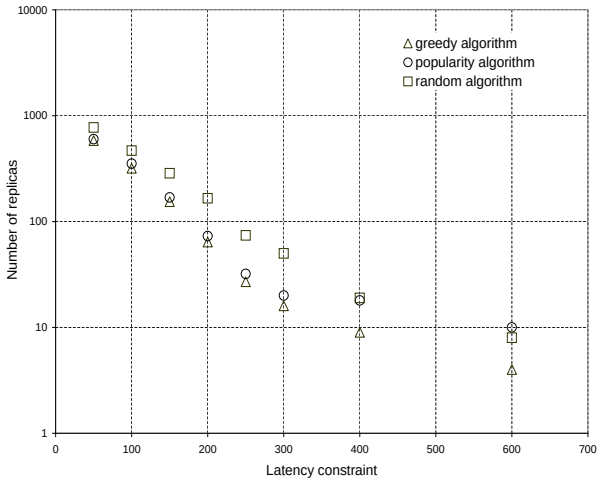

Figure 2: Number of replicas required for varying latency constraints, without considering server capacities. The topology is the GT transit-stub model with 5000 nodes, and 1000 aggregated requests. The confidence intervals for all points are within $\pm$5%.

Figure 3 shows the same experiment, but this time for the case of servers with capacity constraints. In this experiment, each site had a capacity that limited it to serving at most 50 aggregated requests. When the latency constraint becomes large enough (above 300 in Figure 3), the minimum number of replicas is determined by the servers' capacities (e.g., 20 servers in this case). When the latency constraint is smaller than this, the number of replicas required in the capacitated cases are only slightly more that that of the corresponding non-capacitated cases. This is because satisfying the latency constraint requires far more servers than satisfying the capacity constraint would require.

We also investigated the case in which information about client requests is not available. In such a case, it can simply be assumed that all nodes may have client requests. We ran an experiment in which there were 5000 simultaneous aggregated requests, one for each node in the 5000-node network. This scenario is of practical interest because the objective is to have all possible requests covered by at least one replication site. If the client request pattern is unknown, as it generally is, it will still be possible to cover all client requests. The non-capacitated results of our three algorithms under this circumstance are listed in Figure 4. The signficance of these results may be found by comparing Figures 2 and 4. The number of servers needed to satisfy the latency constraints almost doubles from the case of 1000 requests. This implies that the additional flexibility needed to satisfy all possible client requests can be obtained at a reasonable cost.

As stated above, most previous research on the replica placement problem (without latency constraints) has focused on the minimum

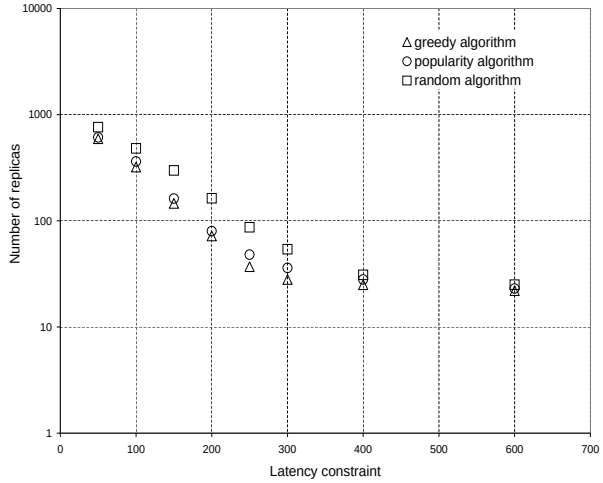

Figure 3: Number of replicas required for varying latency constraints, with each server's capacity constraint $cap = 50$. The topology is the GT transit-stub model with 5000 nodes, and 1000 aggregated requests. The confidence intervals for all points are within $\pm 5\%$.

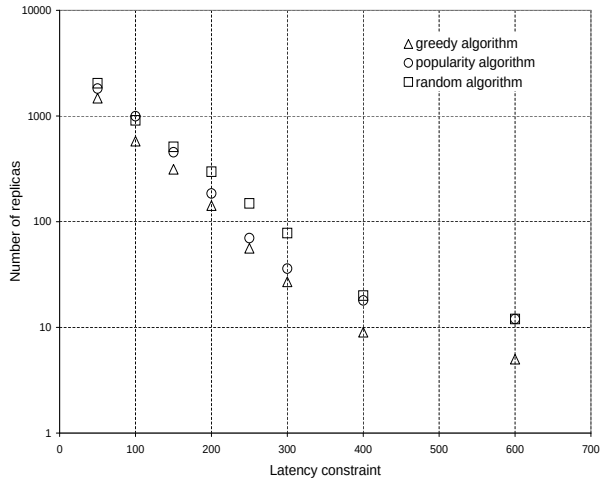

Figure 4: Number of replicas of various replica placement algorithms with latency constraint without considering the servers' capacity constraint. Topology is the GT Transit-stub model with 5000 nodes, and 5000 aggregated requests.

K-median problem. The goal of this problem is to minimize overall cost for a target number K of servers. It is appropriate to investigate how far the solution to the K-median problem deviates from the goal of achieving a given latency bound. Since the minimum K-median problem does not include a latency bound, we created a fair comparison in the following way.

In our simulation, we first computed the number of replicas required for satisfying a given latency constraint, using the greedy and popularity algorithms. We then used the result for the greedy algorithm as the value of K, and solved the minimum K-median problem for this value. Since the minimum K-median problem is NP-complete, we used a greedy approximation due to [13] to solve this problem. The results are shown in Figure 5. This experiment compares the *average* latency of the client requests for the three algorithms, for the same number of replica sites. Since the minimum K-median problem minimizes the total cost, which in our formulation is also the total delay, it may be expected that the average latency for those solutions is very good. The figure shows that our random and greedy heuristics have an average latency which is generally 5% to 10% worse than the solution of the minimum K-median problem, which seems quite acceptable.

Bounding the *maximum* latency is the justification for our algorithms and is accomplished 100% of the time. We also investigated the maximum latency produced by the greedy solution to the minimum K-median problem. The conditions were the same as above, except that we modelled a nonuniform distribution of client requests among the CDN nodes. In a nonuniform distribution of client requests, we assign 1% of the total number of aggregated requests (10 requests in this case) randomly to the nodes in one of the transit domains and all stub domains attached to that transit domain in the graph. The remaining 99% of the total number of aggregate requests (990 requests) are assigned randomly to the remainder of the graph. The results are shown in Figure 6. The uniform request cases in Figures 5, 6, and 7 are different measurements of the same experiment. Under these conditions, the solution to the K-median problem has a maximum client latency as much as 300% greater than the latency bound specified for our algorithms, with the same number of replica sites selected. When the requests are distributed uniformly in the network, the solution of the minimum K-median problem comes much closer to meeting the specified latency bound, as also shown.

We also measured the number of requests whose latency constraint is not met, for the solutions of the K-median problem. These measurements are graphed in Figure 7. The number of requests not meeting the latency bound decreases with an increase in the latency constraint. The number has a maximum of about 10% of the requests for a stringent latency constraint of 50. The measurements for the non-uniform and uniform cases are fairly close for this metric.

Although the number of requests that fail to meet their latency bound for the solution to the minimum K-median problem may not be large, the high latency experienced by some of those requests may be unacceptable for certain applications.

6. CONCLUSIONS

In this paper, we proposed that bounding the latency of client requests may be an important factor in solving the replica placement problem for content distribution networks. We proposed two algorithms for placing replicas with latency constraints efficiently, one centralized, and one distributed. We showed the impact on the number of replicas required as the latency constraint becomes more stringent. In the case where client request patterns are unknown, we showed the additional number of replicas needed is reasonable. Our solutions to the replica placement problem with latency bounds are generally only slightly more expensive than solutions without latency bounds. The conventional solution of the minimum K-median problem does not produce acceptable results for the case of clients that have stringent latency constraints.

7. ACKNOWLEDGMENTS

The second author's work has been supported by the DARPA FTN program under contract F30602-99-1-0540. The views and conclusions in this paper do not represent the policies of the US government.

8. REFERENCES

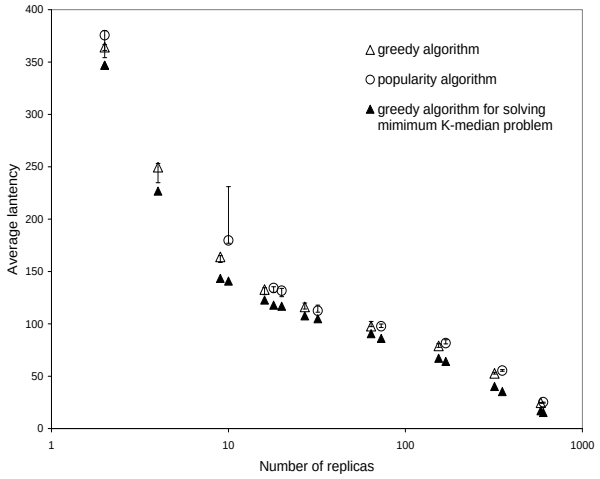

Figure 5: Average latency of various replica placement algorithms, without considering server capacity constraints. The topology is the GT transit-stub model with 5000 nodes, and 1000 aggregated requests. The lower and upper bound and center mark of each errorbar correspond to the minimum, maximum, and median, respectively, of the average latency.

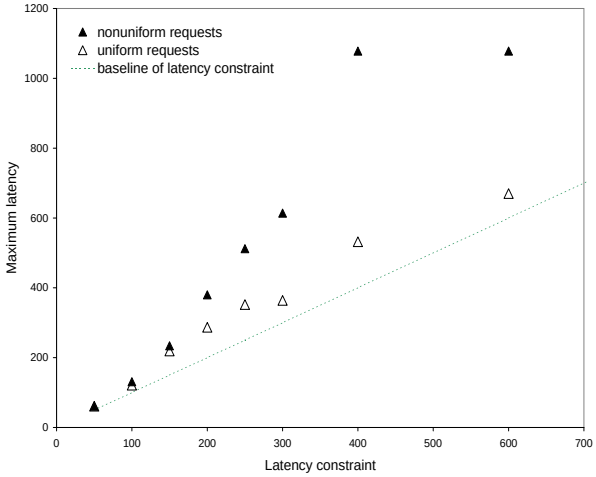

Figure 6: Maximum latency of the greedy algorithm for solving the minimum K-median problem, without considering server capacity constraints. The topology is the GT transit-stub model with 5000 nodes, and 1000 aggregated requests.

[1] *LEDA homepage. http://www.mpi-sb.mpg.de/LEDA/leda.html*.

[2] K. Calvert and E. Zegura. Gt internetwork topology models (gt-itm). *http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm*.

[3] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings Of the 40th Annual IEEE Conference on Foundations of Computer Science*, pages 378–388, 1999.

[4] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceeding of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.

[5] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Peer-to-Peer Systems: First International Workshop, IPTPS 2002*, pages 306–318, March 2002.

[6] F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for the capacitated facility location problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages S875–S876, 1999.

[7] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[8] D. S. Johnson. Approximation algorithms for combinatarial problems. *Journal of Computers and System Sceince*, 9:256–278, 1974.

[9] K. Kalpakisa, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions of Parallel and Distributed Systems*, 12(6):628–637, June 2001.

[10] J. Kanashrrju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25:376–383, 2002.

[11] B. Li, M. J. Golin, G. F. Italiano, and X. Deng. On the optimal placement of web proxies in the internet. In *Proceedings Of IEEE INFOCOM'99*, pages 1282–1290, March 1999.

[12] L. Lovasz. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[13] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings Of IEEE INFOCOM 2001*, pages 1587–1596, 2001.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technical Report UCB/CSD-01-1141, UC Berkeley*, April 2001.

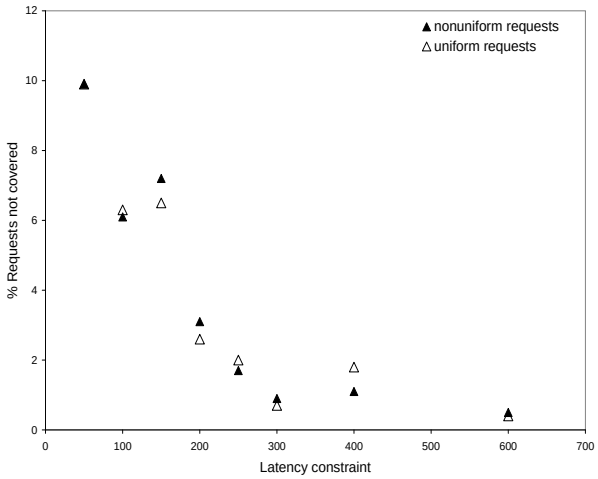

Figure 7: Percentage of requests not covered by using greedy algorithm for solving the minimum K-median problem, without servers' capacity constraint. Topology is the GT Transit-stub model with 5000 nodes, and 1000 aggregated requests.