Replicated Server Placement with QoS Constraints

Georgios Rodolakis

Hipercom Project, INRIA Rocquencourt,

78153 Le Chesnay Cedex, France

Georges.Rodolakis@inria.fr.

Stavroula Siachalou, Leonidas Georgiadis

Electrical and Computer Engineering Dept.,

Aristotle University of Thessaloniki, Greece.

ssiachal@auth.gr, leonid@auth.gr.

Abstract

The problem of placing replicated servers with QoS constraints is considered. Each server site may consist of multiple server types with varying capacities and each site can be placed in any location among those belonging to a given set. Each client can de served by more than one locations as long as the request round-trip delay satisfies predetermined upper bounds. Our main focus is to minimize the cost of using the servers and utilizing the link bandwidth, while serving requests according to their delay constraint. This is an NP-hard problem. A pseudopolynomial and a polynomial algorithm that provide guaranteed approximation factors with respect to the optimal for the problem at hand are presented.

1 Introduction

In today's Internet there is an explosive growth of the World Wide Web. As the Web is becoming a widely accepted medium for distributing data and all kind of services, network traffic increases due to the growing number of Web users that access different sites. In such an environment, it is important to satisfy user requirements for Web access and to optimize system performance. This is achieved by providing Web access response times that satisfy QoS requirements and by minimizing the total cost of using the servers and utilizing the link bandwidth. Improvement of the Web service performance can be achieved by replicating servers at appropriately selected sites.

The problem of maximizing the performance and minimizing the cost of a computing system has been addressed in the past in several papers. Krishnan et al [1] developed polynomial optimal solutions to place a given number of servers in a tree network to minimize the average retrieval cost

of all clients. Li et al [2] investigated the placement of a limited number of Web proxies in a tree so that the overall latency for accessing the Web server is minimized. In [3] two objectives were studied: minimization of the overall access cost by all clients to access the Web site and minimization of the longest delay for any client to access the Web site. The problem was reduced to the placement of proxies in a set of trees whose root nodes are the server replicas. Jia et al [4] took the read and update operations into consideration. Qiu et al [5] also assumed a restricted number of replicas and no restriction of the number of requests served by each replica. A client could be served by a single replica and the cost for placing a replica was also ignored. The objective was to minimize the total cost for all clients to access the server replicas, while the cost of a request was defined as the delay, hop count or the economic cost of the path between two nodes. They compared several heuristic solutions and found that a greedy algorithm had the best performance. Chen et al [6] tackled the replica placement problem from an other angle: minimizing the number of replicas when meeting clients' latency constraints and servers' capacity constraints by using a dissemination tree. In [7] the authors considered the problem of placing a set of mirrors only at certain locations such that the maximum distance from a client to its closest mirror (server replica), based on round trip time, is minimized. They assumed no cost for placing a mirror and showed that placing mirrors beyond a certain number offered little performance gain. Sayal et al presented some selection algorithms to access replicated Web servers in [8]. The algorithms found the closest replicated server for a client based on different metrics such as hop counts, round trip time and the HTTP request latency. In [9] the objective was to minimize the amount of resources, storage and update, required to achieve a certain level of service. They assumed that all servers in the network are organized into a tree structure rooted at the origin server.

In this paper we approach the problem of replicated server placement with QoS constraints from a system administrator's perspective. Thus we are interested in serving the users' requests so that their delay requirements are satisfied, while at the same time minimizing the total cost of placing the servers and using the link bandwidth. We consider that each server site may be comprised of a number of different server types with varying processing capacities. We also assume that a server site may by placed on any of a given set of locations and each client can be served by more than one location. Our objective is to select optimally the locations where the servers must be placed, the types that comprise each server, the proportion of traffic that must be routed by each client to each of the servers and the routes that will be followed by the requests issued by a client to a

server. The problem is NP-hard and therefore an optimal solution is not likely to be found. We present a pseudopolynomial approximation algorithm and a polynomial time algorithm that provide guaranteed approximation factors with respect to the optimal for the problem at hand.

Due to space limitation proofs are omitted. For a complete version of the paper the interested reader is referred to the site http://genesis.ee.auth.gr/georgiadis/english/public/QoS_IP2005.pdf.

2 Problem Formulation

Let $G(V, E)$ represent a network with node set V, and link set E. Let also H be a subset of V. We are interested in placing servers at some of the nodes in H, that will serve requests originated by any of the nodes in V. We assume that the servers contain the same information and hence any node may obtain the requested information by accessing any of the servers.

With link (i, j) there is an associated delay d_{ij}. Requests should be obtained in a timely fashion, and hence there is a bound D on the time interval between the issuing of the request and the reception of the reply. We refer to this bound as the "round-trip delay bound". Note that the processing time of a request at the server can be incorporated in this model by replacing D with $D + d_p$, where d_p is an upper bound on the request processing time at the server.

The load in requests per unit of time originated by node $i \in V$ is g_i. To transfer an amount of x requests per second, it is required to reserve bandwidth αx on the links traversed by the requests. The transfer of server replies corresponding to the x requests back to the requesting node, requires the reservation of βx units of bandwidth on the links traversed by the replies.

The cost of transferring 1 unit of bandwidth on link (i, j) is e_{ij}. Hence the cost of transferring x requests per second on link (i, j) is $\alpha e_{ij} x$ while the cost of transferring the replies to these requests is $\beta e_{ij} x$. A node i can split its load g_i to a number of servers and routes as long as the delay bound D between the issuing of the request and the reception of the reply is satisfied. At each node $j \in H$ there is a set S_j of server types that can be selected. Server type s, $1 \leq s \leq K_j$, $(K_j = |S_j|)$ costs f_j^s units and can process up to U_j^s requests per second.

Our objective is to determine,

1. the locations (subset of the nodes in H) where the servers will be placed,
2. the amount of traffic (in requests per unit of time) that will be routed by each node to each of the selected locations,
3. the routes through which the node traffic will be routed to each of the selected locations,

4. the type of servers and the number of each type that should be opened at each location,

so that,

1. the round-trip delay bound for each request is satisfied,
2. the total cost of using the servers and utilizing the link bandwidth is minimized.

Notice that in the current setup we do not consider link capacities. In effect we assume that the network links have enough bandwidth to carry the requested load by the network nodes. This is a reasonable assumption in an environment where the server requests are a small portion of the total amount of information carried by the network. The general problem where link capacities are also included, is a subject of further research.

2.1 Optimization Problem Formulation

A feasible solution to the problem consists of the following:

- A set of locations $F \subseteq H$ where the servers will be placed.
- A subset of server types $G_j \subseteq S_j$ that should be opened at location $j \in F$.
- The number n_j^s of server types $s \in G_j$ that should be opened at location $j \in F$.
- A set of round-trip routes R_{ij} between node $i \in V$ and facility $j \in F$. A round-trip route, denoted $r_{ij} = (p_{r_{ij}}, q_{r_{ij}})$, consists of two simple paths, $p_{r_{ij}}$ and $q_{r_{ij}}$. Path $p_{r_{ij}}$ originates at node i and ends at server location j, and is used for transferring requests. Path $q_{r_{ij}}$ originates at server location j and ends at node i, and is used for transferring replies.
- The amount of requests per unit of time, $x_{r_{ij}}$, accommodated on round-trip route r_{ij}.

The constraints of the problem are the following:

- The request load of each node should be satisfied. That is,

$$\sum_{j \in F} \sum_{r \in R_{ij}} x_r = g_i, \ i \in V. \tag{1}$$

- The round-trip delay of each round-trip route should be at most D. That is,

$$\sum_{l \in p} d_l + \sum_{l \in q} d_l \leq D, \text{ for } r = (p, q) \in R, \tag{2}$$

where R is the set of all round-trip routes, $R = \cup_{i \in V} \cup_{j \in F} R_{ij}$, and the summation is over all links of the corresponding paths.

- The total server capacity at server location $j \in H$ should be at least as large as the request

rate arriving at location j. That is,

$$\sum_{i \in V} \sum_{r \in R_{ij}} x_r \leq \sum_{s \in G_j} n_j^s U_j^s, \ j \in H. \tag{3}$$

The objective cost function to be minimized is

$$\sum_{j \in F} \sum_{s \in G_j} n_j^s f_j^s + \sum_{l \in E} e_l (\alpha \sum_{\substack{r=(p,q) \in R \\ l \in p}} x_r + \beta \sum_{\substack{r=(p,q) \in R \\ l \in q}} x_r). \tag{4}$$

The first term in (4) corresponds to the cost of opening the servers, while the second term corresponds to the cost of reserving bandwidth on the network links in order to satisfy the node requests. The term involving the factor α corresponds to the bandwidth reserved on a link for transmission of node requests, while the term involving the factor β corresponds to the bandwidth reserved on the same link for transmitting replies.

For the rest of the paper we assume that the node loads g_i, $i \in V$ are nonnegative integers and that splitting of these loads to a number of server locations may occur in integer units. In practice this is not a major restriction, since usually the load is measured in multiples of a basic unit.

2.2 Problem Decomposition

In this section we decompose the problem defined in Section 2.1 into three subproblems which can be solved independently. As will be seen all three problems are NP-hard.

For a round-trip route $r = (p, q)$, define the cost $C_r = \alpha \sum_{l \in p} e_l + \beta \sum_{l \in q} e_l$. Let r_{ij}^* be a minimum-cost round-trip route between node i and server location j, satisfying the round-trip delay D. It can be easily seen that it suffices to restrict attention to solutions that assign all the load from node i to server location j on r_{ij}^*. Hence, setting $c_{ij} = C_{r_{ij}^*}$, the second term in (4) becomes $\sum_{i \in V} \sum_{j \in F} c_{ij} x_{ij}$.

Consider now the first term in (4). Let $f_j(y)$ be the minimum cost server type assignment at location j, under the assumption that the request load at that location is y. By definition, the feasible solution that assigns this minimum cost server assignment at location j for request load $y_j = \sum_{i \in V} \sum_{r \in R_{ij}} x_r$, is at least as good as a solution inducing the same load at location j. Hence, we may replace this term with $\sum_{j \in F} f_j(y_j)$.

For our purposes, it is important to observe that the function $f_j(y)$ defined in the previous

paragraph is subadditive, i.e., it satisfies the inequality

$$f_j(y_1) + f_j(y_2) \geq f_j(y_1 + y_2) \text{ for all } y_1 \geq 0,\ y_2 \geq 0. \tag{5}$$

To see this, note that if $S(y_1)$, $S(y_2)$ is the set of servers achieving the optimal costs $f_j(y_1)$, $f_j(y_2)$ respectively, then the set of servers $S(y_1) \cup S(y_2)$ provides a feasible solution for request load $y_1 + y_2$, with cost $f_j(y_1) + f_j(y_2)$. Since $f_j(y_1 + y_2)$ is by definition the minimum cost server assignment with request load $y_1 + y_2$, (5) follows.

From the discussion above it follows that we need to solve the following problems.

Problem 1. Given a graph, find a round-trip route with minimum cost, satisfying the round-trip delay bound for any node $i \in V$ and server location $j \in H$. This determines $c_{ij},\ i \in V, j \in H$.

Problem 2. Given a set of server types S_j and a required load y at node $j \in H$, find the optimal selection of server types and the number of servers of each type so that the load is accommodated. That is, determine $n_j^s(y)$ so that $\sum_{s \in G_j} n_j^s(y) U_j^s \geq y$ and $f_j(y) = \sum_{s \in G_j} n_j^s(y) f_j^s$ is minimal.

Problem 3. Given non-decreasing subadditive functions $f_j(y)$, costs c_{ij}, integer node loads $g(i) \geq 0,\ i \in V$, solve

$$\min \sum_{j \in H} f_j(y) + \sum_{i \in V} \sum_{j \in H} c_{ij} x_{ij}$$

$$\text{subject to: } \sum_{j \in H} x_{ij} = g(i),\ i \in V,\ \sum_{i \in V} x_{ij} = y,\ j \in H,\ x_{ij} \geq 0.$$

Remark. Regarding Problem 1, in certain environments it may be desirable to have a delay bound only on the route from the server to the requesting node. For example, this may be the case if the requests concern the viewing of a video clip. Problem 1 can be easily modified to account for this type of environment. In fact, the problem is now simpler, since the delay constraint concerns only the one-way end-to-end delay.

The decision problems associated to Problems 1 and 2 are NP-hard. Indeed, when $\beta = 0$, the associated decision problem to Problem 1 is reduced to the Shortest Weight-Constrained Path problem which is known to be NP-hard [10]. Also, the associated decision problem to Problem 2 amounts to the Unbounded Knapsack Problem which is NP-hard [11]. However for both problems pseudopolynomial algorithms exist (see Section 3) and, as will be discussed in Section 4, fully polynomial time approximation algorithms can be developed. Regarding Problem 3, there is an extensive work in the

literature under various assumptions on the function $f_j(y)$ and on the costs c_{ij} ([12], [13], [14], [15], [16], [17], [18]). Most of the work is concentrated on the case of "metric" costs, i.e., it is assumed that costs satisfy the inequality $c_{ij} + c_{jk} \geq c_{ik}$. However, this inequality does not hold in our case.

In the next section, by combining algorithms for the three problems discussed above, we provide a pseudopolynomial time approximation algorithm for the problem addressed in this paper. The algorithm for Problem 3 is based on the algorithm proposed in [18] and uses the fact that $f_j(y)$ is subadditive step function. In Section 4 we modify the algorithm in order to obtain a polynomial time algorithm with approximation factor close to the best possible (unless NP$\subseteq$DTIME($n^{O(\log\log n)}$). The performance of the algorithm in simulated networks is studied in Section 5.

3 Pseudopolynomial Algorithm

In this section we discuss pseudopolynomial algorithms for each of the Problems 1, 2 and 3. By combining theses algorithms we get a pseudopolynomial algorithm for the problem at hand.

3.1 Pseudopolynomial Algorithm for Problem 1

Let $F_{ij}(d)$ be the minimum cost (forward) path from node i to j with delay at most d, and $B_{ij}(d)$ the minimum cost (backward) path from node j to i with delay at most d. Then it can be easily seen that,

$$c_{ij} = \min_{0 \leq d \leq D} \{F_{ij}(d) + B_{ij}(D-d)\} \tag{6}$$

Based on (6), c_{ij} can be determined provided $F_{ij}(d)$ and $B_{ij}(d)$ are known. There are fully polynomial time algorithms for computing these quantities, however in this section we will concentrate on efficient pseudopolynomial algorithms that work well in practice [19], [20]. We provide the discussion for $F_{ij}(d)$, since the same holds for $B_{ij}(d)$. The algorithm in [19], [20] are based on the fact that $F_{ij}(d)$ is a right continuous non-increasing step function with a finite number of jumps. Hence, in order to compute $F_{ij}(d)$ one needs only to compute its jumps, which in several practical networks are not many. Another useful feature of these algorithms is that in one run they compute $F_{ij}(d)$ from a given node $j \in H$ to all other nodes in V.

Let $d_{ij}^f(k)$, $1 \leq k \leq K_{ij}^f$, $i \in V$ be the jump points of $F_{ij}(d)$ such that $d_{ij}^f(k-1) \leq d_{ij}^f(k) \leq D$, $k = 2, .., K_{ij}^f$. Similarly, let $d_{ij}^b(k)$, $0 \leq k \leq K_{ij}^b$, $i \in V$ be the jump points of $B_{ij}(d)$. The optimal round-trip costs c_{ij}, $j \in H$, $i \in V$ can be computed using Algorithm 1. The jumps in steps 2 and 3

can be computed using the algorithm in [19]. The "for" loop in step 6 implements the minimization required by (6).

Algorithm 1 Algorithm for finding the minimum cost round-trip route

Input: Graph G with link costs and delays.
Output: The array c with the costs of the round-trip routes.

1. For any node j in H do
2. Compute jump points of $F_{ij}(d)$, $d_{ij}^f(k)$, $1 \le k \le K_{ij}^f$, $i \in V$
3. Compute jump points of $B_{ij}(d)$, $d_{ij}^b(k)$, $0 \le k \le K_{ij}^b$, $i \in V$
4. For $i \in V$ do
5. $c_{ij} = \infty$
6. For $k = 1$ to K_{ij}^f do
7. Let d_{ij}^b be the largest jump point of $B_{ij}(d)$ not exceeding $D - d_{ij}^f(k)$
8. $c_{ij} \leftarrow \min\left\{c_{ij}, F_{ij}(d_{ij}^f(k)) + B_{ij}(d_{ij}^b)\right\}$

Using the algorithm and the analysis presented in [19] it can be proved that the total worst case running time for the computation of the jump points is $O\left(|V|\, D\left(|V| \log |V| + |E| \log |V|\right)\right)$. The running time of the minimum operation (line 8) is $O\left(|V|^2 D\right)$. Thus the running time of this algorithm is dominated by the time needed to compute the jump points.

3.2 Pseudopolynomial Algorithm for Problem 2

We restate Problem 2 in its generic form, to simplify notation.

Problem 2 (generic form). Given a set of server types S, server capacities U^s, server costs $f^s > 0$ and a required load y, find the optimal selection of server types G and the number of servers of each type so that the load is satisfied. That is, determine $n^s(y)$ so that $\sum_{s \in G} n^s(y) U^s \ge y$ and $f(y) = \sum_{s \in G} n^s(y) f^s$ is minimal.

Problem 2 is similar to the Unbounded Knapsack Problem (UKP) [11]. The difference is that in UKP the inequality constraint is reversed and maximization of the cost $\sum_{s \in G_j} n^s(y) f^s$ is sought. A pseudopolynomial algorithm for Problem 2 can be developed in a manner analogous to the one used for UKP. Specifically, number the servers from 1 to $|S|$ and define $A(f, i)$ to be the largest load achievable by using some among the first i servers so that their total cost is exactly f. The entries of the table $A(f, i)$ can be computed in order of increasing i and f using the dynamic programming equation

$$A(f, i+1) = \min\{A(f, i), U^{i+1} + A(f - f^{i+1}, i+1)\}, \tag{7}$$

with $A(f, 0) = 0$ for all f, $A(f, i) = -\infty$ if $f < 0$, and $A(0, i) = 0$ for all $0 \le i \le K$. The optimal

server selection cost is then determined as $f(y) = \min\{f \mid A(f, K) \geq y\}$. By keeping appropriate structures one can also determine the server types and the number of servers of each type for achieving the optimal solution.

The function $f(y)$ has properties similar to those of $F_{ij}(d)$ and $B_{ij}(d)$. Specifically, it is a right continuous non-decreasing step function. Moreover, based on (7) and using an approach similar to [21], an efficient pseudopolynomial algorithm can be developed for finding the jump points of $f(y)$. Again, an important property in our case is that in one run of the algorithm, all jump points of $f(y)$ up to an upper bound can be determined. The running time of this approach can be bounded by $O(|S|\, y)$, where $|S|$ is the number of server types.

3.3 Pseudopolynomial Algorithm for Problem 3

In [18] a polynomial time algorithm $O(|V|^3 \log |V|)$ is provided for Problem 3 in the case of concave facility cost functions. It is assumed that the cost $f_j(y)$ of placing servers at node $j \in H$ to accommodate load y can be computed at unit cost and that all nodes have unit loads. It is shown that the proposed algorithm achieves an approximation factor of $\ln |V|$ compared to the optimal. In our case we have arbitrary integer node loads g_i and the functions $f_j(y)$ are subadditive and can be computed exactly only in pseudopolynomial time. As observed in [22] the assumption of unit loads can be removed by considering a modified network where node i is replaced with g_i nodes each having the same links (and link costs). However, now the algorithm becomes pseudopolynomial (even assuming unit costs for computing $f_j(y)$) since the number of nodes in the modified network can be as large as $|V g_{\max}|$, where $g_{\max} = \max_{i \in V} \{g_i\}$.

The approximability proof for general costs c_{ij} in [18] carries over without modification if $f_j(y)$ are subadditive rather than concave functions. Hence the approximability factor in our case becomes $\ln |V g_{\max}|$.

The algorithm in [18] is the only known one that can provide performance guarantees in terms of approximability to the optimal for general costs c_{ij}. Moreover, its worst case running time is among the best of the proposed algorithms. Hence, we will use the algorithm in [18] as the basis for our development. We present it below (Algorithm 2) adapted to our situation. For the moment we assume that $f_j(y)$ can be computed exactly at unit cost.

The algorithm performs a number of iterations. At each iteration a node j^* in H is selected and the load of some of the nodes in V is assigned to j^*. Let matrix $\psi(i, j)$ represent the total load from

node i assigned to server location j at the beginning of an iteration (i.e., the beginning of the while loop at Step 3). Hence the load of node i remaining to be assigned is $r(i) = g(i) - \sum_{j \in H} \psi(i,j)$.

A node such that $r(i) > 0$ is called unassigned. For server location $j \in H$ consider the unassigned nodes arranged in non-decreasing order of their costs $c_{i_s j}$, i.e., $c_{i_1 j} \leq c_{i_2,j} \leq .... \leq c_{i_m j}$. Let $R_j(n) = \sum_{s=1}^{n} r(i_s)$, $1 \leq n \leq m$, and $n_j(k) = \max\{n : R_j(n) \leq k\}$. Define also $l_j(k) = k - R_j(n_j(k))$.

The variable $load_j$ holds the total load assigned to node $j \in H$ at the beginning of an iteration. In step 5, the most economical (cost per unit of assigned load) load assignment for each of the server locations is computed. In steps 7 and 8, the server location with the minimum economical assignment is selected and the associated load is placed on this location. In steps 9 to 13 updating is taking place of the remaining loads of the nodes that will place their load on the selected server location.

Algorithm 2 Generic algorithm for solving Problem 3

Input: Graph G, the array c with the costs of the routes and the Knapsack list
Output: Locations and types of servers, load assigned from each client to the selected locations.

1. For $j \in H$ set $load_j = 0$;
2. For $i \in V$ set $\psi(i,j) = 0$;
3. While there is an unassigned node do
4. For $j \in H$ do
5. $t(j) = \min_k \frac{f_j(load_j+k) - f_j(load_j) + \sum_{s=1}^{n_j(k)} r(i_s) c_{i_s j} + l_j(k) c_{i_{n_j(k)+1} j}}{k}$
6. $k(j) = \arg\min_k \frac{f_j(load_j+k) - f_j(load_j) + \sum_{s=1}^{n_j(k)} r(i_s) c_{i_s j} + l_j(k) c_{i_{n_j(k)+1} j}}{k}$
7. Let $j^* = \arg\min_{j \in H} \{t(j)\}$
8. Set $load_{j^*} \leftarrow load_{j^*} + k(j^*)$
9. For $1 \leq s \leq n_{j^*}(k^*)$ do
10. $\psi(i_s, j^*) \leftarrow \psi(i_s, j^*) + r(i_s)$
11. $r(i_s) = 0$
12. $\psi(i_{n_{j^*}(k^*)+1}, j^*) = l_{j^*}(k^*)$
13. $r(i_{n_{j^*}(k^*)+1}) \leftarrow r(i+1) - l_{j^*}(k(j^*))$

The average running time of this algorithm can be improved by taking advantage of the fact that in our case $f_j(y)$ is a step function. Specifically, it can be shown that in order to compute the minimum in step 5, one needs to do the computation only for values of k such that $load_j + k$ is a jump point of $f_j(y)$, or $k = R_j(n)$ for some n.

Based on the implementation in [18], Algorithm 2 has worst case running time
$O(|V|^3 g_{\max}^2 \log(|V|^2 g_{\max}))$.

Letting $|S|$ be the maximum number of server types in any of the server locations, taking into account that the maximum load on any facility is $|V| g_{\max}$ and that we may need to compute at most

$|V|$ functions $f_j(y)$, we conclude that the worst case computation time of the complete algorithm is

$$O\left(|V|\, D\,(|V|\log|V| + |E|\log|V|) + |S|\,|V|^2\, g_{\max} + |V|^3\, g_{\max}^2 \log(|V|^2\, g_{\max})\right).$$

The algorithm presented above works well in practice as will be seen in Section 5. However, it is pseudopolynomial and it is theoretically important to know whether there exists a polynomial time algorithm that can provide a guaranteed approximation factor with respect to the optimal. In the next section we will show that this can be done based on the algorithm presented above.

4 Polynomial Algorithm

In this section, by generalizing the approach in [18] we provide a polynomial time approximation algorithm for arbitrary integer node loads and non-decreasing subadditive functions that are not necessarily computable exactly in polynomial time. Note that a concave function is also subadditive and hence our results carry over to concave functions. However, as will be seen, for concave functions the approximation constants can be made smaller.

The approach we follow is to provide polynomial time approximation algorithms for each of Problems 1, 2 and 3. By combining these algorithms, we get a polynomial time algorithm for the problem at hand with guaranteed performance factor compared to the optimal.

In the previous section the costs c_{ij} and the functions $f_j(y)$ were computed exactly using pseudopolynomial algorithms for Problems 1 and 2 respectively. The use of polynomial time approximation algorithms for these problems provides only approximate values for c_{ij} and $f_j(y)$. That is, we can only ensure that for any $\varepsilon > 0$, we provide in polynomial time values $\overline{c}_{ij}$ and $\overline{f}_j(y)$ (for fixed y) such that $c_{ij} \leq \overline{c}_{ij} \leq (1+\varepsilon)c_{ij}$ and $f_j(y) \leq \overline{f}_j(y) \leq (1+\varepsilon)f_j(y)$, $y \geq 0$. Replacing c_{ij} and $f_j(y)$ with $\overline{c}_{ij}$ and $\overline{f}_j(y)$ in Problem 3 and providing an α-approximate solution for the resulting instance, provides also an $(1+\varepsilon)\alpha$- approximate solution for the original problem. This important observation was used in [22] and we present it here in the next lemma.

Lemma 1 *Consider the problems*

$$\min_{\mathbf{x}\in A} g(\mathbf{x}),\ A \subseteq R^n, \tag{8}$$

$$\min_{\mathbf{x}\in A} \overline{g}(\mathbf{x}),\ A \subseteq R^n, \tag{9}$$

where $g(x) \geq 0$. If for $x \in A$, $g(x) \leq \overline{g}(x) \leq \beta g(x)$, then an $\alpha-$approximate solution for problem (9), $\alpha \geq 1$, is a $\beta\alpha$-approximate solution for (8).

Using Lemma 1 we can proceed as follows.

- Compute in polynomial time approximate values $\overline{c}_{ij}$,
- Compute in polynomial time approximate values $\overline{f}(y)$ (for a given $y \geq 0$),
- Provide an approximation algorithm for Problem 3, based on Algorithm 2, using the approximate values $\overline{c}_{ij}$, $\overline{f}(y)$.

Difficulties arise in the approach outlined above for the following reasons. First, to compute the minimum in Step 5 of Algorithm 2, $\overline{f}(y)$ must be computed for all values of y in the worst case, and the number of these computations is bounded by $\sum_{i \in V} g_i$, i.e., it is not polynomial in the input size, even if $\overline{f}(y)$ is computable at unit cost. Second, the amount of load assigned to a node in H at each iteration of the while loop at step 9 can be 1 in the worst case and hence the number of iterations of the while loop may be again $\sum_{i \in V} g_i$ in the worst case. Third, while $f(y)$ is subadditive, it cannot be guaranteed that $\overline{f}(y)$ is subadditive as well, and hence the approximation constants cannot be guaranteed a priori. Hence, the straightforward application of Algorithm 2 will result in pseudopolynomial worst case running time and will not provide us with guaranteed performance bounds. As will be seen, however, we can modify the approach so that the resulting algorithm runs in polynomial time at the cost of a small increase in the approximation factor.

4.1 Polynomial Algorithms for Problem 1 and 2

A fully polynomial time approximation algorithm for the problem of finding the minimum constrained path from a source to a given destination was developed by Hassin [23]. An improvement of this algorithm was presented in [24]. The latter algorithm can be used as a basis for the development of a fully polynomial time approximation algorithm for Problem 1. We skip the details here and mention that the adaptation consists in modifying Algorithm SPPP in [24] using an approach similar to the one followed for Algorithm 1. The resulting algorithm runs in $t = O(|E||V|(\log\log|V| + 1/\varepsilon))$ for each route.

A fully polynomial time approximation algorithm for Problem 2 can be developed by paralleling the approach for the UKP [11, Section 8.5]. The resulting algorithm has a worst case running time of $T = O(\frac{1}{\varepsilon^2}|S|\log|S|)$, where $|S|$ is the number of server types.

4.2 Polynomial Algorithm for Problem 3

We now address the main problem of Section 4, i.e., the development of a polynomial algorithm for Problem 3, using the approximate costs $\overline{c}_{ij}$ and $\overline{f}_j(y)$. As explained above, we intend to use Algorithm 2 as the basis for the development. Assume for the moment that c_{ij} and $f_j(y)$ are computable exactly. As was mentioned above the fact that the node loads are general nonnegative integers in our case, renders the algorithm pseudopolynomial even under this assumption. However, if the functions $f_j(y)$ are concave, then the algorithm becomes polynomial. This is due to the fact that for concave functions Algorithm 2 can assign all the load of each node to a single server. This is shown in the next lemma. Recall that a function $f(y)$ defined for integer y is called concave if for all y in its domain of definition it holds, $f(y+1) - f(y) \leq f(y) - f(y-1)$.

Lemma 2 *If the functions $f_i(y)$ are concave and Algorithm 2 is applied, then the load of each node in V can be assigned to a single server.*

Lemma 2 implies that when c_{ij} and $f_j(y)$ are computable in polynomial time and $f_j(y)$ are concave, the algorithm runs in polynomial time. The running time is $O(|V|^3 \log |V| + |V|^2 (t+T))$, where t and T are the running times for Problems 1 and 2 as described in the previous section, and the approximation factor is $\log(|V| g_{\max})$, where $g_{\max} = \max_{i \in V} \{g_i\}$.

We now return to the problem at hand. In our case, $f_j(y)$ is subadditive rather than concave and is not computable exactly in polynomial time. Hence the results above cannot be applied directly to obtain a polynomial time algorithm. The approach we follow is to construct in polynomial time a concave function $\widetilde{f}_j(y)$, such that for any y in its domain of definition $\widetilde{f}_j(y)$ is computed in polynomial time and,

$$f_j(y) \leq \widetilde{f}_j(y) \leq \alpha f_j(y). \tag{10}$$

Then, by applying Lemmas 1 and 2 we get a polynomial time algorithm. To proceed we need some definitions.

Consider a nonnegative function $\phi : \{0, 1, \ldots, W\} \rightarrow \mathbb{Q}^+$ ($\mathbb{Q}^+$ is the set of nonnegative rationals) and let A be the convex hull of the set of points $S = \{(y, \phi(y)),\ y = 0, 1, \ldots, W\} \cup \{(0,0), (W,0)\}$. Recall that the convex hull of a set of points S is the smallest convex set that includes these points. In two dimensions it is a convex polygon. The vertices of the polygon correspond to a subset of S, of the form $S' = \{(y_k, \phi(y_k)), k = 1, .., K\} \cup \{(0,0), (W,0)\}$ where $y_k \in \{0, 1, \ldots, W\},\ y_1 = 0,\ y_K = W$, and $y_k < y_{k+1}$ for all k, $1 \leq k \leq K-1$.

Consider the piecewise linear function $\phi_2(y)$ with break points the set S', i.e., for $y_k \leq y < y_{k+1}$, $\phi_2(y)$ is defined as,

$$\phi_2(y) = \phi(y_k) + \frac{\phi(y_{k+1}) - \phi(y_k)}{y_{k+1} - y_k}(y - y_k), \tag{11}$$

The function $\phi_2(y)$ is concave. We call $\phi_2(y)$, the "upper hull" of $\phi(y)$. If $\phi(y)$ is non-decreasing, the same holds for $\phi_2(y)$. By construction it holds for all $y \in \{0, 1, ..., W\}$, $\phi(y) \leq \phi_2(y)$.

If the function $\phi(y)$ is subadditive and non-decreasing, then it also holds that its upper hull is smaller than twice the function value.

Lemma 3 *If a function $\phi : \{0, 1, \ldots, W\} \to \mathbb{Q}$ is subadditive and non-decreasing, then it holds for its upper hull $\phi_2(y)$, $\phi_2(y) \leq 2\phi(y)$.*

Consider now the subadditive function $f(y)$ of interest in our case (we drop the index j for simplicity). As a consequence of the approximate solution to Problem 2, for a given $\varepsilon > 0$ and a given $y \in \{0, 1, ..., W\}$, $W = |V| g_{\max}$, we can construct in polynomial time a non-decreasing function such that

$$f(y) \leq \overline{f}(y) \leq (1 + \varepsilon) f(y). \tag{12}$$

Let $\overline{f}_2(y)$ be the upper hull of $\overline{f}(y)$. By (12), $\overline{f}_2(y)$ is smaller than or equal to the upper hull of $(1 + \varepsilon)f(y)$, which in turn by Lemma 3 is smaller than $2(1 + \varepsilon)f(y)$ (notice that $(1 + \varepsilon)f(y)$ is subadditive). Hence we will have

$$f(y) \leq \overline{f}_2(y) \leq 2(1 + \varepsilon) f(y). \tag{13}$$

Since $\overline{f}_2(y)$ is concave, if we replace c_{ij} with $\overline{c}_{ij}$ and $f(y)$ with $\overline{f}_2(y)$, we can provide an approximate solution to Problem 2 with approximation factor $\log(|V| g_{\max})$. From (13) and Lemma 1 we will then have a solution to our original problem with approximation factor $2(1 + \varepsilon) \log |V g_{\max}|$.

The problem that remains to be solved is the construction of the upper hull of $\overline{f}(y)$ in polynomial time. There are at most $W' = W + 2$ points in the set $\{(y, \overline{f}(y)),\ y = 0, 1, \ldots, W\} \cup \{(0, 0), (W, 0)\}$ and the upper hull of the points in this set can be constructed (i.e., its break points can be determined) in time $W' \log W'$ [25]. However, in our case $W = |V| g_{\max}$ and hence the straightforward construction of the upper hull requires pseudopolynomial construction time.

To address the latter problem, we construct first a non-decreasing step function $\widehat{f}_1(y)$ with polynomial number of jump points (y is a jump point of $\widehat{f}_1(y)$ if $\widehat{f}_1(y - 1) \neq f_1(y)$) that is a good

approximation to $f(y)$, and then we construct the upper hull of $\widehat{f}_1(y)$. Since $\widehat{f}_1(y)$ has polynomial number of jump points its upper hull will also have polynomial number of break points and can be constructed in polynomial time.

We have by definition $\overline{f}(0) = 0,\ \overline{f}(1) = f(1) > 0$. Consider the sequence of integers $\widehat{f}_0 = 0,\ \widehat{f}_k$, $k = 1, ..., K$, generated by Algorithm 3.

Algorithm 3

Input: Algorithm for computing $\overline{f}(y)$, $\epsilon_1 > 0$.
Output: The sequence, $\widehat{f}_k$, $k = 0, 1, ..., K$.
1. $\widehat{f}_0 = 0,\ \widehat{f}_1 = \overline{f}(1)$, $y_1 = 1$, $k = 2$,
2. $\widehat{f}_k = (1 + \epsilon_1)\overline{f}(y_{k-1})$
3. If $\widehat{f}_k > \overline{f}(W)$, set $y_k = W$, $K = k$ and stop. Else,
4. Determine y_k such that $\overline{f}(y_k - 1) \leq \widehat{f}_k \leq \overline{f}(y_k)$,i.e.,
5. $k = k + 1$, go to step 2.

The sequence $\widehat{f}_k$, $k = 0, 1, ..., K$ can be used to construct a step function that is a good approximation to $\overline{f}(y)$. This is shown in the next lemma.

Lemma 4 *a) In Algorithm 3,* $K = O\left(\frac{1}{\epsilon_1} \log \frac{\overline{f}(W)}{\overline{f}_1}\right)$. *The worst case running time of Algorithm 3 is* $O\left(T \log(W) \frac{1}{\epsilon_1} \log \frac{\overline{f}(W)}{\overline{f}_1}\right)$, *where* T *is the worst case time (over all* y, $1 \leq y \leq W$*) needed to compute* $\overline{f}(y)$. *b) Consider the step function defined as follows: if* $y_k \leq y < y_{k+1}$ *for some* k, $1 \leq k \leq K - 1$, *then* $\widehat{f}(y) = \widehat{f}_k$, *and* $\widehat{f}(W) = \widehat{f}_K$. *It holds,*

$$\widehat{f}(y) \leq \overline{f}(y) \leq (1 + \epsilon_1)\widehat{f}(y) \tag{14}$$

Based on (14), we can use $\widetilde{f}(y) = (1 + \epsilon_1)\widehat{f}(y)$ as a function to approximate $f(y)$.

Lemma 5 *Let* $\epsilon_0 > 0$, $\epsilon_1 > 0$ *be given. Let* $f(y)$ *be the optimal solution to Problem 2 and assume that we compute for a given* y *the approximate function* $\overline{f}(y)$ *so that*

$$f(y) \leq \overline{f}(y) \leq (1 + \epsilon_0)\, f(y). \tag{15}$$

a) For the purposes of computing the step function $\widehat{f}(y)$ *satisfying (14),* $\overline{f}(y)$ *may be assumed non-decreasing. b) It holds for* $\widetilde{f}(y) = (1 + \epsilon_1)\widehat{f}(y)$

$$f(y) \leq \widetilde{f}(y) \leq (1 + (\epsilon_1 + \epsilon_0 + \epsilon_1\epsilon_0))\, f(y), \tag{16}$$

c) The number of jump points of $\widehat{f}(y)$, hence of $\widetilde{f}(y)$, is $O\left(\frac{1}{\epsilon_1}\log\left(|V|\,g_{\max}\right)\right)$ and the running time of Algorithm 3 is $O\left(\frac{1}{\epsilon_1}T(\log\left(|V|\,g_{\max}\right))^2\right)$, where T is the worst case time (over all y, $1 \le y \le W$) needed to compute $\overline{f}(y)$.

From the discussion above we have polynomial time Algorithm 4 for computing the server locations.

Algorithm 4 Polynomial Time Algorithm For Calculating Server Locations

Input: Polynomial Algorithm for Problem 1, Algorithms 2 and 3, $\epsilon > 0$.
Output: Array of server locations.

1. For $i \in V$, $j \in H$, compute $\overline{c}_{ij}$ so that $c_{ij} \le \overline{c}_{ij} \le (1+\epsilon)c_{ij}$, $i \in V$, $j \in H$.
2. For $j \in H$, construct the step functions $\widehat{f}_j(y)$,from $\overline{f}_j(y)$ according to Algorithm 3, using as subroutine the algorithm for computing, for a given $y > 0$, $\overline{f}_j(y)$ such that $f_j(y) \le \overline{f}_j(y) \le (1+\epsilon)f_j(y)$.
3. Construct the upper hull of $\widetilde{f}_j(y) = (1+\epsilon)\widehat{f}_j(y)$. Let $\phi_j(y)$ be this upper hull.
4. Use Algorithm 2 to solve Problem 3, where c_{ij} is replaced by $\overline{c}_{ij}$ and $f_j(y)$ is replaced by $\phi_j(y)$.

In the algorithm, for simplicity, we pick a single ϵ for all the approximations. If needed, a separate ϵ can be used for each of the approximations.

The worst case running times of each step are:

1. $O(|V|^2 t) = O(|E||V|^3(\log\log|V| + 1/\epsilon))$
2. $O(|V|\,\frac{1}{\epsilon}T(\log\left(|V|\,g_{\max}\right))^2) = O(\frac{1}{\epsilon^3}|V|\,|S|\log|S|\,(\log\left(|V|\,g_{\max}\right))^2)$
3. $O(K\log K) = O(\frac{1}{\epsilon^2}\log\left(|V|\,g_{\max}\right)\log\log\left(|V|\,g_{\max}\right))$
4. $O(|V|^3\log|V|)$

The resulting algorithm has a guaranteed performance ratio of $2(1+\epsilon)^2\log\left(|V|\,g_{\max}\right)$ and its worst case running time is dominated by steps 1 and 2

$$O(|E||V|^3(\log\log|V| + 1/\epsilon) + \frac{1}{\epsilon^3}|V|\,|S|\log|S|\,(\log\left(|V|\,g_{\max}\right))^2).$$

5 Numerical Results

In this section we evaluate the total cost of using servers and utilizing the link bandwidth on random network topologies using two different routing methods.

To generate random networks a number $|V|$ of nodes and a number $|E| = \gamma\,|N|$ of edges, $\gamma > 1$ is chosen. We use the graph generator *random_graph( )* from the LEDA package [26]. A random edge is generated by selecting a random element from a candidate set C defined as follows. C is initialized

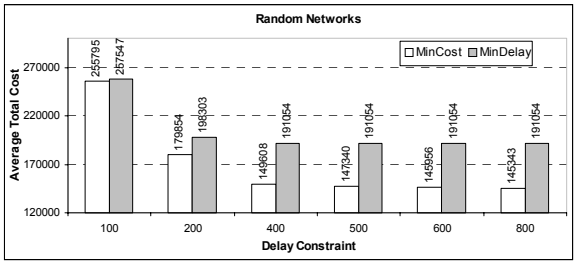

Figure 1: Average Total Cost for different Delay Constraints

to the set of all $|V|(|V|-1)$ pairs of distinct nodes. An edge (i,j) in C is selected randomly and removed from C. The process is repeated until $|E|$ edges are selected.

Ten different random network topologies are generated with $|V| = 100$ nodes and $|E| = 500$ edges. For each network the delay of a link is picked randomly with uniform distribution among the integers $[1, 100]$ and the cost is generated in such a manner that it is correlated to its delay. Thus, for each link l a parameter b_l is generated randomly among the integers $[1, 5]$. The cost of link l is then $b_l(101 - d_l)$. Hence the link cost is a decreasing function of its delay. We run the algorithm for 6 different delay constraints $D = \{100, 200, 400, 500, 600, 800\}$. We assume that the the same server types can be placed in each of the locations. For our simulations we use 4 different server types with capacities and costs equal to $\{(100, 3000)(150, 3500)(250, 4000)(350, 5000)\}$ respectively. We set the factors $\alpha = 0.1$ and $\beta = 0.2$ and assume the load in requests per unit of time originated by each node is randomly chosen among the integers $[1, 100]$. We also assume $H = V$, i.e., servers may be placed in any of the nodes. We run two sets of experiments which differ in the manner the request round-trip routes are selected. Specifically we consider the following algorithms:

MinDelay: In this algorithm the minimum delay round-trip routes are selected without considering the route cost. A route thus selected is rejected if its delay is larger than the specified constraint. This manner of selecting routes has been employed in [8].

MinCost:. This is the algorithm proposed in the current work. That is, the round-trip routes are selected so that they are of minimum cost provided that they satisfy the delay constraint.

For the simulations we used the pseudopolynomial algorithm since it works sufficiently well for the selected instances and its implementation is considerably simpler than the polynomial algorithm.

In Figure 1 we present the average total cost of using the servers and utilizing the link bandwidth. We make the following observations. The cost for both algorithms decreases as the delay constraint increases and levels off after certain value of the delay constraint. This is explained as follows.

For smaller delay constraints, several locations are becoming unreachable by the nodes. Hence the options of directing the node load to the various locations are reduced and as result the overall cost of the solution increases. The leveling-off of the computed cost is due to the fact that as the delay constraints become looser, all the minimum cost round-trip routes are selected by MinCost algorithm and all the minimum delay round-tip routes by the MinDelay algorithm. We also observe in Figure 1 that the total cost of MinCost algorithm is always smaller than MinDelay, as expected, and the significance is becoming more pronounced for larger delays. This behavior is again due to the manner in which routes are selected by the two algorithms for a given delay constraint. For strict delay constraints, both algorithms choose mainly the permissible minimum delay round-trip routes and hence they have similar performance. For looser constraints, the fact that MinCost picks the minimum cost round-trip routes that satisfy the delay constraint instead of simply the minimum delay route (as MinDelay does) allows it to reduce the routing cost.

6 Conclusions

In this paper we presented a pseudopolynomial approximation algorithm and a polynomial time algorithm for the NP-hard problem of replicated server placement with QoS constraints. The pseudopolynomial algorithm works well in several practical instances and is simpler than the polynomial time algorithm. The polynomial time algorithm is significant from a theoretical point of view and can be useful to employ if the problem instance renders the pseudopolynomial time algorithm very slow.

In this work we did not consider link capacities. It is an interesting open problem to incorporate the latter constraint into the problem. Another problem of interest is to consider the case where not all the database is replicated to each of the servers.

References

[1] R. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 568–582, October 2000.

[2] B. Li, M. Golin, G. Italiano, and X. Deng, "On the optimal placement of web proxies in the internet," in *IEEE INFOCOM*, 1999.

[3] X. Jia, D. Li, X. Hu, and D. Du, "Optimal placement of web proxies for replicated web servers in the internet." *The Computer Journal*, vol. 44, no. 5, pp. 329–339, 2001.

[4] X. Jia, X. H. D. Li, W. Wu, and D. Du, "Placement of web-server proxies with consideration of read and update operations on the internet," *The Computer Journal*, vol. 46, no. 4, 2003.

[5] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *IEEE INFOCOM*, April 2001, pp. 1587–1596.

[6] Y. Chen, R. Katz, and J. Kubiatowicz, "Dynamic replica placement for scalable content delivery," in *First International Workshop on Peer-to-Peer Systems*, 2002, pp. 306–318.

[7] S. Jamin, C. Jiu, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror palcement on the internet," in *IEEE INFOCOM*, April 2001, pp. 31–40.

[8] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," in *Workshop on Internet Server Performance, Madison, Wisconsin*, 1998.

[9] X. Tang and J. Xu, "On replica placement for QoS-aware content distribution," in *IEEE INFOCOM*, 2004.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory on NP-Completeness*. W. H. FREEMAN AND COMPANY, 1979.

[11] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Spinger-Verlag, 2004.

[12] F. Chudak and D. Williamson, "Improved approximation algorithms for capacitated facility location problems," in *Integer Programming and Combinatorial Optimization*, 1999.

[13] M. Korupolu, C. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *Journal of Algorithms*, vol. 37, pp. 146–188, 2000.

[14] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," in *In Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001, pp. 21–29.

[15] K. Jain and V. Vazirani, "Approximation algorithms for the metric facility location and k-median problems using the primal-dual schema and the lagrangian relaxation," *Journal of the ACM*, vol. 48, pp. 274–296, 2001.

[16] M. Pal, I. Tardos, and T. Wexler, "Facility location with nonuniform hard capacities," in *IEEE Symposium on Foundations of Computer Science*, October 14-17, 2001, p. 329.

[17] F. Chudak and D. Shmoys, "Improved approximation algorithms for the uncapacitated facility location problem," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 1–25, 2003.

[18] M. T. Hajiaghavi, M. Mahdian, and V. S. Mirrokni, "The facility location problem with general cost functions," *Networks*, vol. 42, no. 1, pp. 42–47, 2003.

[19] S. Siachalou and L. Georgiadis, "Efficient QoS routing," *Computer Networks*, vol. 43, pp. 351–367, 2003.

[20] P. V. Mieghem, H. de Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Computer Networks*, vol. 37, pp. 407–423, 2001.

[21] R. Andonov and S. Rajopadhye, "A sparse knapsack algo-tech-cuit and its synthesis," in *International Conference on Application Specific Array Processors ASPA '94*. IEEE, 1994.

[22] M. Mahdian, E. Markakis, A. Saberi, and V. Varizani, "Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP," *Journal of the ACM*, vol. 50, no. 6, pp. 795–824, 2003.

[23] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, 1992.

[24] D. H. Lorenz and D. Raz, "A simple and efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, vol. 28, no. 5, pp. 213–221, 2001.

[25] M. de Berg, O. Schwarzkoph, M. V. Kreveld, and M. Overmars, *Computional Geometry: Algorithms and Applications*. Springer-Verlag, 2000.

[26] K.Mehlhorn and S. Naher, "Leda: A platform for combinatorial and geometric computing," Cambridge University Press, 2000.