Replicated Server Placement with QoS Constraints

Georgios Rodolakis, Stavroula Siachalou and Leonidas Georgiadis

Abstract

The network planning 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 be served by more than one location as long as the round-trip delay of data requests 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.

Index Terms

Algorithm design and analysis, Constrained optimization, Applications, Distributed file systems, Client/server.

I. Introduction

Communications networks are used widely for distributing data and all kinds of services. In networks such as the Internet, the world's largest computer network, providers strive to satisfy user QoS requirements in a cost effective manner. Within this framework, we concentrate in this paper on the problem of placing replicated data servers in various parts of the network so that the overall cost (i.e., cost of opening the servers and transferring the data) is minimized, while satisfying user requirements that concern constraints on the round-trip delay of data requests. We formulate this network planning problem in a general manner, so that the algorithms we

This paper was presented in part at the Third International Workshop on QoS in Multiservice IP Networks Catania, Italy – February 2-4, 2005.

G. Rodolakis is with Hipercom Project, INRIA Rocquencourt, 78153 Le Chesnay Cedex, France. E-mail: Georges.Rodolakis@inria.fr.

S. Siachalou and L. Georgiadis are with Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece. E-mails: ssiachal@auth.gr, leonid@auth.gr.

provide can be useful in many different contexts and applications, for instance in the placement of web proxies, in distributed file systems, in distributed databases etc.

The problem of replicated server placement 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 replicas of the server. 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 on 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]-[7] tackled the replica placement problem from another angle: minimizing the number of replicas while meeting clients' latency constraints and servers' capacity constraints by self organizing these replicas into a dissemination tree with small delay and bandwidth consumption for update dissemination. In [8] 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. [9] presented a number of selection algorithms to access replicated Web servers. 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 [10] 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. The construction of a distribution tree for a given set of replicas with the objective of minimizing the total communication cost of consistency management has been studied in [11]. Tang et. al. [12] presented a theoretical study on geographical replication of dynamic Web contents with the objective of minimizing the consistency management costs in terms of update transfers and

object reconstruction. Cohen and Shenker [13] defined replication strategies in decentralized unstructured systems. They assumed each node had capacity ρ, which was the number of copies the node could hold and R was the total capacity of the system. Their replication strategy was a mapping from the query rate distribution to the fraction of the total system capacity allotted to each item.

In this paper we approach the problem of replicated server placement with QoS constraints from a system administrator's perspective. Our contributions are the following. In contrast to most of the papers addressing similar problems, instead of heuristics, we provide a solution with provable performance guarantees for any possible network topology and client distribution. Also, rather than attempting to optimize metrics related to communication delays, we impose upper bounds on the round trip delays of data requests and attempt to minimize the operating cost of server placement, while respecting the delay bounds. Moreover, in the optimization we take into account the multiplicity of server types that may be available at a site. As will be seen, 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.

The rest of the paper is organized as follows. In Section II, we formulate the problem in details and we decompose it into three subproblems that can be solved independently. In Section III, we present a pseudopolynomial time approximation algorithm. In Section IV, we provide 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 V. Conclusions are presented in Section VI.

II. 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 second 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. Since we consider link costs, this is a reasonable assumption, because any capacity that is required can be provided by the ISP, which in case of bandwidth saturation can either add more capacity, or charge an extra cost for the use of that link. Furthermore, in an environment where the server requests on a given link are a small portion of the total amount of information that can be supported by the network this assumption is of no consequence. However, in the case where bottleneck links could emerge we need to take special care to apply excessive costs to these links in order to avoid the concentration of too much traffic. The general problem where link capacities are also included, is a subject of further research.

A. 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

d_{ij}, e_{ij} delay and cost of link (i,j)	$F \subseteq H$ set of chosen locations
x requests/sec	$G_j \subseteq S_j$, set of server types opened at $j \in F$
g_i load originated by node i	n_j^s number of server types $s \in G_j$ at $j \in F$
$r_{ij} = (p_{r_{ij}}, q_{r_{ij}})$ round-trip route	R_{ij} set of round-trip routes between $i \in V$ and $j \in F$
D delay bound of r_{ij}	$R = \cup_{i \in V} \cup_{j \in F} R_{ij}$
$H \subseteq V$, set of possible server locations	$x_{r_{ij}}$ requests/sec at route r_{ij}
S_j set of possible server types at $j \in H$	f_j^s cost of server s at $j \in H$
U_j^s requests/sec processed by server s at $j \in H$	$f_j(y)$ minimum cost server cost at location j for load y

Fig. 1. Notation Table

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. For the reader's convenience, we summarize the basic notations used throughout the paper in Figure 1.

B. Problem Decomposition

In this section we decompose the problem defined in Section II-A 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 $C_r = \alpha \sum_{l \in p} e_l + \beta \sum_{l \in q} e_l$. Consider a feasible solution, π, for the optimization problem.

We can rewrite the second term in (4) as follows.

$$\sum_{l \in E} e_l \left(\alpha \sum_{\substack{r=(p,q) \in R \\ l \in p}} x_r + \beta \sum_{\substack{r=(p,q) \in R \\ l \in q}} x_r \right) = \sum_{i \in V} \sum_{j \in F} \sum_{r \in R_{ij}} C_r x_r. \tag{5}$$

Let r_{ij}^* be a minimum-cost round-trip route between node i and server location j, satisfying the round-trip delay D. Consider the feasible solution that uses the same server locations, the same servers at each location, but assigns all the request load from node i to server location j on the round-trip route r_{ij}^*, i.e., it assigns on r_{ij}^* the load $x_{ij} = \sum_{r \in R_{ij}} x_r$.

It follows from (4) and (5) that the new solution has cost smaller than or equal to the cost of solution π. Hence it suffices to restrict attention to solutions that assign all the load from node i to server location j, on the round-trip route r_{ij}^*. In this case, setting $c_{ij} = C_{r_{ij}^*}$, the 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 π. 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

$$f_j(y_1) + f_j(y_2) \geq f_j(y_1 + y_2) \text{ for all } y_1 \geq 0,\ y_2 \geq 0. \tag{6}$$

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 $y_1 + y_2$, (6) 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 $n_j^s(y)$ of each type s so that the load y 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

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

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

A graphical representation of the problem is depicted in Figure 2.

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 [14]. Also, the associated decision problem to Problem 2 amounts to the Unbounded Knapsack Problem which is NP-hard [15]. However for both problems

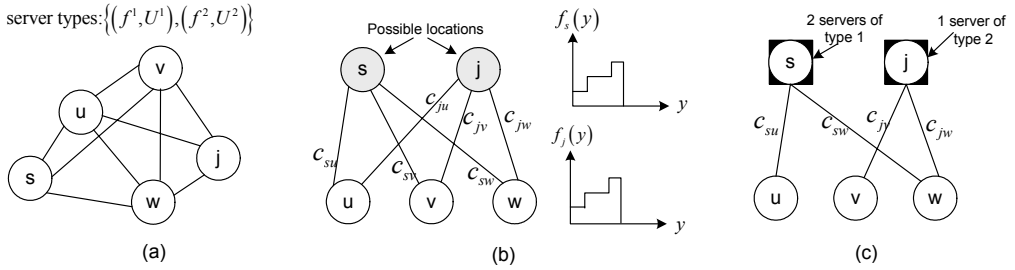

Fig. 2. a) A graph with five clients, $V = \{s, u, v, w, i\}$, two possible locations $H = \{s, j\}$ and two types of servers for each location. b) The modified graph where each link represents the round-trip minimum cost routes, satisfying the delay constraint between a client and a possible location. $f_s(y)$ and $f_j(y)$ are the cost functions at the possible locations. c) The resulting graph. Servers have been placed at the appropriate locations.

pseudopolynomial algorithms exist (see Section III) and, as will be discussed in Section IV, 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} ([16], [17], [18], [19], [20], [21], [22]). 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. Moreover, $f_i(y)$ is assumed computable at unit cost while in our case $f_i(y)$ cannot be computed in polynomial time (unless $P = NP$).

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 [22] and uses the fact that $f_j(y)$ is a subadditive step function. In Section IV we modify the algorithm in order to obtain a polynomial time algorithm with approximation factor close to the best possible (unless NP$\subseteq$DTIME($n^{O(\log\log n)}$)).

III. Pseudopolynomial Algorithm

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

A. Pseudopolynomial Algorithm for Problem 1

Let $F_{ij}(d)$ be the value of the minimum cost (forward) path from node i to j with delay at most d, and $B_{ij}(d)$ the value of the minimum cost (backward) path from node j to i with delay at most d. Here, for the computation of forward and backward paths the link costs are taken as αe_{ij}, βe_{ij} respectively. Then it can be easily seen that,

$$c_{ij} = \min_{0 \leq d \leq D} \{F_{ij}(d) + B_{ij}(D-d)\}. \tag{7}$$

Based on (7), c_{ij} can be determined provided $F_{ij}(d)$ and $B_{ij}(d)$ are known. There are fully polynomial time, generally complex, algorithms for computing these quantities. In this section we will concentrate on efficient pseudopolynomial algorithms that work well in practice [23], [24]. We provide the discussion for $F_{ij}(d)$, since the same holds for $B_{ij}(d)$. The algorithms in [23], [24] 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 K_{ij}^f be the number of jump points of $F_{ij}(d)$ and K_{ij}^b be the number of jump points of $B_{ij}(d)$. 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 $d_{ij}^f(k-1) < d_{ij}^f(k) \leq D,\ k = 2, ..., K_{ij}^f$. Similarly, let $d_{ij}^b(k)$, $1 \leq k \leq K_{ij}^b,\ i \in V$ 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 [23]. The "for" loop in step 6 implements the minimization required by (7), taking into account that $F_{ij}(d)$ and $B_{ij}(d)$ are step functions.

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

Input: Graph G with link costs and delays, round-trip delay bound D.
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 $F_{ij}(d)$, $d_{ij}^f(k)$, $1 \leq k \leq K_{ij}^f$, $i \in V$
3) Compute jump points of $B_{ij}(d)$, $d_{ij}^b(k)$, $1 \leq k \leq K_{ij}^b,\ i \in V$
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) $c_{ij} \leftarrow \min\left\{c_{ij}, F_{ij}(d_{ij}^f(k)) + B_{ij}(d_{ij}^b)\right\}$

We now discuss the complexity of Algorithm 1. For the purposes of complexity analysis, in the rest of the paper we will assume that in the worst case $H = V$. Using the algorithm and the analysis presented in [23] it can be proved that the worst case running time for the computation of the jump points for all nodes in H 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.

B. 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 $\sum_{s\in G} n^s(y)U^s \geq y$ and $f(y) = \sum_{s\in G} n^s(y)f^s$ is minimal.

Problem 2 is similar to the Unbounded Knapsack Problem (UKP) [15]. 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 f. The entries of the table $A(f,i)$ can be computed in order of increasing i and f using the dynamic programming equation

$$A(f,i+1) = \min\{A(f,i), U^{i+1} + A(f-f^{i+1}, i+1)\}, \tag{8}$$

with $A(f,0) = 0$ for all f, $A(f,i) = -\infty$ if $f < 0$, and $A(0,i) = 0$ for all $0 \leq i \leq K$. 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 (8) and using an approach similar to [25], 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 $f(x)$, for integer $x \leq y$, can be determined. The running time of this approach can be bounded by $O(|S|\, y)$, where $|S|$ is the number of server types.

C. Pseudopolynomial Algorithm for Problem 3

In [22] a polynomial time algorithm is provided for Problem 3 for 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 while the functions $f_j(y)$ are subadditive and can

be computed exactly only in pseudopolynomial time. As observed in [26] 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 costs to nodes in H as node i has. 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 [22] carries over without modification if $f_j(y)$ are subadditive rather than concave functions. Hence the approximability factor in our case becomes $\ln(|V|\, g_{\max})$.

To our knowledge, the algorithm in [22] is the only one proposed in the literature, 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 [22] 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.

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 of the remaining loads of the nodes that will place their load on the selected server location is taking place.

The average running time of this algorithm can be improved by taking advantage of the fact that in this case $f_j(y)$ is a step function. Specifically, in the Appendix A it is 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.

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, routes and 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 $j^* = \arg\min_{j \in H} \{t(j)\}$, $k^* = k(j^*)$
8) Set $load_{j^*} \leftarrow load_{j^*} + k^*$
9) For $1 \le s \le n_{j^*}(k^*)$ 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) $r(i_{n_{j^*}(k^*)+1}) \leftarrow r(i+1) - l_{j^*}(k^*)$

In Appendix B, we show that with the use of appropriate data structures and assuming unit cost for computing $f_j(y)$, the running time of Algorithm 2 is

$$O(|V|^3 g_{\max}^2). \tag{9}$$

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

$$O\left(|V| D (|V| \log |V| + |E| \log |V|) + |S| |V|^2 g_{\max} + |V|^3 g_{\max}^2\right). \tag{10}$$

The term $|V| D (|V| \log |V| + |E| \log |V|)$ corresponds to the computation of c_{ij}. The term $|S| |V|^2 g_{\max}$ corresponds to the cost of computing $f_j(|V| g_{\max})$, $j \in H$. Note that as mentioned in Section III-B, for each $j \in H$, computing $f_j(|V| g_{\max})$ also computes all the jump points of $f_j(y)$, $y \le |V| g_{\max}$. As a result, when implementing Algorithm 2, $f_j(y)$ can be computed at unit cost. Hence the third term in (10) represents the worst case computation time for running Algorithm 2. As mentioned in Section III-C the approximability factor of this algorithm is $\ln(|V| g_{\max})$.

As will be seen in Section V the proposed algorithm works well in practice. However, since

(10) involves the input parameters D and $g_{\max}$, the proposed algorithm is pseudopolynomial. 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.

IV. Polynomial Algorithm

In this section, by generalizing the approach in [22] 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 a-approximate solution for the resulting instance, provides also an $(1+\varepsilon)a$- approximate solution for the original problem. This important observation was used in [26] and we present it here in the next lemma.

Lemma 1: Consider the problems

$$\min_{\mathbf{x}\in A} g(\mathbf{x}),\ A \subseteq R^n, \tag{11}$$

$$\min_{\mathbf{x}\in A} \overline{g}(\mathbf{x}),\ A \subseteq R^n, \tag{12}$$

where $g(\mathbf{x}) \geq 0$. If for $\mathbf{x} \in A$, $g(\mathbf{x}) \leq \overline{g}(\mathbf{x}) \leq bg(\mathbf{x})$, then an $a-$approximate solution for problem (12), $a \geq 1$, is a ba-approximate solution for (11).

Proof: The proof is given in Appendix C. ■

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 $|V|\, g_{\max}$, 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 "for" loop at step 9 can be 1 in the worst case and hence the number of iterations of the while loop may be again $|V|\, g_{\max}$ 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 factor with respect to the optimal 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.

A. 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 [27]. An improvement of this algorithm was presented in [28]. The approach in [28] consists in defining a test procedure, which is used iteratively to find upper and lower bounds for the restricted shortest path.

The latter algorithm can be modified in order to develop a fully polynomial time approximation algorithm for Problem 1, that is finding a constrained round trip path.

The adaptation consists in considering bounds for round trip paths and modifying Algorithm SPPP in [28], which constitutes the test procedure mentioned before. The main idea in algorithm SPPP is to scale the cost values and run the pseudopolynomial algorithm to find the smallest possible delay for each cost. Algorithm 3 is our modified implementation of algorithm SPPP.

We denote $D_f(v,i)$ the minimum delay on a forward path from the source node s to a node v, with cost bounded by the value i. Similarly, we use $D_b(v,i)$ to indicate the minimum delay on a backward path from the destination node t to a node v, with cost bounded by the value i. We denote $D(i)$ the minimum delay of a round trip path from s to t, with cost bounded by i.

Algorithm 3 Modified Scaled Pseudo Polynomial Plus [SPPP]

Input: Graph $G(V,E), \{d_l, c_l\}_{l\in E}$, delay T, bounds L, U, factor ϵ, source s, destination t.
Output: Round trip path satisfying the delay bound and corresponding cost.

1) $S \leftarrow \frac{L\epsilon}{2n+1}$
2) for each $l \in E$
3) define $\tilde{c}_l \equiv \lfloor \frac{c_l}{S} \rfloor + 1$
4) $\tilde{U} \leftarrow \lfloor \frac{U}{S} \rfloor + 2n + 1$
5) for all $v \neq s, t$
6) $D_f(v,0) \leftarrow \infty$
7) $D_b(v,0) \leftarrow \infty$
8) $D_f(s,0) \leftarrow 0$
9) $D_b(t,0) \leftarrow 0$
10) for $i = 1, 2, \ldots, \tilde{U}$
11) for $v \in V$
12) $D_f(v,i) \leftarrow D_f(v,i-1)$
13) for $l \in \{(u,v) \mid \tilde{c}_{(u,v)} \leq i\}$
14) $D_f(v,i) \leftarrow \min\{D_f(v,i), d_l + D_f(v, i - \tilde{c}_l)\}$
15) for $i = 1, 2, \ldots, \tilde{U}$
16) for $v \in V$
17) $D_b(v,i) \leftarrow D_b(v,i-1)$
18) for $l \in \{(u,v) \mid \tilde{c}_{(u,v)} \leq i\}$
19) $D_b(v,i) \leftarrow \min\{D_b(v,i), d_l + D_b(v, i - \tilde{c}_l)\}$
20) $D(\tilde{U}) \leftarrow \min_{j=1..\tilde{U}} \{D_f(t,j) + D_b(s, \tilde{U} - j)\}$
21) if $D(\tilde{U}) \leq T$
22) return the corresponding round trip path and cost
23) else return $FAIL$

The resulting algorithm runs in $t = O(|E||V|(\log\log|V| + 1/\varepsilon))$ for each pair of (client, server) nodes.

A fully polynomial time approximation algorithm for Problem 2 can be developed by paralleling the approach for the UKP [15, 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.

B. Polynomial Algorithm for Problem 3

We now address the main problem of Section IV, i.e., the development of a polynomial algorithm for Problem 3, using the approximate costs $\overline{c}_{ij}$ and $\overline{f}_j(y)$. 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.

Proof: The proof is given in Appendix D. ■

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. Indeed, if t is the worst case computation time needed to compute $\varepsilon-$approximate $\overline{c}_{ij}$, $i,j \in V$, according to the algorithm used to solve Problem 1, then the time needed to compute the $|V|^2$ values of $\overline{c}_{ij}$ is $O(|V|^2 t)$. Next, using the $\overline{c}_{ij}$ as input to Algorithm 2, letting T be the worst case computation time needed to compute $\varepsilon-$approximate $\overline{f}_j(y)$ according to the algorithm used to solve Problem 2 and following a similar reasoning as in the proof of (9), it can be seen that the worst case running time of the Algorithm 2 is $O(|V|^3 T))$. Hence the computation time for the whole algorithm is $O(|V|^3 T + |V|^2 t)$ and the approximation factor is $(1+\varepsilon)\log(|V| g_{\max})$. Since as discussed in Section IV-A both t and T are polynomial for given ε, the resulting algorithm is also polynomial.

We now return to the problem at hand. In our case, c_{ij} and $f_j(y)$ are not computable in polynomial time and, moreover, $f_i(y)$ is subadditive rather than concave. Hence the results above cannot be applied directly to obtain a polynomial time algorithm. Regarding c_{ij}, as discussed in Section IV we replace c_{ij} with polynomially computable approximations $\overline{c}_{ij}$.This takes time $O(V^2 t)$. Dealing with $f_j(y)$ requires more care since, as it is also discussed in IV, simply approximating $f_j(y)$ for each y results in pseudopolynomial time algorithm and in no approximation factor guarantees. 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,

$$f_j(y) \leq \widetilde{f}_j(y) \leq a f_j(y). \tag{13}$$

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

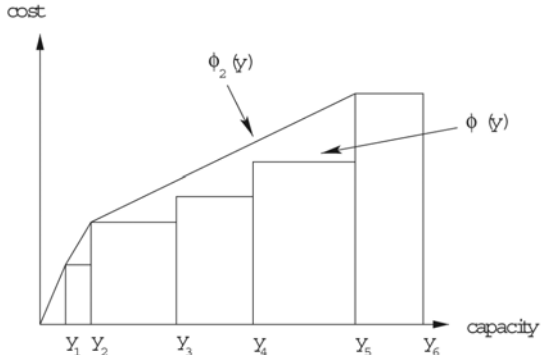

Fig. 3. A subadditive step function $\phi(y)$ with its jump points $y_1, ..., y_6$ and its upper hull $\phi_2(y)$.

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,

$$\phi_2(y) = \phi(y_k) + \frac{\phi(y_{k+1}) - \phi(y_k)}{y_{k+1} - y_k}(y - y_k). \tag{14}$$

The function $\phi_2(y)$ is concave. We call $\phi_2(y)$, the "upper hull" of $\phi(y)$. An example of a subadditive step function and its upper hull is depicted in Figure 3. If $\phi(y)$ is non-decreasing, then $\phi_2(y)$ is also non-decreasing. By construction it holds for all $y \in \{0, 1, ..., W\}$,

$$\phi(y) \leq \phi_2(y). \tag{15}$$

As the next lemma shows, if the function $\phi(y)$ is subadditive and non-decreasing, then it also holds that its upper hull is at most $2\phi(y)$.

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 $\phi_2(y)$, $\phi_2(y) \leq 2\phi(y)$.

Proof: The proof is given in Appendix E. ■

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 $\overline{f}\,(y)$ such that

$$f(y) \leq \overline{f}(y) \leq (1 + \varepsilon) f(y). \tag{16}$$

Let $\overline{f}_2(y)$ be the upper hull of $\overline{f}(y)$. By (16), $\overline{f}_2(y)$ 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

$$f(y) \leq \overline{f}_2(y) \leq 2(1+\varepsilon)f(y). \tag{17}$$

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 (17) and Lemma 1 we will then have a solution to our original problem with approximation factor $2(1+\varepsilon)\log|Vg_{\max}|$.

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 $W'\log W'$ [29]. 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 $\widehat{f}_1(y-1)\neq\widehat{f}_1(y)$) that is a good approximation to $\overline{f}(y)$, 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 4.

Algorithm 4

Input: Algorithm for computing $\overline{f}(y)$, $\epsilon>0$.
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) $\widehat{f}_k=(1+\epsilon)\overline{f}(y_{k-1})$
3) If $\widehat{f}_k>\overline{f}(W)$, set $y_k=W,\ K=k$ and stop. Else,
4) Determine y_k such that $\overline{f}(y_k-1)\leq\widehat{f}_k\leq\overline{f}(y_k)$
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 4, $K=O\left(\frac{1}{\epsilon}\log\frac{\overline{f}(W)}{\overline{f}_1}\right)$.

b)The worst case running time of Algorithm 4 is $O\left(T\log(W)\frac{1}{\epsilon}\log\frac{\overline{f}(W)}{\overline{f}_1}\right)$, where T is the worst case time (over all y, $1 \leq y \leq W$) needed to compute $\overline{f}(y)$.

c) 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,

$$\widehat{f}(y) \leq \overline{f}(y) \leq (1+\epsilon)\widehat{f}(y). \tag{18}$$

Proof: The proof is given in Appendix F. ■

Based on (18), we can use $\widetilde{f}(y) = (1+\epsilon)\widehat{f}(y)$ as a function to approximate $f(y)$. This is shown in the next lemma.

Lemma 5: Let $\epsilon_0 > 0$, $\epsilon > 0$ 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

$$f(y) \leq \overline{f}(y) \leq (1+\epsilon_0)\, f(y). \tag{19}$$

a) For the purposes of computing the step function $\widehat{f}(y)$ satisfying (18), $\overline{f}(y)$ may be assumed non-decreasing.

b) It holds for $\widetilde{f}(y) = (1+\epsilon)\widehat{f}(y)$

$$f(y) \leq \widetilde{f}(y) \leq (1+(\epsilon+\epsilon_0+\epsilon\epsilon_0))\, f(y). \tag{20}$$

c) The number of jump points of $\widehat{f}(y)$, hence of $\widetilde{f}(y)$, is $O\left(\frac{1}{\epsilon}\log\left(|V|\, g_{\max}\right)\right)$ and the running time of Algorithm 4 is $O\left(\frac{1}{\epsilon}T(\log\left(|V|\, g_{\max}\right))^2\right)$, where T is the worst case time (over all y, $1 \leq y \leq W$) needed to compute $\overline{f}(y)$.

Proof: The proof is given in Appendix G. ■

From the discussion above we have polynomial time Algorithm 5 for computing the server locations.

Algorithm 5 Polynomial Time Algorithm For Calculating Server Locations

Input: Polynomial Algorithm for Problem 1, Algorithms 2 and 4, $\epsilon > 0$.
Output: Locations and types of servers, routes and load assigned from each client to the selected locations.

1) For $i \in V$, $j \in H$, compute $\overline{c}_{ij}$ so that $c_{ij} \leq \overline{c}_{ij} \leq (1+\epsilon)c_{ij}$, $i \in V$, $j \in H$.
2) For $j \in H$, construct the step functions $\widehat{f}_j(y)$, from $\overline{f}_j(y)$ according to Algorithm 4, using as subroutine the algorithm for computing, for a given $y > 0$, $\overline{f}_j(y)$ such that $f_j(y) \leq \overline{f}_j(y) \leq (1+\epsilon)f_j(y)$.
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.

Let $|S| = \max_{j\in H}\{|S_j|\}$. Recall that $\phi_j(y),\ j \in H$ are non-decreasing piecewise linear functions with at most K number of break points, where $K = O(\log(|V|\, g_{\max})/\epsilon)$. In fact, the computation of $\phi_j(y)$ in step 3 of Algorithm 5 amounts to storing the break points of $\phi_j(y)$. Hence it takes time $O(\log K)$, in the worst case, to compute $\phi_j(y)$ for a given y and according to the discussion in Section IV-B it takes time $O(|V|^3 \log K)$ to execute Algorithm 2 using $\phi_j(y)$. Taking into account the previous discussion, the worst case running times of each step are:

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

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

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

Note that since in Algorithm 4 the functions $\phi_j(y)$ are concave by construction, by Lemma 2 the algorithm assigns all the load of each node $i \in V$ to a single server node in H.

V. Numerical Results

In this section we evaluate the proposed algorithm using sample topologies following the Power Law. These topologies are taken from the BRITE package [30] by using the Router Barabasi-Albert Model.

We used random parameters for the network, rather than specific application parameters in order to test the overall performance of the algorithm under general conditions. For each network the delay d_l 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 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 are not aware of other approaches in the literature addressing the form of the problem examined in this paper. Hence direct comparison of our proposal to other algorithms cannot be made. However, since there are several proposals that concentrate on metrics related to optimizing the delays of requests in some manner, we examine the performance of the latter algorithms in case operating costs were involved. Specifically, we run two sets of experiments which differ in the manner the round-trip routes for requests are selected.

We generated ten different Power Law network topologies with $|V| = 100$ nodes and $|E| = 970$ edges. We run the algorithm for 6 different delay constraints $D = \{100, 200, 400, 500, 600, 800\}$.

MinDelay: This manner of selecting routes has been employed in [5] and [9] where all requests from client node i to server node j are send over the minimal delay round-trip route. Hence 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.

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 among those that they satisfy the delay constraint.

Once the round-trip routes from any client node i to any node server j are selected using either the MinDelay or MinCost approach, the parameters c_{ij} are determined. We then employ the algorithm proposed in this paper to find a solution to Problem 3. 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.

TABLE I
AVERAGE TOTAL COST FOR DIFFERENT DELAY CONSTRAINTS

D	100	200	400	500	600	800
MinCost	178638	127591	118069	115734	114504	113878
MinDelay	197832	188928	182178	182178	182178	182178

In Table I 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

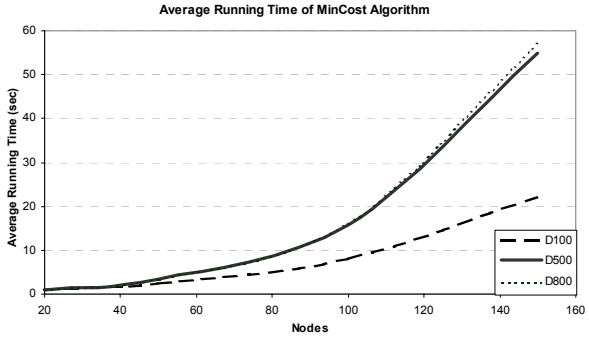

Fig. 4. Average running time of Power Law Networks, $D = \{100, 500, 800\}$.

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 Table I 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.

We also performed experiments for networks of various sizes in order to assert the performance of the proposed algorithm in terms of running time. We generated Power Law directed networks for $|V| = \{20, 50, 100, 150\}$, ratio $r = |E|/|V|$ equal to 3 and two delay constraints $D = \{100, 500, 800\}$. For each $|V|$ and r we generated ten different networks and for each experiment the link cost and delay, the server types, as well as the load of the nodes were generated according to the methods previously described.

The experiments were run on a Pentium PC 4, 1.7GHz, 786MB RAM. In Figure 4 we present the average running time (in seconds) of the proposed algorithm. We make the following observations. For a fixed number of nodes and edges we observe an increase of the running time as the delay constraint increases. The increase becomes more pronounced as the number of nodes increases. This is explained by the fact that as the delay constraint and the number of nodes increase, the number of candidate route-trip paths increases and as a results more iterations are needed for the algorithm to converge. We also observe that for a fixed number of nodes

the increase in running time levels-off when the delay constraint becomes large (800 in our experiments). This is due to the fact that as the delay constraint increases, only the cost of a round-trip path becomes the determining factor for its inclusion in the algorithm; hence the number of candidate round-trip paths levels-off as the delay constraint increases. In general, although pseudopolynomial, tests with a wide variety of networks, show that the algorithm has fairly satisfactory performance. Note that we have assumed $H = V$, that is, servers may be placed in any of the nodes. In practical systems servers are placed in specific locations, that is $H \subset V$. In such cases the running time of the algorithm decreases significantly. Indeed, we run an experiment for a network with $|V| = 100$ nodes, ratio $r = |E| / |V|$ equal to 3, constraint $D = 500$ and ten possible server locations, that is $|H| = 10$, which are randomly chosen among the set of nodes V. The running time and the overall cost were found equal to 3.5 sec and 269722 respectively. By assuming $H = V$ and repeating the latter experiment we observe a) an increase of the running time (15.9sec), b) a decrease of the overall cost (120467) and the placement of servers in 17 different nodes.

VI. 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 significantly simpler than the polynomial time algorithm. The polynomial time algorithm is significant from the 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, D. Li, X. Hu, 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] ——, "SCAN: A dynamic scalable and efficient content distribution network," in *First International Conference on Pervasive Computing*, 2002.

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

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

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

[11] X. Tang and S. Chanson, "The minimal cost distribution tree problem of recursive expiration-based consistency management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 3, pp. 214–227, March 2004.

[12] ——, "Minimal cost replication of dynamic web contents under flat update delivery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, pp. 431–439, May 2004.

[13] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks." ACM SIGCOMM, 2002.

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

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

[25] 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.

[26] 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.

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

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

[29] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*. Springer-Verlag, 2000.

[30] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal topology generation from a user's perspective, Tech. Rep. 2001-003, 1 2001.

APPENDIX

In this appendix we show how the average time of Algorithm 2 can be improved, we present the worst case running time of Algorithm 2 and provide proofs of the lemmas presented in this paper.

A. Step Function $f_j(y)$.

In this section it is shown that the average running time of Algorithm 2 can be improved by taking advantage of the fact that in this case $f_j(y)$ is a step function. For simplicity we drop the index j. Assume the unassigned nodes arranged in non-decreasing order of their costs $c_1 \leq c_2 \leq ... \leq c_m$. Let l be the load already assigned. We then have to compute

$$\min_k \frac{f(l+k) - f(l) + \sum_{i=1}^{k} c_i y_i + y c_{k+1}}{k}.$$

Let

$$Y_k = \sum_{i=1}^{k} y_i,\ C_k = \sum_{i=1}^{k} c_i y_i,\ k = Y_k + y,\ 0 \leq y \leq g_{k+1},$$

and

$$t(y) = \frac{f(l+k) - f(l) + \sum_{i=1}^{k} c_i y_i + y c_{k+1}}{k} =$$
$$= \frac{f(l+Y_k+y) - f(l) + C_k + c_{k+1} y}{Y_k + y} = \quad (21)$$
$$= \frac{f(l+Y_k+y) - f(l) + C_k - c_{k+1} Y_k}{Y_k + y} + c_{k+1} \quad (22)$$

Assume that $f(z)$ is a step function

$$f(z) = f_m,\ \text{if } z_m \leq z < z_{m+1},\ m = 1, 2, ...,$$

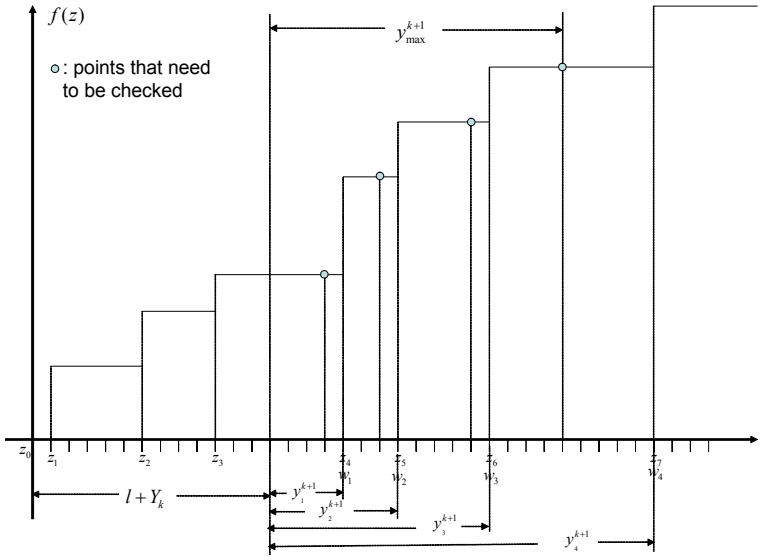

Fig. 5. Points that need to be checked when checking c_{k+1}

where $z_0 = 0,\ f(0) = 0$.We call z_m the point of the m^{th} jump of the function $f(z)$.

Assume that $y \geq 0$ can take maximum value $y_{\max}^{k+1}$, i.e., $y_{\max}^{k+1}$ is the maximum remaining load of the node connected through link $k+1$.

For the rest of the discussion, Figure 5 will be helpful

Let m_{k+1} be the index of the last point before $l + Y_k$ at which a jump of the function $f(z)$ occurs,

$$m_{k+1} = \max\{m :\ z_m \leq l + Y_k\}.$$

Also, let w_m^{k+1}be the m^{th} point after $l + Y_k$ at which a jump of $f(z)$ occurs,

$$w_m^{k+1} = z_{m_{k+1}+m},\ m = 1, 2, ...,$$

and

$$y_m^{k+1} = w_m^{k+1} - l - Y_k$$

Observe from (22) the following:

If

$$f(l + Y_k) - f(l) + C_k - c_{k+1}Y_k < 0$$

then $t(y) < c_{k+1}$ for any $y,\ 1 \leq y \leq y_1^{k+1}$and the algorithm must stop. Note that this cannot happen when we examine c_1 and hence the algorithm always gives some output before it stops.

If on the other hand

$$f(l+Y_k)-f(l)+C_k-c_{k+1}Y_k \geq 0,$$

then the minimum is achieved at $\min\left\{y_1^{k+1}-1,\ y_{\max}\right\}$. Moreover, in this case it holds (since $f(z)$ is increasing),

$$f(l+Y_k+y_m^{k+1})-f(l)+C_k-c_{k+1}Y_k \geq 0,\ \text{for all } m,$$

which implies that the minimum in the interval $\left[y_m^{k+1},\ y_{m+1}^{k+1}-1\right]$ is achieved at

$$\min\left\{y_{m+1}^{k+1}-1,\ y_{\max}\right\}.$$

The above discussion implies that when we need to find the minimum value of $t(y)$, we need only check its values at the points

$$y=y_m^{k+1}-1,\ \text{where } y_m^{k+1}-1 \leq\ y_{\max},\ \text{and } y=y_{\max}.$$

B. Running time of Algorithm 2.

We discuss below an implementation of Algorithm 2 and present its complexity. Let $|V|$ and $|H|$ be the number of client and server locations respectively. We assume for simplicity that $g_{\max}=1$. As discussed in Section III-C, the general case can be treated by considering a modified network with at most $|V|\,g_{\max}$ nodes. We also assume unit cost for compute $f_j(k)$. All references to steps below, concern Algorithm 2.

The implementation is the following.

1) We construct $|H|$ doubly linked lists, one per server location, where each list, L_j, $j\in H$, holds c_{ij}, for all $i\in V$ that are connected to j, in increasing order. Since each sorting takes time $O(|V|\log|V|)$, the time to construct the $|H|$ linked lists is $O(|H|\,|V|\log|V|)$.
2) We also create a matrix $M(i,j)$, $i\in V$, $j\in H$, where $M(i,j)$ holds the address of link (i,j) in the list L_j. If link (i,j) does not exist in L_j, then we set $M(i,j)$ to null. This takes time $O(|H|\,|V|)$. The matrix $M(i,j)$ is used to erase efficiently an element from L_j (see line 3b).
3) In the while loop (step 3) the following operations are performed.
    a) The $t(j)$ for each server location $j\in H$ are computed (step 5). For this, we need to scan through the list L_j, which takes $O(|V|)$ time (since each L_j contains at most

$|V|$ elements). Hence to compute all $t(j)$, $j \in H$ and the minimum of these $t(j)$ (step 7), takes time $O(|H|\,|V|)$.

b) The updating of server and client location loads (steps 9-13) takes $O(1)$ time. However, whenever the remaining load of a node becomes 0 (step 11), the node must be removed from further consideration. Hence, it must be removed from all linked lists L_j, $j \in H$. Therefore, for each of the k^* client locations, say location i, we check first if the link (i,j) exists in the list L_j, i.e. check whether $M(i,j)$ is non null. If so, $M(i,j)$ contains the address of link (i,j) in L_j; using this address we remove link (i,j) from L_j and we set $M(i,j)$ to null. This takes time $O(k^*\,|H|)$. Note that we can dispense with the matrix $M(i,j)$ altogether and search through the whole list L_j in order to remove the link (i,j). This will increase the complexity of this operation to $O(k^*\,|V|\,|H|)$ but will also eliminate the space needed to store the matrix $M(i,j)$. The overall complexity of the algorithm will not be affected.

Regarding the overall complexity, we have the following.

- Line 1 is executed outside the while loop and hence takes time $O(|H|\,|V|\log|V|)$.
- Line 3a is executed inside the while loop. Since in the worst case only one client location may be removed at each iteration, this step may be executed a most $|V|$ times. Hence the total complexity of this operation is $O(|H|\,|V|^2)$.
- If k_n is the number of customers allocated to a facility at iteration n, and it takes m iterations for the algorithm to end, we have $k_1 + k_2 + k_3 + .. + k_m = |V|$. Hence line 3b is executed in time $O(k_1\,|H|) + O(k_2\,|H|) + ... + O(k_m\,|H|) = O(|H|\,|V|)$.

From the above we conclude that the whole algorithm executes in worst case time, $O\left(|H|\,|V|\log(|V|) + |H|\,|V|^2\right) = O\left(|H|\,|V|^2\right)$, which is dominated by the time to compute $t(j)$ in step 5. For general loads, replacing $|V|$ with $|V|\,g_{\max}$, the complexity becomes $O\left(|H|\,|V|^2\,g_{\max}^2\right)$. Since in our setup $|H|$ can be as large as $|V|$, the overall complexity of the algorithm is $O(|V|^3\,g_{\max}^2)$.

C. Proof of Lemma 1

Lemma 1: Consider the problems

$$\min_{\mathbf{x}\in A} g(\mathbf{x}),\ A \subseteq R^n, \tag{23}$$

$$\min_{\mathbf{x}\in A} \overline{g}(\mathbf{x}),\ A \subseteq R^n, \tag{24}$$

where $g(\mathbf{x}) \geq 0$. If for $\mathbf{x} \in A$,

$$g(\mathbf{x}) \leq \overline{g}(\mathbf{x}) \leq bg(\mathbf{x}), \tag{25}$$

then an $a-$approximate solution for problem (24), $a \geq 1$, is a ba-approximate solution for (23).

Proof: Let x^*, $\overline{x}^*$ be the optimal solutions to (23), (24) respectively. Let also $\overline{\mathbf{y}}$ be an a-approximate solution for problem (24), i.e.

$$\overline{g}(\overline{\mathbf{x}}^*) \leq \overline{g}(\overline{\mathbf{y}}) \leq a\overline{g}(\overline{\mathbf{x}}^*). \tag{26}$$

Since $\overline{x}^*$ is optimal for problem (24), it holds

$$\overline{g}(\overline{\mathbf{x}}^*) \leq \overline{g}(\mathbf{x}^*) \leq bg(\mathbf{x}^*) \quad \text{by (25)} \tag{27}$$

Then

$$\begin{aligned} g(\mathbf{x}^*) &\leq g(\overline{\mathbf{y}}) \text{ since } \mathbf{x}^* \text{solves (23)} \\ &\overset{\text{by (25)}}{\leq} \overline{g}(\overline{\mathbf{y}}) \overset{\text{by (26)}}{\leq} a\overline{g}(\overline{\mathbf{x}}^*) \overset{\text{by (27)}}{\leq} abg(\mathbf{x}^*). \end{aligned}$$

Hence $\overline{\mathbf{y}}$ is a ab solution to (23). ■

D. Proof of Lemma 2

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.

Proof: Assume that at the beginning of the tth iteration of the while loop in Algorithm 2 (step 3) the load of any assigned node, has been actually assigned to a single server. We will show that the same is true at the beginning of the $t+1$th iteration. The result will follow by induction.

It suffices to show that the minimum in step 5 is achieved when all the load of each unassigned node is assigned to a single server. To simplify notation, let $c_1 \leq c_2 \leq ... \leq c_m$ be the unassigned nodes connected to node j, and let l be the load assigned to node j at the tth iteration. For simplicity, we drop the index j from the notation. We then have to compute

$$\min_k \frac{f(l+k) - f(l) + \sum_{i=1}^{n(k)} c_i g_i + l(k) c_{n(k)+1}}{k}. \tag{28}$$

Let $Y = \sum_{i=1}^{n} g_i$ and $k = Y + y$, $0 \le y \le g_{n+1}$. It suffices to show that the minimum of

$$s(y) = \frac{f(l+Y+y) - f(l) + \sum_{i=1}^{n} c_i g_i + y c_{n+1}}{Y+y},\ 0 \le y \le g_{n+1},$$

is achieved either at $y = 0$ or at $y = g_{n+1}$. Indeed this implies that the minimum in (28) is achieved when $k = \sum_{i=1}^{j} g_i$ for some j, $1 \le j \le m$, i.e., that all the node loads are assigned to a single server.

Let us rewrite

$$\begin{aligned} s(y) &= \frac{f(l+Y+y) - f(l) + \sum_{i=1}^{n} c_i g_i + c_{n+1} y}{Y+y} \\ &= \frac{f(l+Y+y) - f(l) + \sum_{i=1}^{n} (c_i - c_{n+1}) g_i}{Y+y} + c_{n+1} = \frac{\phi(Y+y)}{Y+y} + c_{n+1}, \end{aligned}$$

where $\phi(x) = f(l+x) - f(l) + \sum_{i=1}^{n} (c_i - c_{n+1}) g_i$.

Notice that $\phi(x)$ is concave. Suppose that a minimum of $s(y)$ occurs at $y = y_0$, such that $0 < y_0 < g_{n+1}$. By the definition of y_0 and the fact that $y_0 > 0$, we have

$$\frac{\phi(Y+y_0)}{Y+y_0} \le \frac{\phi(Y+y_0-1)}{Y+y_0-1}$$

or

$$\phi(Y+y_0) - \phi(Y+y_0-1) \le \frac{\phi(Y+y_0)}{Y+y_0}.$$

Hence, using the concavity of $\phi(x)$,

$$\phi(Y+y_0+1) - \phi(Y+y_0) \le \frac{\phi(Y+y_0)}{Y+y_0}$$

and

$$\frac{\phi(Y+y_0+1)}{Y+y_0+1} \le \frac{\phi(Y+y_0)}{Y+y_0}.$$

Therefore $y_0 + 1$ is a minimum too. Similarly, using the fact that $y_0 < g_{n+1}$ we conclude that $y_0 - 1$ is also a minimum.

This implies by induction that $\frac{\phi(Y+y)}{Y+y}$ is constant for $0 \le y \le g_{n+1}$ and hence the same holds for $s(y)$. We conclude that either the minimum is achieved at $y = 0$ or at $y = g_{n+1}$, or $s(y)$ is a constant for $0 \le y \le g_{n+1}$. In the latter case we can pick $y = g_{n+1}$ as the minimizing point. ■

E. Proof of Lemma 3

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 $\phi_2(y)$, $\phi_2(y) \leq 2\phi(y)$.

Proof: For $y = 0$ and $y = W$ we have by definition $\phi_2(y) = \phi(y)$. Let now y be an integer such that $y_k < y \leq y_{k+1}$. Consider two cases

1) $y_{k+1} - y_k < y$. Then

$$\begin{aligned}\phi_2(y) &\leq \phi(y_{k+1}) \text{ by (14)}\\ &\leq \phi(y_k + y) \text{ since } \phi \text{ is non-decreasing}\\ &\leq \phi(y_k) + \phi(y) \text{ by subadditivity}\\ &\leq 2\phi(y) \text{ since } \phi \text{ is non-decreasing.}\end{aligned}$$

2) $y_{k+1} - y_k \geq y$. Since $\phi_2(y)$ is piecewise linear and y_k, y_{k+1} are consecutive break points, it holds for any z, $y_k \leq z < y_{k+1}$,

$$\frac{\phi_2(y_{k+1}) - \phi_2(y_k)}{y_{k+1} - y_k} = \frac{\phi_2(y_{k+1}) - \phi_2(z)}{y_{k+1} - z}.$$

Since $\phi_2(y_k) = \phi(y_k)$, $k = 1, ..., K$,

$$\begin{aligned}\frac{\phi(y_{k+1}) - \phi(y_k)}{y_{k+1} - y_k} &= \frac{\phi(y_{k+1}) - \phi_2(z)}{y_{k+1} - z}\\ &\leq \frac{\phi(y_{k+1}) - \phi(z)}{y_{k+1} - z} \text{ by (15)}\\ &\leq \frac{\phi(y_{k+1} - z)}{y_{k+1} - z} \text{ by subadditivity.} \qquad (29)\end{aligned}$$

Now set $z = y_{k+1} - y$. Since $y_k \leq z < y_{k+1}$ by hypothesis, using the last inequality we have,

$$\begin{aligned}\phi_2(y) &= \phi(y_k) + \frac{\phi(y_{k+1}) - \phi(y_k)}{y_{k+1} - y_k}(y - y_k)\\ &\leq \phi(y_k) + \frac{\phi(y)}{y}(y - y_k) \text{ by (29)}\\ &\leq \phi(y_k) + \phi(y) \leq 2\phi(y) \text{ since } \phi \text{ is non-decreasing.}\end{aligned}$$

This completes the proof. ■

F. Proof of Lemma 4

Lemma 4: a) In Algorithm 4, $K = O\left(\frac{1}{\epsilon}\log\frac{\overline{f}(W)}{\overline{f}_1}\right)$.

b)The worst case running time of Algorithm 4 is $O\left(T\log(W)\frac{1}{\epsilon}\log\frac{\overline{f}(W)}{\overline{f}_1}\right)$, where T is the worst case time (over all y, $1 \le y \le W$) needed to compute $\overline{f}(y)$.

c) Consider the step function defined as follows: if $y_k \le y < y_{k+1}$ for some k, $1 \le k \le K-1$, then $\widehat{f}(y) = \widehat{f}_k$, and $\widehat{f}(W) = \widehat{f}_K$. It holds,

$$\widehat{f}(y) \le \overline{f}(y) \le (1+\epsilon)\widehat{f}(y). \tag{30}$$

Proof: a) According to steps 2 and 4 it holds,

$$\overline{f}(y_k - 1) \le (1+\epsilon_1)\overline{f}(y_{k-1}) \le \overline{f}(y_k). \tag{31}$$

Since $\overline{f}(y)$ is non-decreasing and $\overline{f}(1) > 0$ (hence $\overline{f}(y) > 0$, $y \ge 1$), inequality (31) implies that $y_{k-1} < y_k$. Moreover, $(1+\epsilon)\widehat{f}_{k-1} \le (1+\epsilon)\overline{f}(y_{k-1}) = \widehat{f}_k$.This implies that $(1+\epsilon)^{k-1}\,\widehat{f}_1 \le \widehat{f}_k$. Hence the algorithm stops in $O\left(\log_{1+\epsilon}\frac{\overline{f}(W)}{\overline{f}_1}\right) = O\left(\log\frac{\overline{f}(W)}{\overline{f}_1}/\epsilon\right)$ steps.

b) Since $\overline{f}(y)$ is non-decreasing, we can use binary search in $[1, W]$ to find each y_k. Hence the determination of each y_k takes $O(T\log W)$ computations and the total running time is $O(TK\log W)$.

c) For $y_k \le y < y_{k+1}$, taking into account (31) we have

$$\begin{aligned}\overline{f}(y) &\le \overline{f}(y_{k+1}-1) \text{ since } \overline{f}(y) \text{ is non-decreasing}\\ &\le (1+\epsilon)\overline{f}(y_k) = (1+\epsilon)\widehat{f}(y).\end{aligned}$$

By definition, $\widehat{f}(y) = \overline{f}(y_k)$. Taking also into account that $\overline{f}(y_k) \le \overline{f}(y)$, (30) follows. ■

G. Proof of Lemma 5

Lemma 5: Let $\epsilon_0 > 0$, $\epsilon > 0$ 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

$$f(y) \le \overline{f}(y) \le (1+\epsilon_0)\, f(y). \tag{32}$$

a) For the purposes of computing the step function $\widehat{f}(y)$ satisfying (30), $\overline{f}(y)$ may be assumed non-decreasing.

b) It holds for $\widetilde{f}(y) = (1+\epsilon)\widehat{f}(y)$

$$f(y) \leq \widetilde{f}(y) \leq (1 + (\epsilon + \epsilon_0 + \epsilon\epsilon_0))\, f(y). \tag{33}$$

c) The number of jump points of $\widehat{f}(y)$, hence of $\widetilde{f}(y)$, is $O\left(\frac{1}{\epsilon}\log\left(|V|\, g_{\max}\right)\right)$ and the running time of Algorithm 4 is $O\left(\frac{1}{\epsilon}T(\log\left(|V|\, g_{\max}\right))^2\right)$, where T is the worst case time (over all y, $1 \leq y \leq W$) needed to compute $\overline{f}(y)$.

Proof: a) To see that for the purposes of constructing $\overline{f}(y)$ we can assume without loss of generality that $\overline{f}(y)$ is non-decreasing, proceed as follows. If at some point during the computations for the construction of $\overline{f}(y)$ we obtain for $y_1 < y_2$, $\overline{f}(y_1) > \overline{f}(y_2)$, then we can replace $\overline{f}(y_2)$ with $\overline{f}_1(y_2) = \overline{f}(y_1)$ without violating (32). Indeed,

$$f(y_2) \overset{\text{by (32)}}{\leq} \overline{f}(y_2) < \overline{f}(y_1) = \overline{f}_1(y_2), \text{ by assumption}$$

and

$$\overline{f}_1(y_2) = \overline{f}(y_1) \overset{\text{by (32)}}{\leq} (1+\epsilon_0)\, f(y_1) \leq (1+\epsilon_0)\, f(y_2) \text{ since } f(y) \text{ is increasing.}$$

b) Let $y_k \leq y \leq y_{k+1} - 1$. We then have

$$\begin{aligned} f(y) &\overset{\text{by (32)}}{\leq} \overline{f}(y) \overset{\text{by (30)}}{\leq} (1+\epsilon)\widehat{f}(y) \\ &= \widetilde{f}(y). \end{aligned} \tag{34}$$

On the other hand,

$$\begin{aligned} f(y) &\overset{\text{by (32)}}{\geq} \frac{1}{1+\epsilon_0}\overline{f}(y) \overset{\text{by (30)}}{\geq} \frac{1}{1+\epsilon_0}\widehat{f}(y) \\ &= \frac{\widetilde{f}(y)}{(1+\epsilon_0)\,(1+\epsilon)}. \end{aligned} \tag{35}$$

From (34), (35), we obtain (33).

c) Since $f(y)$ is subadditive, it holds $f(W)/f(1) \leq W$. The result then follows from Lemma 4 a) and b). ■