Alcom-FT Technical Report Series
ALCOMFT-TR-01-71

Approximation Algorithms for Data Management in Networks*

Christof Krick
Heinz Nixdorf Institute and
Department of Mathematics
& Computer Science
Paderborn University
Germany

Harald Räcke
Heinz Nixdorf Institute and
Department of Mathematics
& Computer Science
Paderborn University
Germany

Matthias Westermann
Heinz Nixdorf Institute and
Department of Mathematics
& Computer Science
Paderborn University
Germany

Abstract

This paper deals with static data management in computer systems connected by networks. A basic functionality in these systems is the interactive use of shared data objects that can be accessed from each computer in the system. Examples for these objects are files in distributed file systems, cache lines in virtual shared memory systems, or pages in the WWW. In the static scenario we are given read and write request frequencies for each computer-object pair. The goal is to calculate a placement of the objects to the memory modules, possibly with redundancy, such that a given cost function is minimized.

With the widespread use of commercial networks, as, e.g., the Internet, it is more and more important to consider commercial factors within data management strategies. The goal in previous work was to utilize the available resources, especially the bandwidth, as good as possible. We will present data management strategies for a model in which commercial cost instead of the communication cost are minimized, i.e., we are given a metric communication cost function and a storage cost function.

We introduce new deterministic algorithms for the static data management problem on trees and arbitrary networks. Our algorithms aim to minimize the total cost. To our knowledge this is the first analytic treatment of this problem that is NP-hard on arbitrary networks. Our main result is a combinatorial algorithm that calculates a constant factor approximation for arbitrary networks in polynomial time. Further, we present an algorithm for trees that calculates an optimal placement of all objects in X on a tree $T = (V, E)$ in time $O(|X| \cdot |V| \cdot \mathrm{diam}(T) \cdot \log(\deg(T)))$.

*Partially supported by the DFG-Sonderforschungsbereich 376 and the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT). To appear in *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001.

1 Introduction

In recent years, large computer systems connected by networks have become part of our everyday live. A good example is the widespread use of the Internet and Internet-related applications such as the World Wide Web (WWW). A basic functionality in these systems is the interactive use of shared data objects that can be accessed from each computer in the system. Examples for these objects are files in distributed file systems for Ethernet-connected workstations, cache lines in virtual shared memory systems for massively parallel computers, or pages in the WWW.

The dramatic growth of computer systems necessitates more and more an intelligent management of shared data objects. With the widespread use of commercial networks, as, e.g., the Internet, it is more and more important to consider commercial factors within data management strategies. The goal in previous work was to utilize the available resources, especially the bandwidth, as good as possible. In such a scenario it is usually assumed that the use of the resources is for free. We will present data management strategies for a model in which commercial cost instead of the communication cost are minimized.

Our model mirrors, e.g., the perspective of a content provider that offers information via pages in the WWW. For that purpose, the content provider has to rent or buy some amount of the resources bandwidth and memory. We assume that there is a fee per transmitted byte for each communication link and a fee per stored byte for each memory module in the network. Then, the total cost of the content provider is charged in dependency of the amount of bytes that are sent along communication links or stored in memory modules.

In this cost based model we will present static data management strategies. In the static scenario we are given read and write request frequencies for each computer-object pair. A data management strategy has to answer the following questions.

- How many copies of a shared data object should be created?
- On which memory modules should these copies be placed?
- How should the read and write requests be served?

The goal is to calculate a placement of the objects to the memory modules, possibly with redundancy, such that the total cost are minimized.

In general, the goal in previous work was to minimize the total communication load (see, e.g., [1, 2, 4, 9]) which is defined as follows: The *communication load* of a link is the amount of data transmitted by this connection. The *total communication load* is the sum over the load of all links divided by the bandwidth of the respective link. Obviously, a data management strategy, that is provably good in our cost based model, minimizes also the total communication load, if the fee of each link is equivalent to the reciprocal value of the bandwidth of the respective link, and if the fee of each memory module is 0. Thus, our cost based model is a generalization of the total load model.

1.1 The cost based model

The computer system is modeled by an undirected graph $G = (V, E)$ with node set V and edge set E such that the nodes represent the processors with their memory modules, and the edges represent the links. The *cost per stored data object* for the memory modules are described by a function $c_s : V \longmapsto \mathbb{R}_0^+$ and the *cost per transmitted data object* for the links are described by a function $c_t : E \longmapsto \mathbb{R}_0^+$. For simplicity, we assume that all the data objects have uniform size. Thus, the functions c_s and c_t do not depend on the data objects. However, all our results hold also in a non-uniform model.

Let $c_t(v, v')$ denote the *cost per transmitted data object* from a node v to a node v'. We define $c_t(v, v') := \min_{\text{path } p \text{ from } v \text{ to } v'} \left\{ \sum_{\text{edge } e \text{ lies on } p} c_t(e) \right\}$. Then, the function c_t defines a *metric space* over the node set V, since c_t is non-negative, symmetric, and satisfies the triangle inequality. Thus, c_t can also be seen as a distance function. To keep our algorithms and proofs simple and clear we often use this view of c_t.

The *static data management problem* is defined as follows (compare [10]). We are given a *set X of shared data objects* and the *read and write request frequencies* for each node-object pair which are described by the functions $f_r : V \times X \longmapsto \mathbb{N}$ and $f_w : V \times X \longmapsto \mathbb{N}$, respectively. For each object $x \in X$, we have to determine a set of nodes holding copies of x. Then, it remains to specify how each request r for x will be served. The node issuing r is denoted the *home $h(r)$ of the request r*.

- In case of a read request r, $h(r)$ simply reads the nearest copy of x. The node holding this copy is denoted the *node $s(r)$ that serves the request r*. Note that for a given set of nodes holding copies of x a read request is always served with optimal cost.

- In case of a write request r, an update is sent from $h(r)$ to all copies of x. Thus, we have to determine the edges along which this update is sent. This is modeled by a multi-set of edges E_{U_r} which is denoted the *update set of request r*. Note that some edges in this multi-set can induce a multi-cast tree that branches at arbitrary nodes and that some edges can appear several times in this multi-set. For technical reasons, the node holding the copy of x that is nearest to $h(r)$ is denoted the *node $s(r)$ that serves the request r*.

The goal in the *cost based model* is to calculate a placement of the objects to the memory modules such that the total cost are minimized. The *total cost* are defined as follows.

- A copy of object x on the node v increases the total cost by $c_s(v)$.

- A read request r for object x increases the total cost by $c_t(h(r), s(r))$.

- A write request r for object x increases the total cost by $\sum_{e \in E_{U_r}} E_{U_r}(e) \cdot c_t(e)$, where $E_{U_r}(e)$ denotes the number of appearances of edge e in the multi-set E_{U_r}.

The model described above is slightly restrictive in the sense that it fixes the update policy to a certain range. In particular, it does not include strategies that allow only a fraction of the copies to be updated in case of a write, which, e.g., is implemented in strategies using the majority trick introduced in [14]. However, all strategies using such techniques add time stamps to the copies. This requires that there is some definition of uniform time among different nodes. Since it is not clear how to realize this in an asynchronous setting, we restrict ourselves to strategies that update or invalidate all copies in case of a write.

1.2 Previous and related work

The first approaches to solve static data management problems concentrate on modeling it by mixed integer programs and solving these programs efficiently by using heuristics. Here several models with different cost functions and constraints have been developed. The survey paper by Dowdy and Foster [7] gives an overview of this work.

Recently and independently, Baev and Rajaraman [3] consider static data management in arbitrary networks with memory capacity constraints in a model that is similar to our model. The major difference is that they only consider read requests. They present a constant factor approximation algorithm that aims to minimize the total cost. This algorithm is based on solving a linear programming relaxation of the problem and rounding the obtained solution.

Milo and Wolfson [15] show that the static data management problem in arbitrary networks is already NP-hard in the total communication load model. Further, they present polynomial time algorithms for completely-connected networks, trees, and rings that calculate optimal placements in the total communication load model.

Maggs et. al. [10] present static and dynamic data management strategies for trees, meshes, and Internet-like clustered networks. All these strategies aim to minimize the congestion, i.e., the maximum over the load of all edges divided by the bandwidth of the respective edge.

They present a deterministic strategy for static data management on trees that minimizes the communication load on all edges simultaneously. Obviously, this yields minimum congestion as well as minimum total communication load. Further, they present optimal or close-to-optimal strategies that aim to minimize the congestion on meshes and Internet-like clustered networks. For example, the strategy for d-dimensional meshes with n nodes achieves optimal congestion up to a factor of $O(d \cdot \log n)$. The sequential running time of these algorithms is linear in the input size. Moreover, they show that the static problem is already NP-hard on a 3×3 mesh in the congestion model.

In the dynamic setting, no knowledge about the request pattern is assumed. An adversary specifies requests at run-time. Here, they present optimal or close-to-optimal strategies that also aim to minimize the congestion on trees, meshes, and Internet-like clustered networks. These strategies are investigated in a competitive model. Meyer auf der Heide et. al. [11, 12] extend these results in the dynamic setting to non-uniform cost models with different communication costs for accessing and migrating data objects, to networks with memory capacity constraints, and other classes of networks.

Awerbuch et. al. [1, 2] consider the dynamic data management problem in arbitrary networks. In [1], they present a centralized strategy that achieves an optimal competitive ratio $O(\log n)$ and a distributed strategy that achieves competitive ratio $O((\log n)^4)$ on an arbitrary network with n nodes. In [2], the distributed strategy is adapted to networks with memory capacity constraints. All competitive ratios are with respect to the total communication load.

The static data management problem in our cost based model is similar to the facility location problem. The major difference is that in the facility location problem there exists nothing comparable to the write requests in the data management problem. Aardal et. al. [13] gave the first constant factor approximation algorithm for the facility location problem. This was subsequently improved by Chudak and Shmoys [5, 6] who achieved the currently best known approximation ratio of $1 + 2/e \approx 1.736$. All these algorithms are based on solving linear programming relaxations of the facility location problem and rounding the obtained solutions. Korupolu et. al. [8] analyzed a well known local search heuristic and showed that it achieves an approximation factor of $5 + \epsilon$ with $\epsilon > 0$.

1.3 New results

We introduce new deterministic algorithms for the static data management problem on trees and arbitrary networks. Our algorithms aim to minimize the total cost in our cost based model. To our knowledge this is the first analytic treatment of this problem. Our main result, presented in Section 2, is a combinatorial algorithm that calculates a constant factor approximation for arbitrary graphs in polynomial time. Further, we present in Section 3 an algorithm for trees that calculates an optimal placement of all objects in X on a tree $T = (V, E)$ in time $O(|X| \cdot |V| \cdot \mathrm{diam}(T) \cdot \log(\deg(T)))$, where $\mathrm{diam}(T)$ denotes the unweighted *diameter* of T, i.e., the maximum number of edges on a path connecting two arbitrary nodes in T, and $\deg(T)$ denotes its maximum node *degree*.

2 The approximation algorithm for arbitrary networks

In this section, we present a combinatorial algorithm that calculates a constant factor approximation for the static data management problem on arbitrary graphs. Our algorithm places all objects independently from each other. Thus, fix an object x.

The focus in our algorithm lies on the calculation of a good placement for object x. Given such a placement write accesses are handled as follows. A node that issues a write request r for x first sends a message to the closest node holding a copy, i.e., $s(r)$. Then an update of all copies of x via a minimum spanning tree is initiated, i.e., $s(r)$ sends out one message that is transmitted along the branches of a minimum spanning tree that connects all nodes holding copies of x. Thus, the update set E_{U_r} contains all edges on the shortest path between $h(r)$ and $s(r)$ and all edges of the minimum spanning tree. Note that edges on the path from $h(r)$ and $s(r)$ can be contained twice in the multi-set E_{U_r}.

In order to show the quality of our solution we will compare it to an optimal solution that fulfills the following additional constraints.

1. A write request r for x first sends a message to $s(r)$ and then initiates the update of all copies of x via a multi-cast tree. All write requests for x use the same multi-cast tree T_x.
2. Each copy of x serves at least W requests, with $W = \sum_{v \in V} f_w(v)$ denoting the total number of write requests for x.

A placement fulfilling these constraints is called *restricted*. The following lemma ensures that there exists a restricted placement that has nearly optimal cost for reading, storing and updating global objects.

Lemma 1 *Let OPT and OPT_W denote an optimal and optimal restricted placement, respectively. Then*

$$C^{\mathrm{OPT}_W} \leq 4C^{\mathrm{OPT}} \ ,$$

where C^{OPT_W} and C^{OPT} denote the total cost of OPT_W and OPT, respectively.

Proof. Suppose the optimum placement OPT is given together with the optimal update sets of all requests. We will successively transform this placement into a restricted placement as follows. First, we replace each update set $E_{U_r}^{\mathrm{OPT}}$ of the optimum placement by the edge set of a minimum spanning tree (MST) connecting all copies plus the edges on the path from $h(r)$ to $s(r)$. The following claim shows that the total cost of the placement is only doubled by this transformation.

Claim 2 *Let OPT' denote the new placement and $E_{U_r}^{\mathrm{OPT}'}$ its update set for request r. Then*

$$\sum_{e \in E_{U_r}^{\mathrm{OPT}'}} E_{U_r}^{\mathrm{OPT}'}(e) \cdot c_t(e) \leq 2 \sum_{e \in E_{U_r}^{\mathrm{OPT}}} E_{U_r}^{\mathrm{OPT}}(e) \cdot c_t(e) \ .$$

Proof. Let E_{ST} and E_{MST} denote the edge set of a minimum Steiner tree and a minimum spanning tree, respectively, connecting $h(r)$ with all nodes holding a copy of x. Then $\sum_{e \in E_{\mathrm{MST}}} c_t(e) \leq 2 \sum_{e \in E_{\mathrm{ST}}} c_t(e) - \sum_{e \in E_{\mathrm{P}}} c_t(e)$, where E_{P} denotes the edge set containing all edges of an arbitrary path P of the Steiner tree. By choosing the path from $h(r)$ to $s(r)$ for P we get

$$\begin{aligned}\sum_{e\in E_{U_r}^{\mathrm{OPT}'}} E_{U_r}^{\mathrm{OPT}'}(e)\cdot c_t(e) &\leq \sum_{e\in E_{\mathrm{MST}}} c_t(e) + \sum_{e\in E_{\mathrm{P}}} c_t(e)\\ &\leq 2\sum_{e\in E_{\mathrm{ST}}} c_t(e) = 2\sum_{e\in E_{U_r}^{\mathrm{OPT}}} E_{U_r}^{\mathrm{OPT}}(e)\cdot c_t(e)\ \ ,\end{aligned}$$

which yields the claim. ■

In a second step the placement OPT' is transformed into a restricted placement OPT_W as follows.

> As long as the set $\mathcal{C}_{\leq W}$ of copies that do not serve at least W requests is not empty, delete the copy $c \in \mathcal{C}_{\leq W}$ with maximum tree distance from the root node of the MST, which is rooted at an arbitrary node. (The tree distance between a node u and v is the length of the unique path that connects u and v via edges of the MST.)
> After the deletion of c, each request previously assigned to c is reassigned to its nearest remaining copy in the network.

This algorithm terminates because the number of requests is larger than W and thus the last copy will not be deleted. Obviously, the resulting placement is *restricted* because each remaining copy serves at least W requests. It remains to show that the additional cost that incur by reassigning requests to other copies is small.

Each request that is reassigned from a copy on node v to another copy on node v' increases the total cost by $c_t(h(r),v') - c_t(h(r),v)$. Note that this holds for write requests as well, because the algorithm has to maintain the first constraint of a restricted placement, i.e., the cost for a message from $h(r)$ to v has to be taken into account.

Let v_f denote the father node of v according to the tree structure given by the MST. Each reassignment increases the cost by

$$\begin{aligned}c_t(h(r),v') - c_t(h(r),v) &\leq c_t(h(r),v_f) - c_t(h(r),v)\\ &\leq c_t(v,v_f)\ \ .\end{aligned}$$

The first step holds since $c_t(h(r),v') \leq c_t(h(r),v_f)$. v' holds the copy closest to $h(r)$, but v_f still holds a copy, because copies closer to the root node are deleted later. The second step holds because of the triangle inequality.

At most W requests are reassigned in a deletion step because otherwise the copy would not have to be deleted since it would have served more than W requests. The total cost-increment caused by a deletion on node v is therefore at most $W \cdot c_t(v,v_f)$. Summing this over all nodes holding a copy yields that the total cost-increment does not exceed the cost expended by OPT' for updating objects. Together with Claim 2 this yields the lemma. ■

In the remainder of this section we only consider restricted placements. We split the total cost of such a placement into read, update, and storage cost which are defined as follows. For a given placement P, the storage cost C_s^P are defined as $C_s^P := \sum_{v\in V \text{ holding a copy}} c_s(v)$, the update cost C_u^P are defined as $C_u^P := W \cdot \sum_{e\in E_{T_x}} c_t(e)$, with E_{T_x} denoting the edge set of the multi-cast tree connecting all copies of x and $W = \sum_{v\in V} f_w(v)$ denoting the total number of write requests for x, the read cost C_r^P are defined as $C_r^P = \sum_{\text{request } r \text{ for } x} c_t(h(r),s(r))$. Furthermore, $C_r^P(\mathcal{S}) := \sum_{r\in\mathcal{S}} c_t(h(r),s(r))$ denotes the read cost for a set $\mathcal{S}$ of requests.

Note that these definitions are only useful for restricted placements because the cost for a write request r (represented by the multi-set E_{U_r}) are partitioned into the cost for a multi-cast tree E_{T_x} and the cost for the path from $h(r)$ to $s(r)$. The latter cost are defined to belong to the read cost. By defining the read cost this way, we do not differentiate between read and write requests any more. The write requests are only represented by their total number W. Their exact location in the network does not influence the update cost. Obviously, for the total cost C^P of a restricted placement P holds $C^P = C_r^P + C_u^P + C_s^P$.

2.1 Proper placements

In this section, we define proper placements and prove some helpful properties of such proper placements. First, we introduce some notations and definitions. For a node v, let $\mathcal{R}_v^z$ denote the set of those z distinct requests that are closest to v. Further, let $d(v,z)$ denote the average distance between v and the requests in $\mathcal{R}_v^z$, i.e., $d(v,z) := \frac{1}{|\mathcal{R}_v^z|} \cdot \sum_{r \in \mathcal{R}_v^z} c_t(h(r), v)$.

For each node v, we define the *write radius* $r_w(v) := d(v, W)$, where W denotes the total number of write requests for x. Furthermore, we choose the *storage radius* $r_s(v) \in \mathbb{R}_0^+$ and the *storage number* $z_s(v) \in \mathbb{N}_0$ such that

$$\begin{aligned} (z_s(v) - 1) \cdot r_s(v) \ &\leq \ c_s(v) \ < \ z_s(v) \cdot r_s(v) \quad \text{and} \\ d(v, z_s(v) - 1) \ &\leq \ r_s(v) \ < \ d(v, z_s(v)) \ . \end{aligned}$$

This is done as follows. Obviously $z_s(v)$ can be chosen such that $(z_s(v) - 1) \cdot d(v, z_s(v) - 1) \leq c_s(v) < z_s(v) \cdot d(v, z_s(v))$. Then $r_s(v)$ is chosen from the interval $[d(v, z_s(v) - 1), d(v, z_s(v)))$ such that the first inequalities hold. The idea behind these definitions is that the write radius and the storage radius of a node v give an indication of the optimal distance from v to the nearest copy in the network.

This is formalized in the following. We call a placement *proper* if the copies are distributed according to the write and storage radii of the nodes as follows.

1. Every node v has a copy in distance at most $k_1 \cdot \max\{r_w(v), r_s(v)\}$, where k_1 denotes a suitable constant.

2. Two nodes u and v both holding a copy have at least distance $2k_2 \cdot \max\{r_w(u), r_w(v)\}$, where k_2 denotes a suitable constant.

In the remaining part of this section we will show that any *proper* placement guarantees a constant approximation factor for the read and update cost. In Section 2.2 we will present an algorithm that calculates a proper placement with low storage cost.

Theorem 3 *For the read and update cost of a proper placement* PRO *holds*

$$\begin{aligned} C_r^{\mathrm{PRO}} &\leq (k_1 + 1) \cdot (C_r^{\mathrm{OPT}_W} + C_s^{\mathrm{OPT}_W}) \quad \textit{and} \\ C_u^{\mathrm{PRO}} &\leq 2 \left(\frac{k_2}{k_2 - 1} \cdot (C_r^{\mathrm{PRO}} + C_r^{\mathrm{OPT}_W}) + C_u^{\mathrm{OPT}_W} \right) \ , \end{aligned}$$

where OPT_W *denotes an optimal restricted placement.*

Proof. First, the following lemma relates the read cost of a proper placement PRO to the read cost and storage cost of a optimal restricted placement OPT_W.

Lemma 4 *For the read cost C_r^{PRO} of a proper placement holds $C_r^{\mathrm{PRO}} \le (k_1+1)\cdot(C_r^{\mathrm{OPT}_W} + C_s^{\mathrm{OPT}_W})$.*

Proof. In order to show the lemma we compare the proper placement and the optimal restricted placement OPT_W. Suppose a copy is placed on node v in the optimal restricted placement OPT_W. Let $\mathcal{S}_v$ denote the set of requests served by this copy in OPT_W. We show that $C_r^{\mathrm{PRO}}(\mathcal{S}_v) \le (k_1+1)\cdot(C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + c_s(v))$. Then the lemma follows immediately by summing over all request sets.

Let $r \in \mathcal{S}_v$ denote a request issued at node $h(r)$ and served at node $s(r)$ in the proper placement. The distance between $h(r)$ and $s(r)$ can be estimated by

$$\begin{aligned} c_t(h(r),s(r)) &\le c_t(h(r),v') \\ &\le c_t(h(r),v) + c_t(v,v') \ , \end{aligned}$$

where v' denotes the node holding the copy nearest to v in the proper placement PRO. The first step uses the fact that $s(r)$ is the closest copy to $h(r)$ and the second step holds due to the triangle inequality.

Then the read cost $C_r^{\mathrm{PRO}}(\mathcal{S}_v)$ of the proper placement PRO can be bounded by

$$\begin{aligned} C_r^{\mathrm{PRO}}(\mathcal{S}_v) &= \sum_{r\in\mathcal{S}_v} c_t(h(r),s(r)) \\ &\le \sum_{r\in\mathcal{S}_v} c_t(h(r),v) + \sum_{r\in\mathcal{S}_v} c_t(v,v') \\ &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v|\cdot c_t(v,v') \\ &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v|\cdot k_1\cdot\max\{r_s(v),r_w(v)\} \ . \end{aligned}$$

Recall that $c_t(v,v') \le k_1\cdot\max\{r_s(v),r_w(v)\}$ due to the first property of a proper placement. Now, we distinguish two cases according to the maximum of $\{r_s(v),r_w(v)\}$.

- Suppose $r_w(v) = \max\{r_s(v),r_w(v)\}$.
  Obviously, $C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) \ge |\mathcal{S}_v|\cdot r_w(v)$, since the copy on node v serves $|\mathcal{S}_v| \ge W$ requests in the optimal restricted placement OPT_W. Then

$$\begin{aligned} C_r^{\mathrm{PRO}}(\mathcal{S}_v) &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v|\cdot k_1\cdot\max\{r_s(v),r_w(v)\} \\ &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + k_1\cdot|\mathcal{S}_v|\cdot r_w(v) \\ &\le (k_1+1)\cdot C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

- Suppose $r_s(v) = \max\{r_s(v),r_w(v)\}$.
  In this case we distinguish two sub-cases according to the cardinality of $\mathcal{S}_v$.
  - Suppose $|\mathcal{S}_v| < z_s(v)$. Then

$$\begin{aligned} C_r^{\mathrm{PRO}}(\mathcal{S}_v) &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v|\cdot k_1\cdot\max\{r_s(v),r_w(v)\} \\ &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + (z_s(v)-1)\cdot k_1\cdot r_s(v) \\ &\le C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + k_1\cdot c_s(v) \ . \end{aligned}$$

    Recall that $(z_s(v)-1)\cdot r_s(v) \le c_s(v)$ by the definition of $z_s(v)$ and $r_s(v)$.

– Suppose $|\mathcal{S}_v| \geq z_s(v)$. Then

$$\begin{aligned} C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) &\geq |\mathcal{S}_v| \cdot d(v, |\mathcal{S}_v|) \\ &\geq |\mathcal{S}_v| \cdot d(v, z_s(v)) \\ &\geq |\mathcal{S}_v| \cdot r_s(v) \ . \end{aligned}$$

Thus, for the read cost of the proper placement PRO holds

$$\begin{aligned} C_r^{\mathrm{PRO}}(\mathcal{S}_v) &\leq C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot \max\{r_s(v), r_w(v)\} \\ &\leq C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot r_s(v) \\ &\leq (k_1 + 1) \cdot C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

Altogether this yields the lemma. ■

Finally, the following lemma relates the write cost of a proper placement PRO to the cost of an optimal restricted placement OPT_W.

Lemma 5 *For the update cost of a proper placement* PRO *holds* $C_u^{\mathrm{PRO}} \leq 2(\frac{k_2}{k_2-1} \cdot (C_r^{\mathrm{PRO}} + C_r^{\mathrm{OPT}_W}) + C_u^{\mathrm{OPT}_W})$.

Proof. First, we prove the following claim showing that every copy in the proper placement PRO serves a certain number of requests.

Claim 6 *Every copy in a proper placement serves at least* $(1 - \frac{1}{k_2}) \cdot W$ *requests.*

Proof. In the proper placement PRO each request r with $c_t(h(r), v) \leq k_2 \cdot r_w(v)$ is served by the copy on v due to the second property of a proper placement. Now, the claim follows from a simple averaging argument.

Assume for contradiction that a copy on node v serves less than $(1 - \frac{1}{k_2}) \cdot W$ requests. Then at least $W - (1 - \frac{1}{k_2}) \cdot W = \frac{W}{k_2}$ requests in $\mathcal{R}_v^W$ have a distance to v larger than $k_2 \cdot r_w(v)$. This yields

$$r_w(v) = \frac{1}{|\mathcal{R}_v^W|} \cdot \sum_{r \in \mathcal{R}_v^W} c_t(h(r), v) > \frac{1}{W} \cdot \frac{W}{k_2} \cdot k_2 \cdot r_w(v) = r_w(v) \ ,$$

which is a contradiction. ■

Suppose that a node v holds a copy in the proper placement PRO and that the copy nearest to v in the optimum restricted placement OPT_W is placed on v'. For a request r that is served by v in the proper placement, let $s(r)$ denote the node serving r in OPT_W. Then

$$c_t(v, v') \leq c_t(v, s(r)) \leq c_t(v, h(r)) + c_t(h(r), s(r)) \ .$$

Let $\mathcal{S}_v$ denote the set of requests served by v in the proper placement. Summing the above inequality over all requests in $\mathcal{S}_v$ yields

$$\begin{aligned} |\mathcal{S}_v| \cdot c_t(v, v') &= \sum_{r \in \mathcal{S}_v} c_t(v, v') \\ &\leq \sum_{r \in \mathcal{S}_v} c_t(v, h(r)) + \sum_{r \in \mathcal{S}_v} c_t(h(r), s(r)) \\ &= C_r^{\mathrm{PRO}}(\mathcal{S}_v) + C_r^{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

Now, we compare the update cost of the proper placement PRO and the optimal restricted placement OPT_W. The additional cost in PRO caused by the update messages for the copy on v are less than

$$W \cdot c_t(v,v') \le \frac{k_2}{k_2-1} \cdot |\mathcal{S}_v| \cdot c_t(v,v')$$
$$\le \frac{k_2}{k_2-1} \cdot (C_r^{\mathrm{PRO}}(\mathcal{S}_v) + C_r^{\mathrm{OPT}_W}(\mathcal{S}_v)) \ .$$

Recall for the first step that $|\mathcal{S}_v| \ge (1-\frac{1}{k_2}) \cdot W$ due to Claim 6.

Summing this inequality over all nodes holding a copy in the proper placement and taking the cost for updating all copies of the optimal restricted placement into account yields that there exists a Steiner tree for updating all copies of PRO with cost at most $\frac{k_2}{k_2-1} \cdot (C_r^{\mathrm{PRO}} + C_r^{\mathrm{OPT}_W}) + C_u^{\mathrm{OPT}_W}$. Thus, $C_u^{\mathrm{PRO}} \le 2(\frac{k_2}{k_2-1} \cdot (C_r^{\mathrm{PRO}} + C_r^{\mathrm{OPT}_W}) + C_u^{\mathrm{OPT}_W})$, since we use a minimum spanning tree for updating. ■

Lemmata 4 and 5 yield the theorem. ■

2.2 The approximation algorithm

In this section, we present the algorithm that computes a proper placement with low storage cost. The algorithm consists of the following three phases.

1. An initial placement is calculated by an approximation algorithm for the facility location problem. The input is the *related facility location problem*, i.e., the same input as for our data management problem with the difference that all write requests become read requests. Hence, in this phase the update cost are neglected.

2. Additional copies are added to the initial placement. As long as there exists a node v whose nearest copy has a distance to v larger than $5r_s(v)$, a new copy is stored on v.

3. Copies that violate the second property of a proper placement are deleted in the following way. All nodes holding a copy are scanned in ascending order according to their write radii $r_w(v)$. For a current node v holding a copy, a copy on node u is deleted if $c_t(u,v) \le 4r_w(u)$.

Theorem 7 *The placement calculated by the above algorithm achieves a constant approximation factor for the static data management problem.*

Proof. In order to prove the theorem, we first show that the algorithm computes a proper placement. Then we show that the storage cost of this placement are optimal up to a constant factor.

Lemma 8 *The above algorithm calculates a placement fulfilling the following properties.*

- *Every node v has a copy in distance at most $29 \cdot \max\{r_w(v), r_s(v)\}$ from v. This is the first property of a proper placement with $k_1 = 29$.*
- *Two nodes u and v both holding a copy have at least distance $4\max\{r_w(u), r_w(v)\}$. This is the second property of a proper placement with $k_2 = 2$.*

Proof. First, we show that the second property holds. Let u and v denote two nodes both holding a copy. Assume for contradiction that $c_t(v,u) \le 4r_w(u)$. Obviously, the copy on v would have been deleted in the third phase of the algorithm.

In order to show the first property, we make the following observation. If a node v holds a copy after the second phase of the algorithm, there exists a node holding a copy in the final placement with distance at most $4r_w(v)$ to v. Assume that the copy on node v is deleted in phase 3 during the scan of node u that holds a copy. Thus $c_t(v,u) \leq 4r_w(v)$. Assume that the copy on node u is deleted later in phase 3 during the scan of some node v' that holds a copy. Hence, $c_t(u,v') \leq 4r_w(u) \leq 4r_w(v')$. The last inequality holds since the nodes are considered in ascending order according to their write radii. But in this case the copy on node v' would have been already deleted during the scan of node u, since $c_t(u,v') \leq 4r_w(v')$. This is a contradiction.

Now, we bound the distance from a node u to its closest copy in the network. Let v and v' denote the nodes holding the closest copy after the second and third phase of the algorithm, respectively. Then

$$c_t(u,v') \leq c_t(u,v) + c_t(v,v') \leq 5r_s(u) + 4r_w(v) \quad ,$$

because of the above observation and the triangle inequality.

To get the desired result we have to relate $r_w(v)$ to $r_w(u)$. Obviously, $r_w(v) \leq c_t(v,u) + r_w(u)$, since $r_w(v)$ denotes the average distance between v and the set of those W distinct requests that are closest to v. Thus,

$$c_t(u,v') \leq 5r_s(u) + 4r_w(v) \leq 25r_s(u) + 4r_w(u) \quad .$$

Thus choosing $k_1 = 29$ and $k_2 = 2$ we get a proper placement. ■

It remains to prove that the storage cost of the placement calculated by the algorithm are low.

Lemma 9 *The storage cost of the placement calculated by the above algorithm are bounded by $f \cdot (C_s^{\mathrm{OPT}_W} + C_r^{\mathrm{OPT}_W})$, where f denotes the approximation ratio of the facility location algorithm used in the first phase.*

Proof. Let C_s^i and C_r^i denote the storage cost and read cost of the placement after the ith phase of the algorithm, respectively. Further, let C_s^{FLP} and C_r^{FLP} denote the optimum storage cost and read cost, respectively, of the related facility location problem. Then

$$\begin{aligned} C_s^3 &\leq C_s^2 + C_r^2 \\ &\leq C_s^1 + C_r^1 \\ &\leq f \cdot (C_s^{\mathrm{FLP}} + C_r^{\mathrm{FLP}}) \\ &\leq f \cdot (C_s^{\mathrm{OPT}_W} + C_r^{\mathrm{OPT}_W}) \quad . \end{aligned}$$

The first inequality holds, since during the third phase copies are only deleted and thus the storage cost decreases. The second inequality will be shown in Claim 10. The last two inequalities are obvious.

Claim 10 *The sum of storage and read cost does not increase during the second phase of the algorithm, i.e., $C_s^2 + C_r^2 \leq C_s^1 + C_r^1$.*

Proof. In order to show the claim we need the following observation.

Observation 11 *In distance at most $2r_s(v)$ from a node v, there are at least $\frac{z_s(v)}{2}$ requests.*

Proof. We call a request r with $c_t(h(r),v) \leq 2r_s(v)$ *close to* v. Assume for contradiction that there are less than $\frac{z_s(v)}{2}$ close requests. In this case at most $\lceil \frac{z_s(v)}{2} - 1 \rceil$ requests are close to v. Therefore at least $(z_s(v) - 1) - \lceil \frac{z_s(v)}{2} - 1 \rceil = \lfloor \frac{z_s(v)}{2} \rfloor$ requests from the set $\mathcal{R}_v^{z_s(v)-1}$ are *not* close. Recall that $|\mathcal{R}_v^{z_s(v)-1}| = z_s(v) - 1$. Thus

$$\begin{aligned} d(v, z_s(v) - 1) &< \frac{1}{z_s(v) - 1} \cdot \left(\left\lfloor \frac{z_s(v)}{2} \right\rfloor \cdot 2r_s(v) \right) \\ &\leq d(v, z_s(v) - 1) \quad . \end{aligned}$$

This is a contradiction. ■

We will show that the cost always decreases whenever a copy is placed on a node v during the second phase. Before the new copy is added every copy has distance larger than $5r_s(v)$ from v. Therefore the $\frac{z_s(v)}{2}$ requests that are close to v have distance larger than $3r_s(v)$ to the nearest copy. This causes read cost of at least

$$\left\lceil \frac{z_s(v)}{2} \right\rceil \cdot 3r_s(v) \geq \left\lceil \frac{z_s(v)}{2} \right\rceil r_s(v) + c_s(v) \ .$$

The inequality follows directly from the choice of $r_s(v)$. When a copy has been placed on v the cost is

$$\left\lceil \frac{z_s(v)}{2} \right\rceil \cdot d\left(v, \left\lceil \frac{z_s(v)}{2} \right\rceil\right) + c_s(v) \leq \left\lceil \frac{z_s(v)}{2} \right\rceil r_s(v) + c_s(v) \ .$$

Hence, the read and storage cost do not increase when a copy is added during the second phase of the algorithm. ■

This completes the proof of the lemma. ■

Thus, the theorem follows from Theorem 3 and Lemmata 8 and 9. ■

3 The optimal algorithm for trees

In this section, we present an algorithm that calculates an optimal placement for the static data management problem on a rooted tree $T = (V, E)$. Our algorithm places all objects independently from each other. Thus, fix an object x. This section is organized as follows. First, we present for the read-only case, i.e., $f_w(v, x) = 0$ for all $v \in V$, an algorithm that computes an optimal placement for binary trees. By simulation, this algorithm can be used to compute an optimal placement for arbitrary trees. Finally, we show how to adapt this algorithm for the general case, i.e., for arbitrary read and write frequencies.

3.1 The read-only case

Let $\mathrm{diam}(T)$ denote the unweighted diameter of T, i.e., the maximum number of edges on a path connecting two arbitrary nodes in T, and let $\deg(T)$ denote its maximum node degree. For a node v, let T_v denote the subtree rooted at v, i.e., the connected component containing v if the edge incident to v and its father is removed. Then let $|T_v|$ denote the number of nodes in T_v.

A placement P on a subtree T_v consists of a set $\mathcal{C}$ of nodes holding copies, a set $\mathcal{R}_P^{\mathrm{ass}}$ of requests assigned to copies in T_v and a set $\mathcal{R}_P^{\mathrm{out}}$ of outgoing requests, i.e., requests that are not assigned to any copy in T_v. The cost of a placement P on a subtree T_v are defined as $\mathrm{cost}(P) := \sum_{v \in \mathcal{C}} c_s(v) + \sum_{r \in \mathcal{R}_P^{\mathrm{ass}}} c_t(h(r), s(r)) + \sum_{r \in \mathcal{R}_P^{\mathrm{out}}} c_t(h(r), v)$. The copy distance d_P of a placement P is defined to be the distance from node v to the closest copy in T_v. Further, we call a placement P of a subtree T_v *naturally assigned* if it fulfills the following conditions.

- If a request r is assigned to a copy in T_v then this is the closest copy to $h(r)$.
- All requests that pass a node are either assigned to the same copy in T_v or belong all to the set of outgoing requests.

Obviously, there always exists an optimal placement on T_v that is naturally assigned.

Our tree algorithm is based on the key observation that the optimal placement on a subtree T_v does not too heavily depend on the placement decisions made on $T \setminus T_v$. In fact it only depends on a few parameters. Thus, only a restricted number of placements on T_v have to be considered for calculating the optimal placement for T. This observation can be formalized as follows. We call a set $\mathcal{S}_{T_v}$ of placements on T_v *sufficient* if in each naturally assigned placement on T the placement on T_v can be replaced by a placement from $\mathcal{S}_{T_v}$ without increasing the total cost. Hence, only the placements in $\mathcal{S}_{T_v}$ have to be considered by an algorithm that searches for an optimal placement for T.

Lemma 12 *For any subtree T_v exists a sufficient set $\mathcal{S}_{T_v}$ with $|\mathcal{S}_{T_v}| \leq 2|T_v| + 1$.*

Proof. First, we define the set $\mathcal{S}_{T_v}$ as follows. For any $D \in \mathbb{R}_0^+$, the set $\mathcal{S}_{T_v}$ contains a placement $E_v^D := \arg\min_{\text{placement } P}\{\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D\}$ which is denoted the *optimal export placement for distance D*. Furthermore, for any $R \in \mathbb{N}$, the set $\mathcal{S}_{T_v}$ contains a placement $I_v^R := \arg\min_{\text{placement } P}\{\text{cost}(P) + d_P \cdot R\}$ which is denoted the *optimal import placement for request-quantity R*.

Now, we will prove that $\mathcal{S}_{T_v}$ is sufficient. Finally, we will show that it contains at most $2 \cdot |T_v| + 1$ placements. Suppose that we are given a naturally assigned placement P for T. Let P_{T_v} and $P_{T \setminus T_v}$ denote the corresponding sub-placements on T_v and $T \setminus T_v$, respectively. We have to replace the placement P_{T_v} by one of the placements from $\mathcal{S}_{T_v}$ without increasing the cost. We distinguish the following two cases: requests issued in $T \setminus T_v$ are served in T_v and requests issued in T_v are served in $T \setminus T_v$. Note that no other case can occur in a naturally assigned placement.

- Suppose that R requests issued in $T \setminus T_v$ are served in T_v. The total cost of the placement is $\text{cost}(P_{T\setminus T_v}) + \text{cost}(P_{T_v}) + d_{P_{T_v}} \cdot R$. Replacing P_{T_v} with I_v^R yields total cost $\text{cost}(P_{T\setminus T_v}) + \text{cost}(I_v^R) + d_{I_v^D} \cdot R$ which is not larger by the definition of I_v^R.
- Suppose that requests issued in T_v are served in $T \setminus T_v$ by a copy that has distance D from v. The total cost of this placement is $\text{cost}(P_{T\setminus T_v}) + \text{cost}(P_{T_v}) + |\mathcal{R}_{P_{T_v}}^{\text{out}}| \cdot D$. Similar to case 1, we do not increase the cost by replacing P_{T_v} with E_v^D.

Finally, we prove that $|\mathcal{S}_{T_v}| \leq 2|T_v| + 1$. First of all, $|T_v|$ placements suffice in order to contain a placement $I_v^R = \arg\min_{\text{placement } P}\{\text{cost}(P) + d_P \cdot R\}$, for any $R \in \mathbb{N}$, since the distance d_P from v to its nearest copy in T_v can only take $|T_v|$ distinct values, and for every value of d_P there is one optimal placement.

To prove that $|T_v| + 1$ placements in $\mathcal{S}_{T_v}$ suffice in order to contain a placement $E_v^D = \arg\min_{\text{placement } P}\{\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D\}$, for any $D \in \mathbb{R}_0^+$, we argue that $|\mathcal{R}_P^{\text{out}}|$ can only take $|T_v| + 1$ distinct values for placements that minimize $\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D$.

For distances D_1 and D_2 with $D_1 < D_2$, define the placements P_1 and P_2 with $P_i := \arg\min_{\text{placement } P}\{\text{cost}(P) + |\mathcal{R}_P| \cdot D_i\}$, for each $i \in \{1, 2\}$. For these placements hold that the requests of a node are either all assigned to the same copy or belong all to the set of outgoing requests. This follows from the fact that P_1 and P_2 are naturally assigned.

$|\mathcal{R}_P^{\text{out}}|$ takes only $|T_v|+1$ different values because if the requests of a node u are assigned to a copy in P_1, then these requests are assigned to a copy in P_2, as well. This will be shown in the following. Assume for contradiction that the requests of node u are assigned to a copy in P_1 and belong to the set of outgoing requests in P_2. We call a subtree $T_w \subseteq T_v$ *self-contained* if no request issued in T_w is assigned to a copy in $T_v \setminus T_w$ or belongs to the set of outgoing requests, and if no request issued in $T_v \setminus T_w$ is assigned to a copy in T_w. Obviously there exists a self-contained subtree T_w that contains u in the placement P_1. In placement P_2 the sub-placement of T_w has changed such that the requests from u belong to the set of outgoing requests. By exchanging the different sub-placements on T_w between P_1 and P_2 either the placement P_1 or P_2 would improve. This is a contradiction. ■

Now, we give a short sketch of the algorithm that computes an optimal placement for T. Let v_r denote the root node of T, i.e., $T = T_{v_r}$. The algorithm recursively computes the so called relevant parameters of the placements belonging to the sufficient set $\mathcal{S}_{T_{v_r}}$. The relevant parameters of a placement P_v on T_v are parameters that are used for computing the total cost of a placement on $T_w \supset T_v$ that uses P_v as placement on the subtree T_v. For the case that P_v is an import placement these parameters are the cost of P_v, its copy distance, and the node v_i holding the closest copy to v in T_v. If P_v is an export placement the relevant parameters are the cost of P_v, its number of outgoing requests, and an interval $\mathcal{I}_{P_v}$, indicating the distances $D \in \mathcal{I}_{P_v}$ for which P_v is an optimal export placement. When the relevant parameters of the placements in $\mathcal{S}_{T_{v_r}}$ have been computed the cost of E_v^∞ is the cost of an optimal placement on T.

Theorem 13 *The optimal placement on an arbitrary tree T can be computed in time $O(|T| \cdot \mathrm{diam}(T) \cdot \log(\deg(T)))$.*

Proof. The following lemma gives an upper bound on the time needed to compute the relevant parameters of all placements in the sufficient set of a node, if the parameters of the placements, in the sufficient sets of the children of this node are given.

Lemma 14 *For a binary subtree T_v of T the relevant parameters of all placements in the sufficient set $\mathcal{S}_{T_v}$ can be computed in time $O(|T_v|)$ if the parameters of the placements belonging to the sufficient sets of v's children are given.*

Proof. During the algorithm the set $\mathcal{S}_{T_v}$ is coded by a sequence of at most $|T_v|$ import tuples and a sequence of at most $|T_v| + 1$ export tuples that describe the corresponding import and export placements of $\mathcal{S}_{T_v}$. An import tuple (C_P, d_P, v_i) consists of the cost C_P of the corresponding placement P, the node v_i holding the closest copy to v in T_v and the copy distance $d_P := c_t(v, v_i)$ of P. An export tuple $(C_P, |\mathcal{R}_P|, \mathcal{I}_P)$ consists of the cost C_P of the corresponding placement P, the number of outgoing requests $|\mathcal{R}_P|$ of P and an optimality interval $\mathcal{I}_P$. For $D \in \mathcal{I}_P$, the placement P is the optimal export placement E_v^D.

During the algorithm the sequences of import tuples are sorted due to their copy distances and the sequences of export tuples are sorted due to their optimality intervals. Now, we explain how to construct these sequences for a subtree T_v.

If v is a leaf the construction works as follows. For an import placement the subtree T_v must contain a copy. Thus a copy has to be stored on v. This yields cost $c_s(v)$ and copy distance 0. For the construction of the export placements we distinguish two cases according to the distance D to a copy within $T \setminus T_v$.

- If $D < \frac{c_s(v)}{f_r(v)}$ then the optimal export placement E_v^D has no copy on v. The corresponding export tuple is $(0, f_r(v), [0, \ldots, \frac{c_s(v)}{f_r(v)}))$.
- If $D \geq \frac{c_s(v)}{f_r(v)}$ then a copy is placed on v in E_v^D. The export tuple is $(c_s(v), 0, [\frac{c_s(v)}{f_r(v)}, \ldots, \infty))$.

Now, suppose that v is an inner node of T and has two children v_1 and v_2, connected via edges $e_1 = (v_1, v)$ and $e_2 = (v_2, v)$, respectively. First we construct the tuples for the optimal import placements of $\mathcal{S}_{T_v}$.

Claim 15 *The sorted sequence of import tuples of T_v can be computed in time $O(|T_v|)$ if the sorted sequences of import and export tuples of T_{v_1} and T_{v_2} are given.*

Proof. In the proof of Lemma 12 was shown that for each node $v_i \in T_v$ there exists at most one optimal import placement in which v_i holds the closest copy to v in T_v. In the following we will denote this placement with $I_v^{v_i}$. This notation will coexist with the notation I_v^R for the optimal import placement with request-quantity R in the following. Note that each $I_v^{v_i} = I_v^R$ for R in a certain, possibly empty interval. We will construct an import tuple for every placement $I_v^{v_i}$.

The first tuple that has to be computed corresponds to a placement storing a copy on node v, i.e., $v_i = v$. This placement is optimal if a large number of requests from $T \setminus T_v$ are served within T_v, i.e., it equals I_v^∞. The copy distance of this tuple is 0. Furthermore, no requests enter the subtrees T_{v_1} and T_{v_2}. Thus, the cost of the tuple can be obtained by combining the tuple $(C_1, |\mathcal{R}_1|, \mathcal{I}_1)$ of $E_{v_1}^{c_t(e_1)}$ and the tuple $(C_2, |\mathcal{R}_2|, \mathcal{I}_2)$ of $E_{v_2}^{c_t(e_2)}$. It is given by $c_s(v) + C_1 + c_t(e_1) \cdot |\mathcal{R}_1| + C_2 + c_t(e_2) \cdot |\mathcal{R}_2|$.

Now assume w.l.o.g. that $v_i \in T_{v_1}$. Obviously, there exists an optimal placement in which no requests enter T_{v_2} from $T \setminus T_{v_2}$, because the copy on node $v_i \in T_{v_2}$ is at least as close as any copy in T_{v_2}. Thus, we can construct the optimal import placement $I_v^{v_i}$ by combining the placements $I_{v_1}^{v_i}$ for T_{v_1} and $E_{v_2}^{c_t(v_i,v_2)}$ for T_{v_2}. The only problem is that all tuples corresponding to some $I_v^{v_i}$, $v_i \in T_{v_1}$, have to be computed in time $O(|T_v|)$. In order to do this we exploit that the tuples of $I_{v_1}^{v_i}$ and $E_{v_2}^D$ are sorted. We traverse the sequence of import tuples in order of increasing copy distance and search for the required export tuple in the sequence of T_{v_2}. Since the tuples in this sequence are sorted in order of their optimality intervals, we can use linear search and start each search at the position the previous search stopped. Thus, we need only time $O(|T_{v_1}| + |T_{v_2}|) = O(|T_v|)$ for computing all $I_v^{v_i}$ with $v_i \in T_{v_1}$. After all tuples have been computed for $v_i \in T_{v_1}$ and $v_i \in T_{v_2}$, the resulting sequences which are still sorted have to be merged to get a sorted sequence for T_v. This can be done in time $O(|T_v|)$. ■

Now, we show how to construct the export tuples of T_v.

Claim 16 *The sorted sequence of export tuples of T_v can be computed in time $O(|T_v|)$ if the sorted sequences of export tuples of T_{v_1} and T_{v_2} are given.*

Proof. We start with the export tuple for the placement E_v^∞, i.e., the export placement that is optimal for $D = \infty$. Obviously, the number of outgoing requests of the corresponding export tuple is 0, because otherwise the term $\mathrm{cost}(E_v^\infty) + |\mathcal{R}_{E_v^\infty}| \cdot D$ would not be bounded. Since $|\mathcal{R}_{E_v^\infty}| = 0$ this placement can be viewed as an import placement with request-quantity 0. The optimal cost, i.e., $\mathrm{cost}(I_v^0)$, for this placement can be computed due to Claim 15. Thus we choose $C_v^E := \mathrm{cost}(I_v^0)$ for the cost of the tuple. The optimality interval for this tuple will be computed later.

This was the optimal placement for very large distances D. All other export placements have some outgoing requests, because otherwise they are not different from E_v^∞. Obviously, in such a placement no requests enter the subtrees T_{v_1} and T_{v_2}, because the placements are naturally assigned. Therefore each outstanding export placement can be obtained by combining two export placements of T_{v_1} and T_{v_2}. More formally the export tuple for E_v^D can be constructed from the combination of the tuples for $E_{v_1}^{D-c_t(e_1)}$ and $E_{v_2}^{D-c_t(e_2)}$.

In order to compute all required combinations in time $O(|T_v|)$ we first shift the optimality intervals of all export tuples of T_{v_1} and T_{v_2} by $-c_t(e_1)$ and $-c_t(e_2)$, respectively. Then the sequences of export tuples of T_{v_1} and T_{v_2} are traversed in increasing order of their optimality intervals. If the shifted optimality intervals of two export placements (one of subtree T_{v_1} and the other of T_{v_2}) intersect, the corresponding tuples $(C_1, |\mathcal{R}_1|, \mathcal{I}_1)$ and $(C_2, |\mathcal{R}_2|, \mathcal{I}_2)$ are combined for a new tuple $(C, |\mathcal{R}|, \mathcal{I})$ of T_v as follows. The optimality interval $\mathcal{I}$ is simply the intersection of the shifted intervals $\mathcal{I}_1$ and $\mathcal{I}_2$. The cost is given by $C := C_1 + C_2 + |\mathcal{R}_1| \cdot c_t(e_1) + |\mathcal{R}_2| \cdot c_t(e_2)$ and the number of outgoing requests is $|\mathcal{R}| := |\mathcal{R}_1| + |\mathcal{R}_2| + f_r(v)$.

After this computation has finished there are at most $|T_v|$ tuples with $|\mathcal{R}^{\mathrm{out}}| > 0$ and the tuple corresponding to the placement E_v^∞. It remains to determine the lower bound of the optimality interval for the last tuple, and to adjust the other intervals, accordingly. This is done as follows. For each tuple $E = (C_E, |\mathcal{R}_E|, \mathcal{I}_E)$ with $|\mathcal{R}_E| > 0$, we compute the distance D_E for which $C_E + D_E \cdot |\mathcal{R}_E| = \mathrm{cost}(E_v^\infty)$. If D_E is smaller than the lower bound of $\mathcal{I}_E$ the tuple E is deleted because the corresponding placement is never optimal. There exists exactly one tuple $E^* = (C_E^*, \mathcal{R}_E^*, \mathcal{I}_E^*)$ for which $D_E^* \in \mathcal{I}^*$ will hold. For this tuple the upper bound of the optimality interval will be set to D_E^*. Further, the distance D_E^* is the lower bound of the optimality interval for the tuple corresponding to E_v^∞.

Altogether, all these computations can be done in time $O(|T_v|)$ and give the sorted sequence of export tuples of T_v. ■

Claims 15 and 16 yield the lemma. ■

Applying this lemma the relevant parameters of an optimal placement OPT on a binary tree T can be computed in time $O(\sum_v |T_v|) = O(|T| \cdot \mathrm{diam}(|T|))$. This gives only the cost of OPT. To compute the whole placement we use the fact that for any tree T_v there is exactly one export tuple indicating that v holds a copy and at most one import tuple with this property. Thus, if we know which tuple from T_v is used to compute the cost of OPT, we know whether v holds a copy in OPT.

This is done as follows. For every tuple $\mathcal{T}$ we memorize from which tuples $\mathcal{T}_1$ and $\mathcal{T}_2$ it has been constructed. This can easily be realized by assigning two pointers to $\mathcal{T}$ pointing to $\mathcal{T}_1$ and $\mathcal{T}_2$. Finally, every tuple used to compute the cost of OPT can be reconstructed by following the pointers starting at the tuple with optimal cost.

It is easy to see that an arbitrary tree T can be simulated on a binary tree with $O(|T|)$ nodes and diameter $O(\mathrm{diam}(T) \cdot \log(\deg(T)))$, which yields the overall running time of $O(|T| \cdot \mathrm{diam}(T) \cdot \log(\deg(T)))$ for the computation of an optimal placement for T. ■

3.2 The general case

Now, we extend the results of the previous section to the general case where nodes can issue read and write requests to object x. A write request issued from node v increases the total cost by $\sum_{e \in E_{ST}} c_t(e)$ where E_{ST} denotes the edge set of the minimum Steiner-tree connecting v with all nodes holding a copy of x. We define the *write cost* $\mathrm{cost}_W(P)$ of a placement P on a subtree T_v as the cost caused by write messages along edges in T_v. Thus, a write request issued at node v increases $\mathrm{cost}_W(P)$ by $\sum_{e \in E_{ST} \cap E_{T_v}} c_t(e)$.

Unfortunately, this definition of the write cost of a placement on T_v depends on the placement of $T \setminus T_v$. More precisely, it depends on whether a copy is placed in $T \setminus T_v$ or not. Let $\mathrm{cost}_W^1(P)$ and $\mathrm{cost}_W^0(P)$ denote the write cost of P under the condition that at least one or no copy is placed in $T \setminus T_v$, respectively. These definitions are independent of the placement in $T \setminus T_v$ and either $\mathrm{cost}_W(P) = \mathrm{cost}_W^0(P)$ or $\mathrm{cost}_W(P) = \mathrm{cost}_W^1(P)$ holds.

Now, we will show that there exists a sufficient set with small cardinality for the new cost function. Adopting the notations and definitions from the previous section we call a set $\mathcal{S}_{T_v}$ *sufficient*, if in each naturally assigned placement on T the placement on T_v can be replaced by a placement in $\mathcal{S}_{T_v}$ without increasing the total cost. We define a sufficient set $\mathcal{S}_{T_v}$ as follows. Firstly, $\mathcal{S}_{T_v}$ contains a placement

$$E_v^D := \arg \min_{\text{placem. } P} \{\mathrm{cost}(P) + \mathrm{cost}_W^1(P) + |\mathcal{R}_P^{\mathrm{out}}| \cdot D\} \ ,$$

for any $D \in \mathbb{R}_0^+$. This is the optimal export placement for distance D. Note that $\mathrm{cost}(P)$, as in the previous section, only denotes the cost for serving read requests and storing objects. Secondly, $\mathcal{S}_{T_v}$ contains the placement E_v in which no node of T_v holds a copy.

Finally, for any $R \in \mathbb{N}$, the set $\mathcal{S}_{T_v}$ contains placements

$$I_v^R := \arg \min_{\text{placem. } P} \{\text{cost}(P) + \text{cost}_W^0(P) + d_P \cdot R\} \text{ and}$$
$$J_v^R := \arg \min_{\text{placem. } P} \{\text{cost}(P) + \text{cost}_W^1(P) + d_P \cdot R\} \quad .$$

These are the optimal import placements of request quantity R. Note that for each request quantity R two import placements are needed in order to ensure that $\mathcal{S}_{T_v}$ is sufficient. By similar techniques as used in Lemma 12 it can be shown that $\mathcal{S}_{T_v}$ has cardinality at most $3 \cdot |T_v| + 2$.

Now, we prove that the set $\mathcal{S}_{T_v}$ as defined above is indeed sufficient. We are given a placement P for T together with the corresponding subplacements P_{T_v} and $P_{T \setminus T_v}$ for T_v and $T \setminus T_v$, respectively. We have to replace the placement P_{T_v} by one of the placements in $\mathcal{S}_{T_v}$ without increasing the total cost. The proof is relatively straightforward and consists of a long enumeration of case distinctions. To shorten the proof we restrict ourselves to a single case.

- Suppose that R requests issued in $T \setminus T_v$ are served in T_v and that a copy is placed in $T \setminus T_v$. The total cost of the placement P is $\text{cost}(P_{T \setminus T_v}) + \text{cost}_W^1(P_{T \setminus T_v}) + \text{cost}(P_{T_v}) + \text{cost}_W^1(P_{T_v}) + d_{P_{T_v}} \cdot R$. Replacing P_{T_v} with J_v^R yields total cost $\text{cost}(P_{T \setminus T_v}) + \text{cost}_W^1(P_{T \setminus T_v}) + \text{cost}(J_v^R) + \text{cost}_W^1(J_v^R) + d_{J_v^R} \cdot R$, which is not larger by the definition of J_v^R.

The algorithm for the general case works similar to the algorithm in the previous section by simply computing the relevant parameters for all placements in each set $\mathcal{S}_{T_v}$. Its description is omitted since it gives no new algorithmic insights.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 164–173, 1993.

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28:67–104, 1998.

[3] I. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001.

[4] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, pages 39–50, 1992.

[5] F. Chudak. Improved approximation algorithms for uncapitated facility location. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 180–194, 1998.

[6] F. Chudak and D. Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 875–876, 1999.

[7] D. Dowdy and D. Foster. Comparative models of the file assignment problem. *Computing Surveys*, 14(2):287–313, 1982.

[8] M. Korupolu, G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facilty location problems. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 1998.

[9] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-line distributed data management. In *Proceedings of the 2nd European Symposium on Algorithms (ESA)*, 1994.

[10] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 284–293, 1997.

[11] F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Provably good and practical strategies for non-uniform data management in networks. In *Proceedings of the 7th European Symposium on Algorithms (ESA)*, pages 89–100, 1999.

[12] F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Caching in networks. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–439, 2000.

[13] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 265–274, 1997.

[14] E. Upfal and A. Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34:116–127, 1987.

[15] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Transactions on Data Base Systems (TODS)*, 16(1):181–205, 1991.