Replica Placement via Capacitated Vertex Cover*

Sonika Arora[1], Venkatesan T. Chakaravarthy[2], Neelima Gupta[1], Koyel Mukherjee[3], and Yogish Sabharwal[2]

1 **Department of Computer Science, University of Delhi**
sonika.ta@gmail.com, ngupta@cs.du.ac.in
2 **IBM India Research Lab, New Delhi, India**
{vechakra,ysabharwal}@in.ibm.com
3 **Department of Computer Science, University of Maryland, College Park**
koyelm@cs.umd.edu

Abstract

In this paper, we study the replica placement problem on trees and present a constant factor approximation algorithm (with an additional additive constant factor). This improves the best known previous algorithm having an approximation ratio dependent on the maximum degree of the tree. Our techniques also extend to the partial cover version. Our algorithms are based on the LP rounding technique. The core component of our algorithm exploits a connection between the natural LP solutions of the replica placement problem and the capacitated vertex cover problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, LP Rounding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.263

1 Introduction

The Replica placement problem in tree networks is a well-studied problem [4, 12, 2, 1] and several variants of it have been examined in the literature. In this paper, we study an important version called replica placement with distances and present a first constant factor approximation algorithm (with an additional additive constant factor) for this problem. Our techniques also extend to the partial cover version of the problem yielding a similar result.

Replica Placement Problem. The input consists of a *rooted* tree (or tree network) $G = (V, E)$. Each leaf node represents a *client* and let A be the set of all clients. Let $|A| = m$. The input specifies a *request* $r(a)$ for each client $a \in A$. The input also includes a *capacity* W. For each edge (u, v) in the tree, the input specifies a distance $d(u, v)$. For a node u and a client a such that u is an ancestor of a, let $d(u, a)$ be the distance from u to a. Each client a is associated with a quantity $d_{\max}(a)$, the maximum distance its request can travel. We assume that the capacity W and the requests $r(\cdot)$ are integral. Furthermore, we assume that $r(a) \leq W$ for all clients $a \in A$ and that W is polynomially bounded in the number of nodes.

A feasible solution selects a subset of nodes and places replicas on them in order to service the requests of the clients. The solution must assign the requests of the clients to the replicas. The request of a client a can be assigned to a node u, only if u is an ancestor of a and $d(a, u) \leq d_{\max}(a)$. Furthermore, the total request assigned to any replica must not exceed

* Full version of the paper is available as an arxiv preprint.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 263–274

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the capacity W. We allow the solution to open replicas at the leaf nodes corresponding to the clients. Our goal is to minimize the number of replicas placed.

In the partial cover version of the replica placement problem, the input also includes an integer $K \leq m$, called the *partiality parameter* and it suffices if some K clients are serviced. Namely, a feasible solution selects a subset of nodes for placing replicas and a set of K clients so that the replicas can service the set of K clients. The full cover version corresponds to the case where K is the number of clients.

Prior Work. The replica placement problem finds applications in internet and video on demand service delivery (see [7, 9, 4]). We refer to [12] for additional applications. The above problem and its variants have been well-studied in the existing literature [4, 12, 2, 8, 1], from both practical and algorithmic perspectives. From an algorithmic perspective, the full cover version and its special cases have been addressed in prior work and approximation algorithms have been developed.

Benoit et al. [2, 1] studied the full cover version of the replica placement problem. They showed that it is NP-hard to approximate the problem within a factor of 3/2 even when there are no distance constraints (i.e., $d_{\max}(a) = \infty$, for all clients a) and the given network is a binary tree. They obtained the above result by showing that the above problem generalizes the bin-packing problem. Benoit et al. [1] presented a 2-approximation for the replica placement problem without distances. For the case with distances, they designed a greedy algorithm with an approximation ratio of $(1 + \Delta)$, where Δ is the maximum number of children of any node. Regarding the replica placement problem, we are not aware of any prior work on the partial cover versions.

Our Results and Discussion. The main goal of this paper is to study the full cover and the partial cover versions of the replica placement problem (with distances). As discussed earlier, the best known algorithm for the full cover version [1] has an approximation ratio of $(1 + \Delta)$, where Δ is the maximum number of children of any node in the tree. Clearly, the parameter Δ could be of the order of the number of nodes in the tree. The main contribution of the paper is a constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version.

> **Main Result:** We present a constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version of the replica placement problem (with distances).

The algorithms of Benoit et al. [2, 1] for different versions of the replica placement problem are primarily based on dynamic programming and the greedy method. It is not clear how to use such techniques to derive constant factor approximation algorithms that handle the dual aspects of distance constraints and partial covering. We should mention that the constant factor obtained in the paper is fairly large. The constant factor can be substantially improved via a more careful book keeping. However, for the sake of exposition and brevity, we defer such an analysis to the full version of the paper.

Our algorithms are based on the LP rounding technique. Our main technical contribution is to show a connection between the natural LP solutions of the replica placement problem and the capacitated vertex cover problem. We will exploit this connection to derive approximation algorithms for our problems. We believe our techniques can be applied to other special cases of the capacitated set cover problem as well. The above connection and our proof techniques are outlined below.

Relationship to Capacitated Set Cover and Proof Techniques. The capacitated set cover problem generalizes the classical set cover problem. In this problem, we are given a set system consisting of a universe of elements U and a collection $\mathcal{S}$ of subsets of the universe. Each set $S \in \mathcal{S}$ in the collection has associated capacity $w(S)$. A feasible solution selects a collection of sets $\mathcal{R} \subseteq \mathcal{S}$ and assigns each element $a \in U$ to some set $S \in \mathcal{R}$ such that $a \in S$. Furthermore, the solution must satisfy the property that for any selected set S, the number of elements assigned to it does not exceed $w(S)$.

Note that in the above problem definition we do not associate requests with the elements, because in such generalization, it is NP-hard even to test whether a solution exists and so, the problem is inapproximable [3]. In our problems, such an issue does not arise, since we allow replicas to be placed at the client nodes. Clearly, the unit request case of the full cover version of the replica placement is a special case of the capacitated set cover problem.

The capacitated vertex cover problem is the special case where each element a appears in exactly two sets. Equivalently, we are given a multi-graph wherein each vertex has a capacity. The solution must select a vertex cover and assign each edge to one of its end points in the cover such that the capacity is not violated at any vertex.

The capacitated set cover problem can be approximated within a factor of $O(\log n)$ using the work of Wolsey [11]. The problem is NP-hard to approximate within a factor of $\Omega(\log n)$ [5]. Chuzhoy and Naor [3] and Gandhi et al. [6] presented algorithms for the capacitated vertex cover problem with an approximation ratio of 3 and 2, respectively. However, their algorithms can handle only the case of simple graph (no parallel edges). Saha and Khuller [10] presented a 34-approximation algorithm for the more general case of multi-graphs. Their technique also extends to hypergraphs and yields an $O(f)$-approximation algorithm for the capacitated set cover problem, where the frequency parameter f is the maximum number of sets an element appears in. All these algorithms are based on rounding of a natural LP formulation of the problem.

Our algorithm for the partial cover version of the replica placement problem also goes via considering a natural LP formulation. The core component of our algorithm takes any LP solution and carefully transforms it into a solution that can be construed as an LP solution of a suitable instance of the capacitated vertex cover problem on multi-graphs. This allows us to utilize the rounding procedure of Saha and Khuller [10] and obtain a constant factor approximation algorithm for our problem.

We highlight some of the challenges involved in the above transformation. Firstly, in our problem, a client can be assigned to an arbitrary number of nodes, whereas in the capacitated vertex cover scenario, an edge can be assigned to only two nodes. Thus the transformation lets us move from a set system having arbitrary frequency parameter to a setting where the parameter is two. Secondly, the transformation converts a solution for a partial cover setting to a full cover setting. In this context, we should note that Saha and Khuller [10] present an algorithm for partial cover version of the capacitated set cover problem. However, they obtain the result via a simple reduction from the partial cover version to the full cover version via creation of a dummy set. We do not know of any such simple reduction for our setting. Thirdly, we note that a client would be considered serviced and counted as part of the partiality parameter K, only when the whole of its request is serviced. Despite the above challenges, there are certain aspects of the problem that let us perform the transformation. The first is that our problem definition allows a solution to place a replica on a client node itself. Secondly, the set systems arising in our context enjoy some special properties because of the structure of the tree.

2 Overview of the Main Result

The goal of this section is to establish the main result of the paper.

▶ **Theorem 1.** *There exists a polynomial time constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version of the replica placement problem.*

The algorithm goes via LP rounding. Here, we shall present an intuitive and detailed overview of the algorithm and prove the above result. In doing so, we will highlight the different components of the algorithm. Discussion on the individual components are deferred to the subsequent sections.

LP Formulation: We consider a natural LP formulation. We say that a solution *opens* node u, if it places a replica on it. We say that a client a is *attachable* to a node u, if u is an ancestor of a and $d_{\max}(a) \leq d(u,a)$. Let $\texttt{Att}(a)$ denote the set of all nodes to which the client a can be attached and let $\texttt{Att}(u)$ denote the set of all clients that can be attached to a node u.

For each node $u \in V$, we introduce a variable $y(u)$ that specifies the extent to which u is open. For each client $a \in A$ and each node $u \in \texttt{Att}(a)$, we introduce a variable $x(a,u)$ that specifies the extent to which a is assigned to u.

$$\min \quad \sum_{u \in V} y(u)$$

$$\sum_{a \in \texttt{Att}(u)} x(a,u) \cdot r(a) \leq y(u) \cdot W \qquad (\forall u \in V) \tag{1}$$

$$\sum_{a \in A} \sum_{u \in \texttt{Att}(a)} x(a,u) \geq K \tag{2}$$

$$x(a,u) \leq y(u) \quad (\forall a \in A, u \in \texttt{Att}(a)) \tag{3}$$

$$\sum_{u \in \texttt{Att}(a)} x(a,u) \leq 1 \qquad (\forall a \in A) \tag{4}$$

$$y(u) \leq 1 \qquad (\forall u \in V) \tag{5}$$

Further, we add non-negativity constraints for all the variables. Constraint (1) (called the *capacity constraint*) enforces that at any node the total request assigned does not exceed the capacity W. Constraint (2) ensures that at least K clients are served. Constraint (3) ensures that a client can be assigned to a node only to an extent the node is open; without this constraint, it can be shown that the LP has an unbounded integrality gap. Constraint (4) enforces that every client is serviced to an extent of at most one. Constraint (5) requires that a node can be opened to an extent of at most one. Consider an LP solution $\sigma = \langle x, y \rangle$. The cost of an LP solution is given by the objective function: $\texttt{cost}(\sigma) = \sum_{u \in V} y(u)$.

LP Rounding: It is not difficult to show that the above LP has an unbounded integrality gap if we restrict ourselves to multiplicative approximation ratios. However, we shall show that the above issue can be overcome by allowing additive errors and prove the following result. We note that in the following lemma and all the subsequent lemmas, the transformations claimed can be implemented in polynomial time.

▶ **Lemma 2.** *Any LP solution σ_{in} can be transformed into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq c_1 \cdot \texttt{cost}(\sigma_{in}) + c_2$, where c_1 and c_2 are constants.*

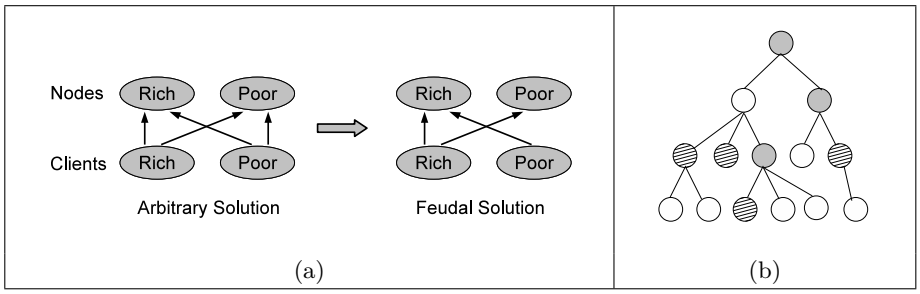

Figure 1 (a) Transformation from arbitrary solution to feudal solution; (b) Hierarchical solution.

The above lemma implies the main result (Theorem 1). We next present an overview of the proof of the lemma. Consider an LP solution $\sigma = \langle x, y \rangle$. We say that a node u is *fully-open*, if $y(u) = 1$; it is said to be *fully-closed*, if $y(u) = 0$. The node is said to be *partially open*, if $0 < y(u) < 1$. We shall make a similar classification of the clients. Let a be a client and consider the extent to which it is served under σ, namely, define $\rho(a) = \sum_{u \in \texttt{Att}(a)} x(a, u)$. The client a is said to be: (i) *fully-serviced*, if $\rho(a) = 1$; (ii) *partially serviced*, if $0 < \rho(a) < 1$; (iii) *unserviced*, if $\rho(a) = 0$. The client a is said to be *serviced*, if $\rho(a) > 0$. A node u is said to service a client a, if $x(a, u) > 0$. The solution σ is said to be *integrally open*, if every node is either fully-open or fully-closed. Similarly, it is said to be *integrally serviced*, if every client is either fully-serviced or unserviced.

Our LP rounding procedure works in four stages and it goes via the important notion of dual assigned solutions. A solution is said to be *dual assigned*, if every client is serviced by at most two nodes. The four stages are as below:

1. Converts any LP solution σ_{in} into an integrally serviced solution σ_1.
2. Converts σ_1 into an integrally serviced, dual assigned solution σ_2.
3. Converts σ_2 into an integrally open, integrally serviced solution σ_3.
4. Converts σ_3 into an integral solution σ_{out}.

Intuitively, the first stage identifies the K clients to be serviced and helps us move from the partial cover setting into a full cover setting. In an arbitrary LP solution, a client may be serviced by any number of nodes. The scenario corresponds to the capacitated set cover setting. On the other hand, a dual assigned solution corresponds to the capacitated vertex cover setting. Thus, the second stage helps us move from the capacitated set cover setting to the capacitated vertex cover setting. These two stages form the main technical component of the paper. As far as the third stage is concerned, we invoke a rounding algorithm for the capacitated vertex cover problem, due to Saha and Khuller [10] and obtain an integrally open, integrally serviced solution. The only task remaining is to make the assignments (i.e., $x(\cdot, \cdot)$) integral, which is performed by the last stage.

Stage 1: Integrally Serviced Solutions. We first identify groups of nodes that are approximately fully-open. Let $\sigma = \langle x, y \rangle$ be an LP solution. We say that a node u is *rich* if $y(u) \geq 1/3$. The node u is said to be *poor* if $0 < y(u) < 1/3$. We also use a similar terminology for the clients. A client a is said to be *rich* if the extent to which it is serviced is at least $1/3$, i.e., $\rho(a) \geq 1/3$. The client is said to be *poor* if $0 < \rho(a) < 1/3$. As it turns out, it is easy to handle two types of solutions: those having only rich nodes and those having only rich clients. For instance, in the first case, we can simply open all the rich nodes and obtain an integrally open solution (at a three factor loss in approximation). So, the main challenge lies in handling solutions wherein both poor nodes and poor clients are present.

Our algorithm would address the issue by applying a sequence of transformations to the given solution. The assignments from clients to nodes can be classified into four types, based on whether a client is rich or poor, and whether a node is rich or poor. Our first transformation removes one of these assignments, namely the assignments from poor clients to poor nodes. Towards that goal we next define the notion of *feudal solutions*. A solution σ is said to be *feudal*, if poor nodes service only rich clients. See Figure 1(a). The first transformation is stated below.

▶ **Lemma 3.** *Any LP solution σ_{in} can be converted into a feudal solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 1 + 2\texttt{cost}(\sigma_{in})$.*

The natural next step would be to remove the assignments from the rich clients to the poor nodes, so that the poor nodes are eliminated altogether. However, we do not know how to achieve the above task in a direct manner. Instead, our next transformation will make progress towards that goal. Let $\sigma = \langle x, y \rangle$ be an LP solution. We say that a node u is *terminal*, if u is not fully-closed and all its descendants are fully-closed. The solution is said to be *hierarchical*, if every poor node is terminal and all the other nodes are either fully-open or fully-closed. See Figure 1(b). In the figure, the colored nodes are fully-open, the hatched nodes are terminal (poor) nodes and the white nodes are fully-closed. A direct implication of the hierarchical property is that any client will be serviced by at most one poor node. Thus, the set of clients serviced by any two poor nodes will be disjoint. Our next transformation is stated below.

▶ **Lemma 4.** *Any feudal solution σ_{in} can be converted into a hierarchical solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 11 \cdot \texttt{cost}(\sigma_{in})$.*

Our next transformation would convert hierarchical solutions into integrally serviced solutions, thereby taking us from the realm of partial covering to the realm of full covering. In doing so, the procedure would preserve the hierarchical property.

▶ **Lemma 5.** *Any hierarchical solution σ_{in} can be converted into a hierarchical integrally serviced solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 6 + 12 \cdot \texttt{cost}(\sigma_{in})$.*

The main idea behind the above lemma is as follows. Ignoring the fully-closed nodes, the input solution σ_{in} contains only fully-open nodes and terminal poor nodes. We consider each fully open node u and perform a shifting procedure on the clients serviced by u. Pick any two such clients a and b with $r(a) \leq r(b)$. We can decrease $x_{in}(a, u)$ by δ and increase $x_{in}(b, u)$ by δ (for a suitable δ). Via this method, we can construct a solution σ' such that the clients fall into two types: (i) fully-serviced clients, which will be serviced only by fully-open nodes; (ii) clients serviced by poor terminal nodes. In order to get a fully-serviced solution, we need to focus only on the second type of clients. We get an integrally serviced solution by exploiting the fact that the set of clients serviced by terminal nodes are disjoint.

Stage 2: Integrally Serviced Dual Assigned Solutions. Our next task is to obtain integrally serviced, dual assigned solutions. Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be a hierarchical integrally serviced solution, as output by Lemma 4. In σ_{in}, all the clients are fully-serviced or unserviced. Let Q be the set of fully-serviced clients. The nodes are of only three types: fully-open, fully-closed and poor. Let U be the set of all fully-open nodes and let P be the set of poor nodes. Any client $a \in Q$ may be serviced by multiple nodes from U, but it can be serviced by at most one node from P. We shall focus on U and Q and apply a "cycle cancellation" procedure to adjust the assignments of the clients to the nodes. We will not modify the extents to

which the nodes are open. This way we shall identify a subset of clients Q' and obtain a new solution $\sigma' = \langle x', y_{in} \rangle$ such that any client $a \in Q'$ is serviced by at most one from U. Thus, under σ', all the clients in Q' are dual-assigned (these will be serviced by at most one fully-open node and at most one poor node). We will ensure that the number of clients left out (namely $|Q| - |Q'|$) is at most $|U|$. We then fully open all the clients in $Q - Q'$ and obtain a dual-assigned solution σ_{out}. The cost of the solution σ_{out} will be at most $\texttt{cost}(\sigma_{in}) + |U|$, which is at most $2 \cdot \texttt{cost}(\sigma_{in})$. We do not know how to perform the above cycle cancellation operation over partially open nodes and so, we focus on the fully-open nodes. Via the above strategy, we will establish:

▶ **Lemma 6.** *Any hierarchical integrally serviced solution σ_{in} can be converted into an integrally serviced, dual assigned solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.*

Stage 3: Integrally Open, Integrally Serviced Solutions. Let σ_{in} be any integrally serviced, dual assigned solution. Our next task is to convert it into an integrally open, integrally serviced solution. Under σ_{in}, each client is either fully-serviced or unserviced. Let Q be the set of fully-serviced clients. Each client $a \in Q$ is serviced by at most two nodes. For the ease of exposition, assume that each client $a \in Q$ is serviced by exactly two nodes. We can construct a multi-graph by taking the nodes to be the vertices. Each client a serviced by two nodes u and v can be represented by a set of $r(a)$ parallel edges between u and v. The LP solution σ_{in} can be construed as an LP solution to the capacitated vertex cover problem on the above graph. Saha and Khuller [10] present a 34-approximation LP rounding procedure for the above problem. Using their procedure we can get an integrally open, integrally serviced solution.

▶ **Lemma 7.** *Any integrally serviced, dual assigned solution σ_{in} can be converted into an integrally open, integrally serviced solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 34 \cdot \texttt{cost}(\sigma_{in})$.*

Stage 4: Integral Solution. The final stage is to convert an integrally open, integrally serviced solution σ_{in} into an integral solution σ_{out}. The only issue with σ_{in} is that the request $r(a)$ of a client a may be split and assigned to multiple nodes. On the other hand, our problem definition requires that the request must be wholly assigned to a single node. We can address the issue by using a cycle cancellation procedure similar to that of Stage 2.

▶ **Lemma 8.** *Any integrally open, integrally serviced solution σ_{in} can be converted into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.*

Overall Algorithm: The overall algorithm is obtained by combining the procedures given by Lemmas 3 – 8. In this paper, we shall only prove Lemmas 3 , 4 and 6 and defer the other ones to the full version of the paper.

3 Feudal Solutions: Proof of Lemma 3

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the given solution. Create a pool $\mathcal{P}$ consisting of all poor nodes. The procedure is iterative and it works in multiple phases. Each phase would modify the current solution into a new solution, which is fed as input to the subsequent phase. To start with, we take the input solution to be the current solution. In each phase, we shall remove one or more nodes from the pool. For each such node u, we will ensure that u is either rich or it services only rich clients (or both). Each phase proceeds as follows.

Let $\pi_{old} = \langle x, y \rangle$ be the current solution. With respect to π_{old}, let H be the rich clients and L be the poor clients. For each node $u \in \mathcal{P}$, compute the ratio:

$$\lambda(u) = \frac{1}{y_{old}(u)} \sum_{a \in \mathtt{Att}(u) \cap L} x_{old}(a, u).$$

Informally, the numerator signifies the contribution towards the partiality parameter K (i.e., "gain") given by the poor clients, and the denominator signifies the cost of the node. Higher the ratio, the node is better. Let u^* be the node in $\mathcal{P}$ having the maximum ratio $\lambda(\cdot)$; we call the node as the *leader* of the phase. The goal of the current phase is to convert u^* into a rich node.

Excluding u^*, arrange all the other nodes in $\mathcal{P}$ in an arbitrary order, say $u_1, u_2, \ldots, u_\ell$. Let s be the least index such that

$$y_{old}(u^*) + \sum_{j=1}^{s} y_{old}(u_j) \geq 1/3.$$

If no such index exists (meaning $y_{old}(u^*) + \sum_{j=1}^{\ell} y_{old}(u_j) < 1/3$), set $s = \ell$. We call the nodes $u_1, u_2, \ldots, u_s$ as the *slaves* of u^*.

Let $\delta = \sum_{j=1}^{s} y_{old}(u_j)$. Construct a new solution $\pi_{new} = \langle x_{new}, y_{new} \rangle$ as follows. First we eliminate all the assignments of poor clients onto the slaves. Namely, for each slave u_j and all clients $a \in \mathtt{Att}(u_j) \cap L$, set $x_{new}(a, u_j) = 0$. This would reduce the partiality value (i.e., LHS of constraint (2)) and hence, the constraint would be violated. To compensate, we increase the assignments of the poor clients to the leader as follows. For each client $a \in \mathtt{Att}(u^*) \cap L$, set

$$x_{new}(a, u^*) = x_{old}(a, u^*) \frac{(y_{old}(u^*) + \delta)}{y_{old}(u^*)}.$$

Using the fact that $\lambda(u_j) \leq \lambda(u^*)$, we can show that the loss in the partiality value is compensated by the above gain. The above increase in the assignments at node u^* may lead to violation of constraints (1) or (3) at node u^*. To overcome the issue, we increase the extent to which u^* is open. Set

$$y_{new}(u^*) = y_{old}(u^*) + \delta.$$

All the other values of $y_{new}(\cdot)$ and $x_{new}(\cdot, \cdot)$ are retained as in π_{old}.

▶ **Lemma 9.** *π_{new} is a feasible solution.*

The above claim is proved via induction by assuming that π_{old} is feasible and then showing that π_{new} is also feasible. The fact that the clients are classified into rich and poor based on the threshold of 1/3 is used to show that no client gets serviced to an extent of more than one. The proof is deferred to the full version of the paper.

At the end of the phase we remove the leader and the slaves from the pool. If the pool is non-empty, we proceed to the next phase and pick a new leader. The solution constructed π_{new} is taken as the current solution. On the other hand, the pool may be empty (this will happen when all the nodes were picked as slaves, i.e., $s = \ell$). In this case, we refer to the leader as the *loner*. We open loner fully and terminate the procedure.

Let us now analyze the solution output by the process. Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the solution input to the procedure and let $\sigma_{out} = \langle x_{out}, y_{out} \rangle$ be the solution output by the procedure. Let R and P be the set of all nodes which are rich and poor with respect to σ_{in},

respectively. The extent to which the nodes in R are open was left undisturbed and so, these remain rich with respect to σ_{out}. The nodes in P were added to the pool and all of them got removed as either leaders or as slaves. We ensured that all the leaders are rich when they left the pool. Regarding the slaves, we ensured that all the poor clients got their assignments to these nodes eliminated. It follows that the output solution is feudal.

Let us analyze the cost of the solution σ_{out}. We did not modify the extent to which the nodes in R and the slaves are open. So, we need to consider only the leaders. Except the loner, consider any leader u. Let $\delta(u)$ be the increase in its extent of opening; $\delta(u)$ is the sum of extents to which the slaves of u are open under σ_{in}. A node can serve as a slave for at most one node. Therefore,

$$\texttt{cost}(\sigma_{out}) \leq \texttt{cost}(\sigma_{in}) + \sum_{u \in S} y_{in}(u),$$

where S is the set of all slaves. The sum in the RHS is at most $\texttt{cost}(\sigma_{in})$. Hence, we get that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$. We need to add an extra cost of one for opening the loner. This completes the proof of Lemma 3.

4 Hierarchical Solutions: Proof of Lemma 4

The lemma is proved in two stages. The first stage converts the input feudal solution into a sandwich solution, defined next. A solution σ is said to be a *sandwich solution* if for any two poor nodes u_1 and u_2 such that u_1 is an ancestor of u_2, there exists a rich node v in between the path connecting u_1 and u_2 in the given tree. We first prove the following lemma.

▶ **Lemma 10.** *Any feudal solution σ_{in} can be converted into a sandwich solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 5 \cdot \texttt{cost}(\sigma_{in})$.*

The sandwich property can be equivalently restated as follows. For a node u, let v be the closest ancestor which is not fully-closed; we call v as the *least non-trivial ancestor* of u. We note that the above definition does not apply to the root node and any node u whose path to the root (including root) consists only of fully-closed nodes. A solution satisfies the sandwich property if for any poor node u having a least non-trivial ancestor v, the node v is rich.

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the given solution. We process the nodes from leaf level to the root level; namely, a node will be processed once all its children are processed. The processing of each node takes the current solution and produces a new solution.

The processing of a node u^* is as follows. First consider some easy cases: (i) u^* is fully-closed; (ii) u^* is rich; (iii) u^* does not have a least non-trivial ancestor; (iv) u^* has a rich least non-trivial ancestor v^*. In these cases we do nothing.

The case left remaining is where both u^* and its least non-trivial ancestor v^* are poor. Let p^* be the parent of u^* (this could be the same as v^*). Let $\pi_{old} = \langle x_{old}, y_{old} \rangle$ be the current solution and we will construct a new solution $\pi_{new} = \langle x_{new}, y_{new} \rangle$. We say that a client a is *critical* at u^* if u^* services a and u^* is the highest node that a can be attached (i.e., $d(a, p^*) > d_{\max}(a)$ and so, $a \notin \texttt{Att}(p^*)$). We consider two cases based on whether there exists a critical client at u^*. If there is such no such client, we perform the following *merge operation*. The idea is to close u^* fully and shift its extent of opening and all its client assignments to the parent p^*. Compute the new solution by setting $y_{new}(u^*) = 0$ and $y_{new}(p^*) = y_{old}(p^*) + y_{old}(u^*)$. For each client a serviced by u^*, set $x_{new}(a, p^*) = x_{old}(a, p^*) + x_{old}(a, u^*)$ and set $x_{new}(a, u^*) = 0$. All other entries of the functions y_{new} and x_{new} are retained as in π_{old}. It can be verified that π_{new} is feasible.

Consider the more interesting case where some client a^* is critical at u^*. In this case, we cannot perform the above merge operation. We simply open the node u^* and set $y_{new}(u^*) = 1$. We next perform a *pull procedure* as follows. The procedure is iterative. Let $\pi_{old} = \langle x_{old}, y_{old} \rangle$ be the current solution. Consider any client a such that a is serviced by some ancestor v of u^*. We will pull the assignment of a from v and reassign it to u^*. In doing so, we must ensure that the capacity constraint at u^* does not get violated. Let the current capacity utilization at u^* be $\texttt{cap}(u^*) = \sum_{a' \in \texttt{Att}(u^*)} r(a') x_{old}(a', u^*)$. Compute

$$\delta = \min \left\{ x(a, v), \frac{W - \texttt{cap}(u^*)}{r(a)} \right\}.$$

Set $x_{new}(a, v) = x_{new}(a, v) - \delta$ and $x_{new}(a, u^*) = x_{old}(a, u^*) + \delta$. The above operation is performed iteratively until we can find no such client a or the capacity utilization at u^* becomes W. If the procedure terminates under the first criterion, we call u^* a *critical node*; if it terminated under the second criterion, we call u^* a *complete node*. This completes the description of the procedure.

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the input solution and $\sigma' = \langle x', y' \rangle$ be the solution output by the procedure. It is easy to see that σ' satisfies the sandwich property. We compare the cost of σ' and σ_{in}. The merge operation preserves the cost and so, we need to bother only about the critical and complete nodes. Let n_C be the number of critical nodes and n_W the number of complete nodes. We have that

$$\texttt{cost}(\sigma') \leq \texttt{cost}(\sigma_{in}) + n_C + n_W.$$

For a client a, let $\theta_{in}(a) = \sum_u x_{in}(a, u)$ and $\theta'(a) = \sum_u x'(a, u)$ be the extent to which a is serviced under σ_{in} and σ', respectively. The above procedure does not change the extent of service for any client and so $\theta_{in}(a) = \theta'(a)$. Thus, the total volume of service extended by the two solutions is the same:

$$\sum_{a \in A} r(a) \theta_{in}(a) = \sum_{a \in A} r(a) \theta'(a).$$

Clearly, the quantity $n_W \cdot W$ is at most the RHS, because the n_W complete nodes have capacity utilization W. Moreover, $\texttt{cost}(\sigma_{in})$ is at least LHS$/W$ since a node has only capacity W. It follows that $n_W \leq \texttt{cost}(\sigma_{in})$.

We now consider the quantity n_C. Let $u_1, u_2, \ldots, u_s$ be the critical nodes, where $s = n_C$. For each $1 \leq j \leq s$, u_j has a critical client a_j under σ'. Under σ_{in}, either the poor node u_j services a_j or some poor node merged with u_j services it. Either way a_j is serviced by some poor node under σ_{in}. Since σ_{in} is a feudal solution, $\theta_{in}(a_j) \geq 1/3$. Let p_j be the path connecting u_j and leaf node a_j in the tree. Let $\mu(a_j)$ the sum of extents to which the nodes found along the above path are open in σ_{in}, i.e., $\sum_{u \in p_j} y_{old}(u)$. By the constraint (3), we have that $\mu(a_j) \geq \theta_{in}(a_j)$ and hence, $\mu(a_j) \geq 1/3$. The pulling procedure ensures that for any a_i and a_j, the paths p_i and p_j are disjoint. Therefore,

$$\texttt{cost}(\sigma_{in}) \geq \sum_{j=1}^{s} \mu(a_j) \geq s/3.$$

This shows that $n_C \leq 3\texttt{cost}(\sigma_{in})$. Hence, we get that $\texttt{cost}(\sigma') \leq 5 \cdot \texttt{cost}(\sigma_{in})$.

We now transform the sandwich solution σ' into a hierarchical solution σ_{out}. With respect to $\sigma' = \langle x', y' \rangle$, let R be the set of rich nodes and let P be the set of poor nodes. Of the poor nodes, let P_1 be the set of terminal nodes and let P_2 be the other nodes. The sandwich property

implies that $|P_2| \leq |R|$. We fully open all the nodes in R and P_2, and obtain a hierarchical solution σ_{out}. We see that $\texttt{cost}(\sigma') \geq |R|/3$ and that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot |R| + \texttt{cost}(\sigma')$. Therefore, $\texttt{cost}(\sigma_{out}) \leq 7 \cdot \texttt{cost}(\sigma')$. This shows that $\texttt{cost}(\sigma_{out}) \leq 35 \cdot \texttt{cost}(\sigma_{in})$ However, by combining the analysis of the two stages, we can derive an improved bound: $\texttt{cost}(\sigma_{out}) \leq 11 \cdot \texttt{cost}(\sigma_{in})$ (details are deferred to the full version of the paper).

5 Proof of Lemma 6

Let σ_{in} be the input hierarchical integrally serviced solution. In σ_{in}, all the clients are fully-serviced or unserviced, and the nodes are of only three type: fully-open, fully-closed and poor. Each client can be serviced by at most one poor node. We will adjust the assignments of the clients to the fully-open nodes and obtain a new solution wherein most of the clients are serviced by at most one fully-open node. The following client-server setup is useful for this purpose.

Consider a bipartite graph with a set of servers U on one side and a set of clients Q on the other side. For a client a, all its neighboring servers are said to be *accessible* to a. Each client has an integral requirement $q(a)$ and each server has an integral capacity W. An edge-assignment g over a subset of clients $Q' \subseteq Q$ is a mapping that takes as input a client $a \in Q'$ and node u accessible to it and outputs a value $g(a,u) \in [0, q(a)]$ (intuitively, it assigns $g(a,u)$ part of the request $q(a)$ to u). We require that for any client a, $\sum_u g(a,u) = q(a)$ and for any node u, $\sum_a g(a,u) \leq W$. We allow $q(a)$ and $g(a,u)$ to be non-integral. We say that an edge-assignment is a *single policy assignment*, if for any client $a \in Q$, all its request is assigned to a single accessible server. The following lemma shows how to convert any assignment into a single policy assignment with only a minimal loss.

▶ **Lemma 11.** *There exists a polynomial time procedure that takes any edge-assignment g over Q as input and outputs a subset of clients Q' and an edge-assignment g' over Q' such that: (i) g' is a single policy edge-assignment over Q'; (ii) the number of ignored clients is at most U, i.e., $|Q| - |Q'| \leq |U|$.*

Proof. Let $G = (V, E)$ be the weighted bipartite graph where $V = U \cup Q$ and E is the set of edges between clients and their neighboring servers. The weight on an edge $(a,u) \in E$ is defined to be the assignment $g(a,u)$. Let G^+ be the graph induced on G by considering only the edges with strictly positive weights, i.e., corresponding to which the assignments g are non-trivial ($g(a,u) > 0$).

We first show that it is possible to eliminate cycles from the induced graph G^+. The main idea is to employ cycle cancelling. Consider any cycle, C, in G^+. Since G is a bipartite graph, the cycle is of even length. We partition the edges of the cycle C into two sets C_{even} and C_{odd} by considering alternate edges in each set. We now modify the weights as follows. Let δ be the weight of the minimum weight edge in the cycle. For every edge, (a,u) in C_{even}, we increase $g(a,u)$ by an amount $\delta/q(a)$ and for every edge (a,u) in C_{odd}, we decrease $g(a,u)$ by an amount $\delta/q(a)$. This breaks the cycle. We apply this procedure repeatedly until all the cycles are eliminated.

We next analyze the forest G^+. Let $Q' = \{a \in Q \cap G^+ \ : \ degree(a) = 1 \text{ in } G^+\}$. Note that all these clients are already fully assigned. Now consider the forest $\widetilde{G^+}$ induced by the vertices of $U \cup (Q \setminus Q')$ on G^+. Consider any tree H in the forest. Pick an arbitrary vertex $r \in U \cap H$ and make it the root of the tree. For any vertex v of H, let $\ell(v)$ denote the level (distance from the root r) of the vertex v in the tree. Since G^+ is a bipartite graph, we note that the vertices at even level (odd level resp.) belong to $U \cap H$ ($Q \cap H$ resp.). Note that

all the leaves in the tree are in U. We now construct a one-to-one mapping from $Q \cap H$ to $U \cap H$. This is obtained by mapping every element of Q in the tree to any one of its children arbitrarily. This shows that $|Q \cap H| \leq |U \cap H|$. We repeat the above procedure for every tree in the forest $\widetilde{G^+}$. The one-to-one mapping gives us that $|Q| - |Q'| \leq |U|$. This completes the proof of the lemma. ◀

We can now prove Lemma 6 using the above client-server setup. Consider the input solution $\sigma_{in} = \langle x_{in}, y_{in} \rangle$. Let U be the set of all fully-open nodes in σ_{in} and let Q be the set of fully-serviced clients. For each $a \in Q$, let $q(a)$ be the portion of the request of a serviced by the nodes in U; let $q(a) = \sum_{u \in U \cap \texttt{Att}(a)} r(a) \cdot x_{in}(a, u)$. Construct an edge-assignment g: for each client $a \in Q$ and $u \in U \cap \texttt{Att}(a)$, let $g(a, u) = x_{in}(a, u)r(a)$. Invoke Lemma 11 and obtain a set of clients $Q' \subseteq Q$ and an edge-assignment g' over Q'. Using g' we can construct a new solution $\sigma_{out} = \langle x_{out}, y_{out} \rangle$ such that all the clients in Q' are serviced by at most one fully-open node. All these clients are dual assigned. As far as the clients in $Q - Q'$ are concerned, we simply fully open the leaf nodes corresponding to these clients. The new solution σ_{out} will have cost at most $\texttt{cost}(\sigma_{in}) + |Q| - |Q'|$. The above lemma guarantees that $|Q| - |Q'| \leq |U|$. Since $\texttt{cost}(\sigma_{in}) \geq |U|$, we have that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.

References

1. A. Benoit, H. Larchevêque, and P. Renaud-Goud. Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks. In *IPDPS*, pages 1022–1033, 2012.
2. A. Benoit, V. Rehn-Sonigo, and Y. Robert. Replica placement and access policies in tree networks. *IEEE Trans. on Parallel and Dist. Systems*, 19:1614–1627, 2008.
3. J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM Journal Computing*, 36(2):498–515, 2006.
4. I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40:205–218, 2002.
5. U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
6. R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1), 2006.
7. K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Trans. on Parallel and Distributed Systems*, 12:628–637, 2001.
8. M.J. Kao and C.S. Liao. Capacitated domination problem. *Algorithmica*, pages 1–27, 2009.
9. Y.F. Lin, P. Liu, and J.J. Wu. Optimal placement of replicas in data grid environments with locality assurance. In *ICPADS*, 2006.
10. B. Saha and S. Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *ICALP*, 2012.
11. L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.
12. J.J. Wu, Y.F. Lin, and P. Liu. Optimal replica placement in hierarchical data grids with locality assurance. *J. of Parallel and Dist. Computing*, 68:1517–1538, 2008.