Placement Algorithms for Hierarchical Cooperative Caching*

Madhukar R. Korupolu[†] *C. Greg Plaxton*[*] *Rajmohan Rajaraman*[‡]

June 22, 1999

Abstract

Consider a hierarchical network in which each node periodically issues a request for an object drawn from a fixed set of unit-size objects. Suppose further that the following conditions are satisfied: the frequency with which each node accesses each object is known; each node has a cache of known capacity; any cache can be accessed by any node; any request is satisfied by the closest node with a copy of the desired object. In such an environment, it is desirable to fill the available cache space with copies of objects in such a way that the average access cost is minimized. We provide both exact and approximate polynomial-time algorithms for this hierarchical placement problem. Our exact algorithm is based on a reduction to min-cost flow, and does not appear to be practical for large problem sizes. Thus we are motivated to search for a faster approximation algorithm. Our main result is a simple constant-factor approximation algorithm for the hierarchical placement problem that admits an efficient distributed implementation.

1 Introduction

Cooperative caching [14] is an emerging paradigm in the design of scalable high-performance distributed systems. In traditional caching schemes, the primary function of a cache is to act as a fast intermediate storage between a client or a collection of clients and the servers. In such schemes, each request is satisfied by the cache associated with the requesting node or by the server of the requested object. Moreover, the storage decisions made by one cache are independent of those made by other caches in the system. The defining characteristic of cooperative caching schemes, on the other hand, is that the caches cooperate in serving one another's requests as well as in storage decisions.

A number of recent studies have discussed the benefits of cooperative caching for distributed file systems and large-scale information systems such as digital libraries and the World Wide Web. These studies include analytical results (e.g., [3, 22]), simulation experiments (e.g., [8, 16, 17, 28]) and prototypes and products (e.g., Harvest [9, 11], xFS [1]). The widely deployed and studied Harvest cache system [9], employs a hierarchical arrangement of object caches to improve access performance. In the Squid cache system [30], which is a successor to Harvest, the caches cooperate via the Internet Cache Protocol [31] to serve one another's misses and thus reduce overall traffic. Recent experimental work of [28] also indicates the potential for significant performance gains by cooperative caching on the Internet. In the context of local-area networks, the xFS system [1] utilizes cooperative caches to obtain a serverless file system. While the appropriate level of cooperation depends on the kind and

*A preliminary version of this paper appears in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pages 586–595.

†Department of Computer Science, University of Texas, Austin, TX 78712. Supported by NSF grant CCR–9504145. Email: {madhukar,plaxton}@cs.utexas.edu.

‡College of Computer Science, Northeastern University, Boston, MA 02115. Part of this work was done while the author was at DIMACS, which is an NSF Science and Technology Center, funded under contract STC–91–19999 and partially supported by the New Jersey Commission on Science and Technology. Email: rraj@ccs.neu.edu.

scale of the application, it is evident from these studies that cooperative caching will play a significant role in future information systems.

This paper studies an important component of cooperative caching schemes, which we refer to as placement. A cooperative caching scheme can be loosely divided into three components: placement, search, and consistency. The *placement* component determines where to place the copies of the objects. The *search* component directs each request to an appropriate copy of the requested object. Finally, the *consistency* component maintains the desired level of consistency among the various copies of an object.

In [3], Awerbuch, Bartal, and Fiat study a general on-line cooperative caching problem on arbitrary networks and present a polylog(n)-competitive algorithm, where n is the number of nodes in the network. Their result is impressive in that it addresses the search, placement, and consistency problems in a general adversarial setting. However, the time and space bounds established may exceed optimal by a polylogarithmic factor. In this paper, we study a special case of cooperative caching with the aim of developing simple algorithms that obtain near-optimal (e.g., constant-factor approximation) solutions. We focus our attention on the placement component of cooperative caching and develop placement algorithms for a class of networks that we refer to as *hierarchical networks*. Our definition of hierarchical networks, which is based on the ultrametric cost model used in [20], is motivated by the fact that modern wide-area networks tend to admit natural hierarchical decompositions. In fact, the existence of a hierarchical decomposition is implicit in several previous studies (e.g., see [11, 29, 30]).

1.1 The problem

We address the following placement problem for hierarchical networks. Let Ψ denote the set of objects and let $size(u)$ denote the cache size at node u. We are given for each node u and each object ψ the access frequency for ψ at u. Further assume that for any node u and object ψ, the cost of satisfying an access request for ψ originating at u is given by the communication cost between u and v, where v is the closest node (with respect to the communication cost function) that holds a copy of ψ. The objective of a placement algorithm is to determine a placement of object copies in the node caches subject to space constraints such that the average access cost over all nodes and all objects is minimized. See Section 2 for a formal statement of the problem.

Our problem formulation is most suitable for applications in which writes are infrequent and changes in the access pattern over short time intervals are moderate. Infrequent writes imply a low overhead in maintaining consistency among the copies of an object, allowing us to separate the concerns of consistency and placement. Moderate changes in the access pattern can be addressed by invoking the placement algorithm at regular intervals. Finally, we consider our assumption that each request is satisfied by the nearest copy of the requested object. This assumption is justified by the existence of algorithms for the search component that direct each request to a nearby, if not the nearest, copy of the requested object [4, 16, 25, 29]. Another useful idea in this regard is the summary cache protocol of [16] in which each cache maintains a synopsis of the contents of nearby caches so that it can redirect a request to a nearby copy (if one exists) in the event of a cache miss.

1.2 Our results

We first present a polynomial-time exact algorithm for the hierarchical placement problem based on a reduction to minimum-cost flow. Our reduction, described in Section 3, generalizes the approach of Leff, Wolf, and Yu [22], who solved the problem for the special case of a single-level hierarchy. While the algorithm of Section 3 runs in polynomial time, the degree of the polynomial is sufficiently high to make the algorithm largely impractical.

Thus, we focus on the goal of developing a fast, simple constant-factor approximation algorithm for the hierarchical placement problem that admits an efficient distributed implementation. In Section 4.1 we consider a natural bottom-up greedy algorithm, but find that this algorithm has approximation ratio $\Omega(n)$. The lower bound proof leads us to a natural refinement of the greedy algorithm, the amortizing algorithm, described in Section 4.2. Like the greedy algorithm, the amortizing algorithm starts with a placement in which each the cache of each node holds the locally optimal set of objects. The algorithm then iteratively improves the placement in a bottom-up manner as the nodes cooperate and share information about the access frequencies across larger regions of the network.

Our main result, shown in Section 5, is that the amortizing algorithm achieves a constant-factor approximation for the hierarchical placement problem. (The constant factor is less than 14.) The proof of our main technical lemma appears in Section 6. In Section 7, we describe an efficient distributed implementation of the amortizing algorithm.

It is worth noting that the recent results on the approximation of general metrics by tree metrics [5, 6, 12] imply that any hierarchical placement algorithm can be used to obtain a placement algorithm for general metrics giving up an extra $O(\log n \log \log n)$ factor in the approximation.

1.3 Related work

Dowdy and Foster [15] initiated the study of cooperative caching in the context of allocating files in a distributed network [15]. A sequence of results [2, 7, 23] obtained improved algorithms for centralized as well as distributed file allocation. These results, however, did not consider cache capacities at the individual nodes. As mentioned earlier in this section, Awerbuch, Bartal, and Fiat [3] provide a polylog(n)-competitive on-line algorithm for the general placement problem under the assumption that the size of each cache in the on-line algorithm is polylog(n) times more than the size in the optimal algorithm. In contrast, we obtain an optimal centralized and a constant-factor approximate distributed algorithm for the off-line version of the problem on hierarchical networks without any blowup in the cache sizes.

In [22], Leff, Wolf, and Yu study the placement problem for a network of workstations, which they model as a single-level hierarchy. In addition to providing an optimal centralized algorithm for this case, they give heuristics for a distributed solution. These heuristics, however, make use of particular properties of a single-level hierarchy that are not applicable in an arbitrary hierarchical setting.

By adopting a communication model based on a fixed cost function, we endeavor to separate the concerns of caching (a higher-level operation) from routing (a lower-level operation). In contrast, some recent papers have incorporated routing issues into caching by either combining the two problems or making use of available routing information. For example, the algorithms developed in [18, 26, 32] tend to cache copies of an object in nodes that either lie on or are close to the path along which the object is being transferred. Routing information is also used in the placement algorithms developed in [24], where the primary aim is to minimize the network congestion that may occur when requests and their responses are routed within the network. We also remark that cost models have been adopted in uniprocessor caching systems to model scenarios in which the costs incurred in the retrieval of objects on cache misses may vary from one object to another [10, 19, 33].

With regard to uniprocessor caching schemes, recent research has addressed the challenge of designing cache replacement policies that take into account the differing costs incurred in the retrieval of objects on cache misses. This has led to studies formulating generalizations of the traditional uniprocessor caching problems that account for the differing costs [10, 19, 33].

In a recent experimental study [21], Korupolu and Dahlin evaluate the practical performance of several placement and replacement algorithms for cooperative caching. Their simulation experiments demonstrate that, in practice, both our greedy placement algorithm as well as our amortizing placement algorithm are in fact very close to the optimal.

We remark that the placement problem can be viewed as an instance of facility location with multiple types of facilities and constraints on the number of facilities that can located at a point. To the best of our knowledge, this multiple facilities location problem has not been studied previously. For a survey of results related to facility location, see [13, 27].

2 Preliminaries

In this section, we formally define the hierarchical placement problem. To simplify the exposition, we define this problem with respect to a fixed tuple $(\Psi, \mathcal{V}, \mathit{distance}, \mathit{frequency}, \mathit{size}, \mathit{penalty})$, where Ψ is a set of unit-size read-only *objects*, $\mathcal{V}$ is a set of *nodes*, $\mathit{distance} : \mathcal{V} \times \mathcal{V} \to \mathbf{R}$, $\mathit{frequency} : \mathcal{V} \times \Psi \to \mathbf{R}$, $\mathit{size} : \mathcal{V} \to \mathbf{N}$, and *penalty* is a real number. We assume that the set of nodes $\mathcal{V}$ forms a hierarchy as defined in Section 2.1. We assume that *penalty* is at least as large as $\mathit{diameter}(\mathcal{V})$, where for any set of nodes U, $\mathit{diameter}(U)$ is defined as the maximum value of $\mathit{distance}(u, v)$ over all nodes u and v in $\mathcal{V}$. The hierarchical placement problem is defined in Section 2.2.

2.1 Hierarchies

We now inductively define the notion of a *hierarchy*. For any node u in $\mathcal{V}$, the singleton set $\{u\}$ is a hierarchy iff $\mathit{distance}(u, u) = 0$. A set of nodes U such that $|U| > 1$ is a hierarchy iff there is a partition of U into $k > 1$ disjoint hierarchies α_i, $0 \le i < k$, such that $\mathit{distance}(u, v) = \mathit{diameter}(U)$ (resp., $\mathit{distance}(u, v) < \mathit{diameter}(U)$) for all nodes u in α_i and v in α_j for which $i \ne j$ (resp., $i = j$). Note that for any non-singleton hierarchy α, this partition is unique. For any non-singleton hierarchy α with associated partition $\{\alpha_i : 0 \le i < k\}$, we define each hierarchy α_i as a *child* of the hierarchy α, and we define the *parent* of each α_i, denoted $\mathit{parent}(\alpha_i)$, as α. We inductively define the notion of a *descendant* of a hierarchy α as follows: A hierarchy β is a descendant of a hierarchy α iff $\alpha = \beta$ or β is a descendant of some child of α. A descendant β of α is *proper* iff $\alpha \ne \beta$.

Hierarchies can be used to model a large class of distributed networks. For example, a homogeneous, k-node local-area network may be modeled as a single-level hierarchy. In fact, this is precisely the model used in [1, 22] in the study of caching schemes for networks of workstations. Furthermore, it seems plausible that some nontrivial hierarchy should provide a reasonable first-order model for a complex, heterogeneous wide-area network such as the Internet.

Hierarchies can also be used to model multi-level storage; we can introduce a hierarchy for each level of storage and incorporate the disparate speeds of the different levels in the distance function. As a simple example, consider a machine with two levels of storage (e.g., memory and disk) having local access latencies a and b, $a < b$. This machine may be modeled as a hierarchy with diameter b and two children: (i) a singleton hierarchy with zero access frequencies and storage capacity equal to that of the slow level of storage, and (ii) a hierarchy with diameter a and two singleton children, one with zero access frequencies and storage capacity equal to that of the fast level of storage, and another with zero storage and access frequencies equal to those of the original machine. This approach can be generalized to capture both network and cache latencies in a heterogeneous distributed network with caches of varying speeds.

2.2 The hierarchical placement problem

Having fixed the tuple $(\Psi, \mathcal{V}, \mathit{distance}, \mathit{frequency}, \mathit{size}, \mathit{penalty})$ as specified at the beginning of Section 2, any descendant of the hierarchy $\mathcal{V}$ determines an instance of the hierarchical placement problem. We now present a sequence of definitions leading up to the definition of the hierarchical placement problem.

It is convenient to extend the definitions of the functions *frequency* and *size* to act on hierarchies. For any hierarchy α and any object ψ, we define $\mathit{frequency}(\alpha, \psi)$ as the sum of $\mathit{frequency}(u, \psi)$ over all nodes u in α. For any hierarchy α, we define $\mathit{size}(\alpha)$ as the sum of $\mathit{size}(u)$ over all nodes in α. The following definitions involving hierarchies will also prove to be useful. For any proper descendant α of $\mathcal{V}$, we define $\mathit{miss}(\alpha)$ as $\mathit{diameter}(\mathit{parent}(\alpha))$. For the hierarchy $\mathcal{V}$ itself, we define $\mathit{miss}(\mathcal{V})$ as *penalty*. For any hierarchy α and any object ψ, we define $\mathit{value}(\alpha, \psi)$ as $\mathit{frequency}(\alpha, \psi) \cdot (\mathit{miss}(\alpha) - \mathit{diameter}(\alpha))$.

A *copy* is a pair (α, ψ) where α is a hierarchy and ψ is an object. A copy (α, ψ) is *concrete* iff α is singleton. A set of copies is *concrete* iff it is a set of concrete copies. A *refinement* of a set of copies P is a set of copies Q for which $|P| = |Q|$ and there exists a bijection $\phi : P \to Q$ such that for all $p = (\alpha, \psi)$ in P, the copy $\phi(p) = (\beta, \psi)$ for some descendant β of α. A concrete set of copies P is *feasible* iff $|\{(\alpha, \psi) \in P : \psi \in \Psi\}| \leq size(\alpha)$ for each singleton hierarchy α in $\mathcal{V}$. A non-concrete set of copies is *feasible* iff it admits a feasible concrete refinement. A *placement* is a pair (α, P) where α is a hierarchy, P is a set of copies, and each copy in P is of the form (β, ψ) for some descendant β of α. For any placement $A = (\alpha, P)$, we define $Hierarchy(A)$ and $Copies(A)$ as α and P, respectively. For any hierarchy α, a placement A is a *α-placement* iff $Hierarchy(A) = \alpha$. A placement A is *concrete* (resp. *feasible*) iff $Copies(A)$ is concrete (resp., feasible). A placement (α, P) is a *refinement* of a placement (β, Q) iff $\alpha = \beta$ and P is a refinement of Q. For any placement A and any descendant α of $Hierarchy(A)$, we define $restrict(A, \alpha)$ as the placement (α, P) where P is the set of all copies (β, ψ) in $Copies(A)$ such that β is a descendant of α. The following lemma is straightforward to prove.

Lemma 2.1 *For any hierarchy α, an α-placement A is feasible iff $|Copies(restrict(A, \beta))| \leq size(\beta)$ for all descendants β of α.* ■

For any placement A and object ψ, a copy p is an *(A, ψ)-copy* iff p belongs to $Copies(A)$ and has associated object ψ. For any placement A and object ψ, we define $count(A, \psi)$ as the number of (A, ψ)-copies. For any placement A, we define $Missing(A)$ as the set of all objects ψ such that $count(A, \psi) = 0$. For any placement A, node u in $Hierarchy(A)$, and object ψ, we define $latency(A, u, \psi)$ as follows: If ψ belongs to $Missing(A)$, then $latency(A, u, \psi) = miss(Hierarchy(A))$; otherwise, $latency(A, u, \psi)$ is the minimum value of $diameter(\alpha)$ over all descendants α of $Hierarchy(A)$ such that u belongs to α and $count(restrict(A, \alpha), \psi) > 0$. Note that if A is a refinement of B, then $latency(A, u, \psi) \leq latency(B, u, \psi)$. For any placement A, we define the *cost* of A, denoted $cost(A)$, as the sum of $frequency(u, \psi) \cdot latency(A, u, \psi)$ over all nodes u in $Hierarchy(A)$ and all objects ψ in Ψ. Given a hierarchy α, the *hierarchical placement problem* is to find a feasible concrete α-placement of minimum cost. We remark that if A is a refinement of B, then $cost(A) \leq cost(B)$; it follows that some minimum-cost feasible α-placement is concrete.

A set of placements is defined to be *comparable* iff each placement in the set has the same associated hierarchy. Two placements A and B are defined to be *coupled* iff they are comparable and $count(A, \psi) = count(B, \psi)$ for all objects ψ. A triple of placements (A, B, C) is defined to be *nice* iff A, B, and C are comparable and A and B are coupled.

3 A reduction to minimum-cost flow

In this section we reduce a given instance α of the hierarchical placement problem to a corresponding instance G_α of the minimum-cost flow problem. If the hierarchy α is singleton then the placement problem is trivial; our presentation assumes that α is non-singleton.

The minimum-cost flow instance G_α is constructed as follows. The vertex set consists of the following: a vertex β for every singleton descendant β of α; a vertex $\langle\psi, \beta\rangle$ for every object ψ and descendant β of α; a source s and sink t. The edge set consists of four types of edges: for each singleton descendant β of α and each object ψ, there is a unit-capacity edge $(\langle\psi, parent(\beta)\rangle, \beta)$ with cost $-value(\beta, \psi)$; for each non-singleton proper descendant β of α and each object ψ, there are two parallel edges $(\langle\psi, parent(\beta)\rangle, \langle\psi, \beta\rangle)$ with capacities 1 and ∞, and costs $-value(\beta, \psi)$ and 0, respectively; for each object ψ, there are two parallel edges $(s, \langle\psi, \alpha\rangle)$ with capacities 1 and ∞, and costs $-value(\alpha, \psi)$ and 0, respectively; for each singleton descendant β of α, there is an edge (β, t) with capacity $size(\beta)$ and cost 0.

Let c denote the sum, over all nodes u in α and all objects ψ in Ψ, of $frequency(u, \psi) \cdot miss(\alpha)$. The next two lemmas imply that an integral minimum-cost flow in G_α corresponds to a feasible concrete α-placement of minimum cost.

Lemma 3.1 *For every feasible concrete α-placement A, there is an integral flow in G_α with cost equal to $cost(A) - c$.*

Proof: For any hierarchy β, let A_β denote the placement $restrict(A, \beta)$. We construct the desired flow as follows. For each edge $(\langle\psi, parent(\beta)\rangle, \beta)$, we set the flow to 1 if $count(A_\beta, \psi) > 0$, and to 0 otherwise. For each unit-capacity edge $(\langle \psi, parent(\beta)\rangle, \langle \psi, \beta\rangle)$, we set the flow to 1 if $count(A_\beta, \psi) > 0$, and to 0 otherwise. For each infinite-capacity edge $(\langle \psi, parent(\beta)\rangle, \langle \psi, \beta\rangle)$, we set the flow to $\max\{0, count(A_\beta, \psi) - 1\}$. For each edge $(s, \langle \psi, \alpha\rangle)$, we set the flow to 1 if $count(A_\beta, \psi) > 0$, and to 0 otherwise. For each edge (β, t), we set the flow to the sum over all objects ψ of $count(A_\beta, \psi)$. It is straightforward to prove that the above flow is feasible and has cost equal to $cost(A) - c$. ∎

Lemma 3.2 *For every integral minimum-cost flow with cost c' in G_α, there is a feasible concrete α-placement A such that $cost(A) = c + c'$.*

Proof: Given an integral minimum-cost flow in G_α, we define a corresponding concrete α-placement A as follows: For any singleton descendant β of α and any object ψ, $Copies(A)$ includes the concrete copy (β, ψ) iff the flow along edge $(\langle \psi, parent(\beta)\rangle, t)$ is 1. (Note that A is concrete and feasible.) It remains to establish that $cost(A) = c + c'$. To establish this, we make the following key observation regarding the flows on adjacent parallel arcs. Let e_0 and e_1 denote the parallel edges with capacities 1 and ∞, respectively, between a pair of vertices. If the flow along e_1 is positive, then the flow along e_0 is 1. Otherwise, transferring a unit of flow from e_1 to e_0 would yield another feasible flow with smaller cost, a contradiction. It follows that if $count(restrict(A, \beta), \psi) > 0$ for some proper descendant β of α, then the flow along edge $(\langle \psi, parent(\beta)\rangle, \langle \psi, \beta\rangle)$ is 1, and hence there is an appropriate negative contribution to the cost of the flow. ∎

4 An approximation algorithm

While the algorithm of Section 3 computes an optimal solution to the hierarchical placement problem, its run-time complexity is prohibitively high, at least quadratic in the product of the number of nodes n and the number of objects m. This motivates us to seek a faster approximation algorithm. Two candidate algorithms are presented in this section. The first algorithm, which we refer to as the greedy algorithm, uses a natural local improvement heuristic. We show, however, that the greedy algorithm has an approximation ratio of $\Omega(n)$. The lower bound proof leads us to a variant of the greedy algorithm that we refer to as the amortizing algorithm. Section 5 establishes that the amortizing algorithm is a constant-factor approximation algorithm. Section 7 outlines an efficient distributed implementation of the amortizing algorithm.

Given any feasible non-concrete placement A, the following simple procedure can be used to obtain a feasible concrete refinement of A. Note that $Copies(A)$ contains a copy p of the form (α, ψ) for some non-singleton hierarchy α and object ψ. Using Lemma 2.1, we conclude that there exists a child β of α for which $|Copies(restrict(A, \beta))| < size(\beta)$. Hence we can obtain a feasible refinement of A by removing p from $Copies(A)$ and replacing it with the copy (β, ψ). Repeated application of this argument yields a concrete refinement of A. (Remark: For any feasible placement A, a minimum-cost feasible concrete refinement of A can be obtained by solving a suitably defined weighted matching problem.)

The greedy and amortizing placement algorithms described in Sections 4.1 and 4.2, respectively, each compute a placement A that is feasible but not necessarily concrete. The refinement procedure of the preceding paragraph may then be applied to obtain a feasible concrete placement B such that $cost(B) \leq cost(A)$.

The greedy and amortizing algorithms, though well-defined for arbitrary hierarchies, are only intended to be directly applied to hierarchies that are "λ-separated" for some constant $\lambda > 1$. A hierarchy U is *λ-separated* iff $miss(V) \geq \lambda \cdot diameter(V)$ for every descendant V of U. It is straightforward to show that any c-approximation algorithm for the λ-separated hierarchical placement problem implies a $c\lambda$-approximation algorithm for the hierarchical placement problem. (The main idea is to transform the given hierarchy into a λ-separated hierarchy by rounding up all distances to the nearest integral power of λ.) Thus, for the purposes of obtaining a constant-factor approximation algorithm, we may

Swapping Procedure: $Swap(A, p)$

- Set $Copies(A)$ to $(Copies(A) - \{p\}) \cup \{(Hierarchy(A), candidate(A))\}$.

Figure 1: The swapping procedure. The input is a placement A and a copy p in $Copies(A)$.

assume without loss of generality that the input hierarchy is λ-separated for an arbitrary constant $\lambda > 1$.

4.1 The greedy placement algorithm

For any placement A and any copy p in $Copies(A)$, we define *min-benefit*(A, p) as the amount by which $cost(A)$ would increase if p were removed from $Copies(A)$. Given a placement A for which $Copies(A)$ is nonempty, the *greedy elimination rule* removes from $Copies(A)$ the copy p minimizing *min-benefit*(A, p); this copy is denoted $victim(A)$. (Ties may be broken in an arbitrary consistent manner. For example, we could assign a unique integer ID to each copy, and use these IDs to break ties. Such tie-breaking conventions will be assumed throughout the remainder of the paper without further comment.)

In the following definitions, let A denote a placement, let ψ denote an object, let k denote $|Copies(A)|$, let $A_0 = A$, let A_{i+1} denote the placement $(Hierarchy(A), Copies(A_i) - \{victim(A_i)\})$, $0 \le i < k$, let p denote a (A, ψ)-copy, and let j denote the maximum value of i such that p belongs to $Copies(A_i)$. We define *benefit*(A, p) as *min-benefit*(A_j, p). The copy p belongs to the set $Primary(A)$ iff there is no other (A_j, ψ)-copy. We define $Secondary(A)$ as $Copies(A) - Primary(A)$. For any placement A, we define *secondary-victim*(A) as the copy p in $Secondary(A)$ minimizing *benefit*(A, p). If $Secondary(A)$ is empty, then *secondary-victim*(A) is undefined and it is convenient to assume that *benefit*$(A, \textit{secondary-victim}(A)) = \infty$.

In order to facilitate the next definition, we assume that $Missing(A)$ is nonempty for any placement A. This assumption is made without loss of generality since the set of objects Ψ can be augmented with arbitrarily many dummy objects for which the associated access frequencies are all zero. For any placement A, we define $candidate(A)$ as the object ψ in $Missing(A)$ maximizing $value(Hierarchy(A), \psi)$.

The swapping procedure of Figure 1 is used in all of our approximation algorithms for the hierarchical placement problem. Lemma 2.1 implies that if the placement passed to the swapping procedure is feasible, then the updated placement is also feasible.

The proofs of the following six lemmas are all straightforward and hence are omitted.

Lemma 4.1 *For any placement A such that $Copies(A)$ is nonempty, and any p in $Copies(A)$, we have $benefit(A, victim(A)) \le benefit(A, p)$.* ■

Lemma 4.2 *For any placement A and any object ψ such that there is at least one (A, ψ)-copy, a unique (A, ψ)-copy p belongs to $Primary(A)$ and $benefit(A, p) \ge benefit(A, q)$ for all (A, ψ)-copies q.* ■

Lemma 4.3 *Let α denote a λ-separated hierarchy for some $\lambda > 1$, let A denote an α-placement, and let p denote a (A, ψ)-copy in $Copies(A)$. Then $benefit(A, p) \le \frac{\lambda}{\lambda - 1} \cdot value(\alpha, \psi)$.* ■

Lemma 4.4 *(Delete) Let p denote $victim(A)$ (resp., secondary-victim(A)) for some placement A, and let A' denote the placement $(Hierarchy(A), Copies(A) - \{p\})$. Then the following conditions hold for all q in $Copies(A')$: $benefit(A, q) = benefit(A', q)$; q belongs to $Primary(A)$ (resp., $Secondary(A)$) iff q belongs to $Primary(A')$ (resp., $Secondary(A')$).* ■

Lemma 4.5 *(Insert) Let α denote a hierarchy, let A denote an α-placement, let ψ denote an object in $Missing(A)$, let p denote the copy (α, ψ) and let A' denote the placement $(\alpha, Copies(A) \cup \{p\})$.*

Greedy Algorithm

- **Combining.** Initialize $\mathit{Hierarchy}(A)$ to α. If α is singleton, initialize $\mathit{Copies}(A)$ to $\{(\alpha, \psi) : \psi \in S\}$, where S is an arbitrarily chosen subset of Ψ of size $\mathit{size}(\alpha)$. Otherwise, initialize $\mathit{Copies}(A)$ to $\cup_{0 \le i < k} \mathit{Copies}(A_i)$, where the A_i's are the placements previously computed at the k children of α.
- **Swapping.** While $\mathit{benefit}(A, \mathit{victim}(A)) < \mathit{value}(\alpha, \mathit{candidate}(A))$, call $\mathit{Swap}(A, \mathit{victim}(A))$.

Figure 2: The greedy algorithm. We assume that the children, if any, of a given hierarchy α have already been processed, and describe the computation associated with α.

Then p belongs to $\mathit{Primary}(A')$, $\mathit{benefit}(A', p) = \mathit{value}(\alpha, p)$, and the following claims hold for all q in $\mathit{Copies}(A)$: $\mathit{benefit}(A, q) = \mathit{benefit}(A', q)$; q belongs to $\mathit{Primary}(A)$ (resp., $\mathit{Secondary}(A)$) iff q belongs to $\mathit{Primary}(A')$ (resp., $\mathit{Secondary}(A')$). ■

Lemma 4.6 *(Combine) Let α denote a non-singleton hierarchy with k children α_i, $0 \le i < k$, let A_i denote an α_i-placement, $0 \le i < k$, and let A denote the placement $(\alpha, \cup_{0 \le i < k} \mathit{Copies}(A_i))$. For any object ψ such that there is at least one (A, ψ)-copy, let p denote the unique (A, ψ)-copy in $\mathit{Primary}(A)$, and let x denote $\mathit{benefit}(A, p) - \mathit{value}(\alpha, \psi)$. Then for any i, $0 \le i < k$, and any (A_i, ψ)-copy q, the following claim holds: If $p = q$ then $\mathit{benefit}(A_i, q) = x$; otherwise, $\mathit{benefit}(A, q) = \mathit{benefit}(A_i, q) \le x$.* ■

The greedy algorithm is presented in Figure 2. Using Lemmas 4.4 and 4.5, it is straightforward to establish termination of the swapping loop; it follows that the greedy algorithm terminates. It is straightforward to prove that the placement A computed by the greedy algorithm is feasible.

One might conjecture the greedy algorithm to be a constant-factor approximation algorithm for the hierarchical placement problem. Unfortunately, this conjecture is false; below we construct an n-node hierarchy α for which the placement computed by the greedy algorithm has cost exceeding the optimal by a $\Theta(n)$ factor.

We label n nodes from 0 to $n-1$ and construct the hierarchy $\alpha = \alpha_{n-1}$ as follows. First, we create a hierarchy α_1 with two singleton children corresponding to nodes 0 and 1. Then, for i running from 2 to $n-1$, we create a hierarchy α_i with two children: the hierarchy α_{i-1} constructed previously and a singleton hierarchy containing node i. The cache size of each node is 1. We define the distance function between the nodes in such a way that $\mathit{diameter}(\alpha_i)$ equals n^{i-1}, $1 \le i < n$. The *penalty*, which is required to be at least as large as $\mathit{diameter}(\alpha)$, is set to n^{n-1}. We assign nonzero frequencies to n objects ψ_i, $0 \le i < n$. For node 0, $\mathit{frequency}(0, \psi_i) = 1/n^i$, $0 \le i < n$. For node 1, $\mathit{frequency}(1, \psi_i)$ is 1 if i is 0, and 0 otherwise. For node j, $2 \le j < n$, $\mathit{frequency}(j, \psi_i)$ is 0 for all i.

For this example, the set of copies associated with an optimal feasible concrete α-placement A consists of $(\{0\}, \psi_0)$, $(\{1\}, \psi_1)$, and $(\{i\}, \psi_i)$, $2 \le i < n$. The cost of A is $2 - 1/n$ since $\mathit{latency}(A, 0, \psi_i)$ is n^{i-1}, $1 \le i < n$, and $\mathit{latency}(A, 1, \psi_0)$ is 1. In contrast, we claim that the cost of the greedy α-placement is $n-1$. (Furthermore, any refinement of this α-placement also has cost $n-1$.) It can be shown by induction that, for each hierarchy α_i, $1 \le i < n$, the set of copies associated with the greedy α_i-placement consists of $(\{0\}, \psi_0)$, $(\{1\}, \psi_0)$, and (α_j, ψ_{j-1}), $2 \le j \le i$. The cost of the greedy α-placement B is $n-1$ since $\mathit{latency}(B, 0, \psi_i)$ is n^i for $1 \le i < n$. We conclude that the approximation ratio of the greedy algorithm is $\Omega(n)$.

Amortizing Algorithm

- **Combining.** This step is the same as the combining step of the greedy algorithm, except that we also initialize an auxiliary potential variable Φ. If α is singleton, then Φ is set to 0. Otherwise, Φ is set to the sum of the potentials Φ_i, $0 \leq i < k$, computed at the k children of α.
- **Local Initialization.** Initialize Δ to the sum over all objects ψ in $\mathit{Missing}(A)$ of $\mathit{value}(\alpha, \psi)$.
- **Amortized Swapping.** This step is similar to the swapping step of the greedy algorithm, except that the potential Φ is used to reduce the benefits of certain secondary copies.

  1. Let $x = \mathit{benefit}(A, \mathit{secondary\text{-}victim}(A))$, let $y = \mathit{benefit}(A, \mathit{victim}(A))$, and let $z = \mathit{value}(\alpha, \mathit{candidate}(A))$.
  2. If $x - \Phi \leq \min(y, z)$ then call $\mathit{Swap}(A, \mathit{secondary\text{-}victim}(A))$, subtract z from Δ, set Φ to $\max(0, \Phi - x)$, and goto line 1.
  3. If $y < z$, then call $\mathit{Swap}(A, \mathit{victim}(A))$, add $y - z$ to Δ, and goto line 1.

- **Potential Update.** Add Δ to Φ.

Figure 3: The amortizing algorithm. We assume that the children, if any, of a given hierarchy α have already been processed, and describe the computation associated with α.

4.2 The amortizing placement algorithm

The construction given at the end of Section 4.1 leads us to consider a natural variant of the greedy algorithm that we call the amortizing algorithm. The amortizing algorithm is presented in Figure 3. Using Lemmas 4.4 and 4.5, it is straightforward to establish termination of the amortized swapping loop; it follows that the amortizing algorithm terminates. It is straightforward to prove that the placement A computed by the amortized algorithm is feasible.

An efficient distributed implementation of the amortizing algorithm is given in Section 7. It is particularly noteworthy that the amortized swapping loop executed at each non-singleton hierarchy is highly parallelizable.

5 Analysis of the amortizing algorithm

In this section we prove our main result, namely, that the cost of the placement constructed by the amortizing algorithm is within a constant factor of optimal. To facilitate our analysis, we introduce another placement algorithm that we call the bridging algorithm. The bridging algorithm computes three comparable feasible placements that we refer to as the amortizing, arbitrary, and bridging placements, respectively. The amortizing placement is identical to that computed by the amortizing algorithm. The arbitrary placement is simply an arbitrary concrete placement. The construction of the bridging placement depends on the choice of the arbitrary placement, and is designed to ensure that the cost of the bridging placement can be relatively easily compared to that of the amortizing and arbitrary placements. In particular, we establish our main theorem via the following two main steps. First, we prove that the cost of the amortizing placement is at most that of the bridging placement. Second, we prove that the cost of the arbitrary placement is at least some constant fraction of that of the bridging placement.

5.1 The bridging algorithm

For any placement A and copy $p = (\alpha, \psi)$ in *Copies*(A), we define *Region*(A, p) as the set of nodes u such that *latency*(A, u, ψ) would increase if p were removed from *Copies*(A); it is straightforward to prove that *Region*(A, p) is a hierarchy. For any comparable placements A and B, we define *Min-matched*(A, B) as the set of all $p = (\alpha, \psi)$ in *Copies*(A) for which there is a copy $q = (\beta, \psi)$ in *Copies*(B) such that β is a descendant of *Region*(A, p). We define *Min-unmatched*(A, B) as *Copies*(A) $-$ *Min-matched*(A, B).

In the following definitions, let A and B denote two comparable placements, let ψ denote an object, and let P (resp., Q) denote the set of all (A, ψ)-copies in *Min-matched*(A, B) (resp., *Min-unmatched*(A, B)). If there are one or more (A, ψ)-copies, then we define *quasivictim*(A, B, ψ) as follows: If Q is empty (resp., nonempty), then *quasivictim*(A, B, ψ) is the copy p in P (resp., Q) minimizing *min-benefit*(A, p). For the remaining definitions in this paragraph, let k denote the number of (A, ψ)-copies, let $A_0 = A$, let A_{i+1} denote the placement $= (\mathit{Hierarchy}(A), \mathit{Copies}(A_i) - \{\mathit{quasivictim}(A_i, B, \psi)\})$, $0 \le i < k$, let p denote a (A, ψ)-copy, and let j denote the maximum value of i such that p belongs to *Copies*(A_i). We define *quasibenefit*(A, B, p) as *min-benefit*(A_j, p). The copy p belongs to the set *Quasiprimary*(A, B) iff $j = k - 1$. We define *Quasisecondary*(A, B) as *Copies*(A) $-$ *Quasiprimary*(A, B). The copy p belongs to the set *Matched*(A, B) iff p belongs to *Min-matched*(A_j, B). We define *Unmatched*(A, B) as *Copies*(A) $-$ *Matched*(A, B).

The next six lemmas are analogous to Lemmas 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6, respectively. The proofs are all straightforward and hence are omitted.

Lemma 5.1 *Let A and B denote two comparable placements, let ψ denote an object such that $\mathit{count}(A, \psi) > 0$, and let P (resp., Q) denote the set of all (A, ψ)-copies in Matched(A, B) (resp., Unmatched(A, B)). Then if Q is empty (resp., nonempty), quasivictim(A, B, ψ) belongs to P (resp., Q) and for all p in P (resp., Q), $\mathit{quasibenefit}(A, B, \mathit{quasivictim}(A, B, \psi)) \le \mathit{quasibenefit}(A, B, p)$.* ■

Lemma 5.2 *Let A and B denote two comparable placements, let ψ denote an object such that $\mathit{count}(A, \psi) > 0$, and let P (resp., Q) denote the set of all (A, ψ)-copies in Matched(A, B) (resp., Unmatched(A, B)). Then a unique (A, ψ)-copy p belongs to Quasiprimary(A, B), p belongs to P unless P is empty, and $\mathit{quasibenefit}(A, B, p) \ge \mathit{quasibenefit}(A, B, q)$ for all q in P (resp., Q).* ■

Lemma 5.3 *Let α denote a λ-separated hierarchy for some $\lambda > 1$, let A and B denote two α-placements, and let p denote an (A, ψ)-copy in Copies(A). Then $\mathit{quasibenefit}(A, B, p) \le \frac{\lambda}{\lambda - 1} \cdot \mathit{value}(\alpha, \psi)$.* ■

Lemma 5.4 *(Delete) Let A and B denote two comparable placements, let ψ denote an object such that $\mathit{count}(A, \psi) > 0$, and let A' denote the placement $(\mathit{Hierarchy}(A), \mathit{Copies}(A) - \{\mathit{quasivictim}(A, B, \psi)\})$. Then the following claims hold for all p in Copies(A'): $\mathit{quasibenefit}(A, B, p) = \mathit{quasibenefit}(A', B, p)$; p belongs to Quasiprimary(A, B) (resp., Quasisecondary(A, B), Matched(A, B), Unmatched(A, B)) iff p belongs to to Quasiprimary(A', B) (resp., Quasisecondary(A', B), Matched(A', B), Unmatched(A', B)).* ■

Lemma 5.5 *(Insert) Let α denote a hierarchy, let A and B denote two α-placements, let ψ denote an object in Missing(A), let p denote the copy (α, ψ) and let A' denote the placement $(\alpha, \mathit{Copies}(A) \cup \{p\})$. Then p belongs to Quasiprimary(A', B), $\mathit{quasibenefit}(A', B, p) = \mathit{value}(\alpha, p)$, and the following claims hold for all q in Copies(A): $\mathit{quasibenefit}(A, B, q) = \mathit{quasibenefit}(A', B, q)$; q belongs to Quasiprimary(A, B) (resp., Quasisecondary(A, B), Matched(A, B), Unmatched(A, B)) iff q belongs to to Quasiprimary(A', B) (resp., Quasisecondary(A', B), Matched(A', B), Unmatched(A', B)).* ■

Lemma 5.6 *(Combine) Let α denote a non-singleton hierarchy with k children α_i, $0 \le i < k$, let A_i and B_i denote two α_i-placements, $0 \le i < k$, let A denote the placement $(\alpha, \cup_{0 \le i < k} \mathit{Copies}(A_i))$,*

Coupled Swapping Procedure: *Swaps*(A, B, C, p)

- Let ψ be such that p is a (A, ψ)-copy.
- Call *Swap*(A, p) and *Swap*$(B, \textit{quasivictim}(B, C, \psi))$.

Figure 4: The coupled swapping procedure. The input is a nice triple of placements (A, B, C) and a copy p in *Copies*(A). Because A and B are coupled, *candidate*(A) = *candidate*(B) and hence the same copy is inserted in both calls to the swapping procedure. It follows that the output triple (A, B, C) is nice.

let B denote the placement $(\alpha, \cup_{0\le i<k} \textit{Copies}(B_i))$, let P denote $\cup_{0\le i<k} \textit{Matched}(A_i, B_i)$, let Q denote $\cup_{0\le i<k} \textit{Unmatched}(A_i, B_i)$, let ψ denote an object such that $\textit{count}(A, \psi) > 0$, let p denote the unique (A, ψ)-copy in $\textit{Quasiprimary}(A, B)$, and let x denote $\textit{quasibenefit}(A, B, p) - \textit{value}(\alpha, \psi)$. Then the following claims hold: if P is nonempty then p belongs to P; $\textit{quasibenefit}(A_i, B, q) = x$; p belongs to $\textit{Matched}(A, B)$ (resp., $\textit{Unmatched}(A, B)$) iff $\textit{count}(B, \psi) > 0$ (resp., $\textit{count}(B, \psi) = 0$). Furthermore, for any i, $0 \le i < k$, and any (A_i, ψ)-copy q different from p, the following claims hold: q belongs to $\textit{Matched}(A, B)$ (resp., $\textit{Unmatched}(A, B)$) iff q belongs to P (resp., Q); $\textit{quasibenefit}(A, B, q) = \textit{quasibenefit}(A_i, B_i, q)$; if q belongs to $\textit{Quasisecondary}(A_i, B_i)$ then q belongs to $\textit{Quasisecondary}(A, B)$; if Q is empty or q belongs to Q then $\textit{quasibenefit}(A, B, q) \le x$. ■

In the following definition, let (A, B, C) denote a nice triple of placements, let $k = |\textit{Copies}(A)|$, let $A_0 = A$ and $B_0 = B$, let A_{i+1} denote the placement $(\textit{Hierarchy}(A), \textit{Copies}(A_i) - \{\textit{victim}(A_i)\})$, let B_{i+1} denote the placement $(\textit{Hierarchy}(B), \textit{Copies}(B_i) - \{\textit{quasivictim}(B_i, C, \psi)\})$ where ψ denotes the object associated with $\textit{victim}(A_i)$, $0 \le i < k$, let p denote a copy in $\textit{Copies}(B)$, and let j denote the maximum value of i such that p belongs to $\textit{Copies}(B_i)$. (Note that each triple (A_i, B_i, C), $0 \le i < k$, is nice.) We define $\textit{mate}(A, B, C, p)$ as $\textit{victim}(A_j)$.

The coupled swapping procedure of Figure 4 executes a pair of swaps, one involving each of the coupled placements in a given nice triple. Lemmas 5.7 and 5.8 below are useful for analyzing the effect of a call to the coupled swapping procedure for which the parameter p is either *victim*(A) or *secondary-victim*(A).

Lemma 5.7 *(Delete) Let p be equal to $\textit{victim}(A)$ (resp., $\textit{secondary-victim}(A)$) for some placement A belonging to a nice triple (A, B, C), let ψ be such that p is an (A, ψ)-copy, let A' denote the placement $(\textit{Hierarchy}(A), \textit{Copies}(A) - \{\textit{victim}(A)\})$, and let B' denote the placement $(\textit{Hierarchy}(B), \textit{Copies}(B) - \{\textit{quasivictim}(A, B, \psi)\})$. Then (A', B', C) is nice.*

Proof: Straightforward from Lemmas 4.4 and 5.4. ■

Lemma 5.8 *(Insert) Let (A, B, C) denote a nice triple of placements, let ψ denote an object in $\textit{Missing}(A)$, let A' and denote the placement $(\textit{Hierarchy}(A), \textit{Copies}(A) \cup \{(\textit{Hierarchy}(A), \psi)\})$, and let B' denote the placement $(\textit{Hierarchy}(B), \textit{Copies}(B) \cup \{(\textit{Hierarchy}(B), \psi)\})$. Then (A', B', C) is nice.*

Proof: Straightforward from Lemmas 4.5 and 5.5. ■

Lemma 5.9 *(Combine) Let α denote a non-singleton hierarchy with k children α_i, $0 \le i < k$, let A_i, B_i, and C_i denote α_i-placements such that (A_i, B_i, C_i) is nice, $0 \le i < k$, and let A, B, and C denote the placements $(\alpha, \cup_{0\le i<k} \textit{Copies}(A_i))$, $(\alpha, \cup_{0\le i<k} \textit{Copies}(B_i))$, and $(\alpha, \cup_{0\le i<k} \textit{Copies}(C_i))$, respectively. Then (A, B, C) is nice.*

Bridging Algorithm

- **Combining.** Initialize A and Φ as in the amortizing algorithm. If α is singleton, initialize B and C to A. Otherwise, initialize *Hierarchy*(B) (resp., *Hierarchy*(C)) to α, and initialize *Copies*(B) (resp., *Copies*(C)) to $\cup_{0\le i<k}$*Copies*(B_i), where the B_i's (resp., C_i's) are the bridging (resp., arbitrary) placements previously computed at the k children of α.
- **Local Initialization.** Set Δ to 0 and Υ to *Missing*(A).
- **Amortized Swapping.** This step applies the same sequence of swaps to the amortizing placement A as in the amortizing algorithm. Each of these swaps is accompanied by a corresponding swap involving the bridging placement B. (For a proof that the latter swaps are well-defined, see Lemma 5.10 below.)

  1. Let $x = \mathit{benefit}(A, \mathit{secondary\text{-}victim}(A))$, let $y = \mathit{benefit}(A, \mathit{victim}(A))$, and let $z = \mathit{value}(\alpha, \mathit{candidate}(A))$.
  2. If $x - \Phi \le \min(y, z)$ then call $\mathit{Swaps}(A, B, C, \mathit{secondary\text{-}victim}(A))$, set Φ to $\max(0, \Phi - x)$, remove $\mathit{candidate}(A)$ from Υ, and goto line 1.
  3. If $y < z$, then call $\mathit{Swaps}(A, B, C, \mathit{victim}(A))$, add y to Δ, remove $\mathit{candidate}(A)$ from Υ, and goto line 1.

- **Accounting.** For each object ψ in Υ, add $\mathit{value}(\alpha, \psi)$ to Δ and remove ψ from Υ.
- **Potential Update.** Add Δ to Φ.

Figure 5: The bridging algorithm. We introduce this algorithm for the sole purpose of analyzing the amortizing algorithm. We assume that the children, if any, of a given hierarchy α have already been processed, and describe the computation associated with α.

Proof: Straightforward from Lemmas 4.6 and 5.6. ■

The bridging algorithm is presented in Figure 5. The program variables A, B, and C correspond to the amortizing, bridging, and arbitrary placements, respectively.

The following lemma implies that the swaps associated with the bridging placement are well-defined. Given this lemma, it is straightforward to prove that all of the placements associated with the bridging algorithm are feasible. Furthermore, it is straightforward to prove that the amortizing placement A computed by the bridging algorithm is the same as the placement computed by the amortizing algorithm.

Lemma 5.10 *After the combining step of the bridging algorithm, and after each iteration of the amortized swapping loop of the bridging algorithm,* (A, B, C) *is nice.*

Proof: Lemmas 5.7 and 5.8 imply that the claim cannot fail for the first time after an iteration of the amortized swapping loop. We now argue that the claim cannot fail for the first time after the combining step associated with some hierarchy α. If α is singleton, then A, B, and C are all equal and the claim follows. If α is non-singleton then (A_i, B_i, C_i) are nice, $0 \le i < k$, then Lemma 5.9 implies that (A, B, C) is nice. The lemma follows since no other part of the code modifies A, B, or C. ■

5.2 Cost comparison: amortizing versus bridging

In this section we compare the cost of the amortizing and bridging placements computed by the bridging algorithm for a given hierarchy.

For any multiset of reals X, we define $\mathit{sum}(X)$ as the sum of the elements of X. For any multiset of reals X, and any integer i such that $0 \le i \le |X|$, we define $\mathit{Big}(X, i)$ (resp., $\mathit{Little}(X, i)$) as the multiset consisting of the i largest (resp., smallest) reals in X. For any two multisets of reals X and Y such that $|X| = |Y|$, we write $X \le Y$ to mean that the ith largest element of X is less than or equal to the ith largest element of Y, $1 \le i \le |X|$. A triple (X, Y, Z) is defined to be *good* iff the following conditions hold: (i) X, Y, and Z are finite multisets of reals, (ii) $|X| + |Y| = |Z|$, (iii) $X \ge \mathit{Little}(Z, |X|)$, (iv) $Y \le \mathit{Big}(Z, |Y|)$, and (v) $\mathit{sum}(X) + \mathit{sum}(Y) \le \mathit{sum}(Z)$.

Lemma 5.11 *For any real x, the triples $(\{x\}, \{\}, \{x\})$ and $(\{\}, \{x\}, \{x\})$ are good.* ■

Lemma 5.12 *Let (X, Y, Z) be good and assume that $Z' = \mathit{Big}(Z, |Z| - 1)$. If X is empty, then let $X' = X$ and $Y' = \mathit{Big}(Y, |Y| - 1)$. Otherwise, let $X' = \mathit{Big}(X, |X| - 1)$ and $Y' = Y$. In either case, (X', Y', Z') is good.* ■

Lemma 5.13 *Let (X_i, Y_i, Z_i) be good, $1 \le i \le k$, and let $X = \cup_{1 \le i \le k} X_i$, $Y = \cup_{1 \le i \le k} Y_i$, and $Z = \cup_{1 \le i \le k} Z_i$. Then (X, Y, Z) is good. Furthermore, if Y is empty and x is the maximum element of X, then $(X - \{x\}, \{x\}, Z)$ is good.* ■

Lemma 5.14 *Let (X, Y, Z) be good and let w be a nonnegative real. Assume that z is a maximum element of Z and let $Z' = (Z - \{z\}) \cup \{w + z\}$. If Y is nonempty, then let $X' = X$ and $Y' = (Y - \{y\}) \cup \{w + y\}$ where y is a maximum element of Y. Otherwise, let $Y' = Y$ and $X' = (X - \{x\}) \cup \{w + x\}$ where x is a maximum element of X. In either case, (X', Y', Z') is good.* ■

Lemma 5.15 *Let A, B, and C denote the corresponding program variables after the combining step of the bridging algorithm, or after some iteration of the amortized swapping loop of the bridging algorithm. Let X, Y, and Z denote the multisets $\{\mathit{quasibenefit}(B, C, p) : p \in \mathit{Unmatched}(B, C)\}$, $\{\mathit{quasibenefit}(B, C, p) : p \in \mathit{Matched}(B, C)\}$, and $\{\mathit{benefit}(A, p) : p \in \mathit{Copies}(A)\}$, respectively. Then (X, Y, Z) is good.*

Proof: After the combining step, there are two cases to consider. If α is singleton, then $A = B = C$, $\mathit{Unmatched}(B, C)$ is empty, $\mathit{Matched}(B, C) = \mathit{Copies}(A)$, and $\mathit{benefit}(A, p) = \mathit{quasibenefit}(B, C, p)$ for all p in $\mathit{Copies}(A)$; the claim follows. If α is non-singleton, the claim follows by Lemmas 4.6, 5.6, 5.13, and 5.14. (Remark: Lemma 5.14 takes care of the increase in benefit/quasibenefit associated with the primary/quasiprimary copy of each object.)

It remains to consider the effect of the pair of swapping operations (one applied to A, the other to B) occurring in some iteration of the amortized swapping loop of the bridging algorithm. This pair of swaps can be viewed as a pair of deletions followed by a pair of insertions. Lemmas 4.4, 5.4, and 5.12 imply that the claim holds after the pair of deletions. Lemmas 4.5, 5.5, and 5.11 imply that the claim holds after the pair of insertions. ■

Lemma 5.16 *For any hierarchy α, let A (resp., B) denote the amortizing (resp., bridging) placement computed by the bridging algorithm. Then $\mathit{cost}(A) \le \mathit{cost}(B)$.*

Proof: Let c denote the cost of the empty α-placement, $(\alpha, \{\})$, and note that $\mathit{cost}(A)$ equals $c - \sum_{p \in \mathit{Copies}(A)} \mathit{benefit}(A, p)$ while $\mathit{cost}(B)$ equals $c - \sum_{p \in \mathit{Copies}(B)} \mathit{quasibenefit}(B, C, p)$. The claim now follows from Lemma 5.15 and condition (v) in the definition of a good triple. ■

5.3 Cost comparison: bridging versus arbitrary

In this section we compare the cost of the bridging and arbitrary placements computed by the bridging algorithm for a given hierarchy. The following lemma is useful for our analysis.

Lemma 5.17 *Let A, B, and C denote the corresponding program variables after the combining step of the bridging algorithm, or after some iteration of the amortized swapping loop of the bridging algorithm. Let q belong to Copies(B), let p = mate(A,B,C,q), let x = benefit(A,p), and let y = quasibenefit(B,C,q). If q belongs to Matched(B,C) (resp., Unmatched(B,C)) then $x \geq y$ (resp., $x \leq y$).*

Proof: Follows from Lemma 5.15 and conditions (iii) and (iv) in the definition of a good triple. ■

We now state our main technical lemma. The proof of this lemma is given in Section 6.

Lemma 5.18 *Let B (resp., C) denote the bridging (resp., arbitrary) placement computed by the bridging algorithm for a given λ-separated hierarchy. Then cost$(B) \leq (1 + \frac{3\lambda}{\lambda-1}) \cdot$ cost(C).* ■

Given that the arbitrary placement is potentially an optimal placement, we conclude that the cost of the bridging placement is within a constant factor of optimal.

5.4 The main theorem

Using Lemmas 5.16 and 5.18, we obtain the following result.

Lemma 5.19 *For any λ-separated hierarchy, the cost of the placement computed by the amortizing algorithm is at most $(1 + \frac{3\lambda}{\lambda-1})$ times optimal.* ■

Recall that while the placement A computed by the amortizing algorithm is not necessarily concrete, A can easily be refined to a concrete placement B such that cost$(B) \leq$ cost(A), as discussed at the beginning of Section 4. Lemma 5.19 assumes that the given hierarchy is λ-separated as defined in Section 4. If this is not the case, we first transform the given hierarchy into a λ-separated hierarchy as indicated in Section 4, introducing an extra factor of λ into the approximation bound.

Theorem 5.1 *For any hierarchy α and for any constant $\lambda > 1$, the cost of the placement computed by the amortizing algorithm is at most $\lambda \cdot (1 + \frac{3\lambda}{\lambda-1})$ times optimal.* ■

The above approximation ratio is less than 13.93 for the optimal choice of $\lambda = 1 + \sqrt{3}/2 \approx 1.866$.

6 Proof of the main technical lemma

In this section we prove Lemma 5.18 which compares the cost of the bridging and the arbitrary placements computed by the bridging algorithm. Throughout this section, we let A, B, and C denote the program variables of the bridging algorithm corresponding to the amortizing, bridging, and arbitrary placements. Note that by Lemma 5.10, the triple (A,B,C) is nice and therefore Hierarchy(A) = Hierarchy(B) = Hierarchy(C). We let α denote this common hierarchy, and furthermore if α is not a singleton we let α_i, $0 \leq i < k$, denote the children of α. For conciseness, the above notational conventions are assumed throughout this section without further repetition.

To facilitate the comparison of the bridging placement B with the arbitrary placement C, we introduce and maintain another comparable placement D that is closely related to C.

The rest of this section is organized as follows. First, in Section 6.1 we introduce the notions of "emulation" and "domination" to describe the relationship that we maintain between the placements B, C, and D. In Section 6.2 we list the variables used to specify the state of the computation. In Section 6.3 we list a number of invariants that are claimed to hold at particular points in the execution of the bridging algorithm, and prove that Lemma 5.18 follows from these invariants. Finally, in Sections 6.4 through 6.9 we examine how the state is affected by each step of the algorithm, and prove that the claimed invariants are indeed maintained.

Pruning Procedure: $Prune(D, B, p)$

- Let ψ be such that p is a (B, ψ)-*copy*.
- Update $Copies(D)$ by removing all copies of the form (β, ψ), where β is a descendant of $Region(B, p)$.

Figure 6: The pruning procedure. The input is two comparable placements D and B, and a copy p in $Copies(B)$ such that $p = quasivictim(B, D, \psi)$ belongs to $Matched(B, D)$. This procedure is used to modify the placement D whenever the placement B is modified by deleting the copy p from $Copies(B)$. (Remark: By Lemma 6.1, if D is a (B, C)-emulator, then $Matched(B, C) = Matched(B, C)$ and $quasivictim(B, C, \psi) = quasivictim(B, D, \psi)$.)

6.1 Emulation and domination

A placement D is said to be a (B, C)-*emulator* iff the following two conditions hold for all objects ψ and all descendants β of α: (i) if $count(restrict(C, \beta), \psi) = 0$ then $count(restrict(D, \beta), \psi) = 0$, and (ii) if $count(restrict(C, \beta), \psi) > 0$ and $count(restrict(B, \beta), \psi) > 0$ then $count(restrict(D, \beta), \psi) > 0$. Note that C itself is a (B, C)-emulator. Furthermore, if D is a (B, C)-emulator, $count(restrict(C, \beta), \psi) > 0$, and $count(restrict(B, \beta), \psi) = 0$, then there is no requirement on $count(restrict(D, \beta), \psi)$. The next two lemmas are straightforward and are stated without proof.

Lemma 6.1 *If D is a (B, C)-emulator, then $Matched(B, C) = Matched(B, D)$ and $Unmatched(B, C) = Unmatched(B, D)$.* ■

Lemma 6.2 *If D is a (B, C)-emulator, then $|Unmatched(B, C)| \geq |Copies(B)| - |Copies(D)|$.* ■

The placement B is said to *dominate* a comparable placement D iff the following condition holds for all objects ψ and for all descendants β of α: If $count(restrict(B, \beta), \psi) = 0$ then $count(restrict(D, \beta), \psi) = 0$. The following lemma is immediate.

Lemma 6.3 *If B dominates D, then $cost(B) \leq cost(D)$.* ■

We now introduce a slightly weakened version of domination. Given a set S of objects, the placement B is said to "S-dominate" a comparable placement D iff the following two conditions hold for all objects ψ and for all descendants β of α: if ψ does not belong to S and $count(restrict(B, \beta), \psi) = 0$, then $count(restrict(D, \beta), \psi) = 0$; if ψ belongs to S then $count(B, \psi) = 0$, $count(D, \psi) = 1$, and the unique copy of ψ in $Copies(D)$ is (α, ψ). Note that the placement B dominates D iff B $\emptyset$-dominates D. The following three lemmas will be used to prove that certain emulation and domination properties are preserved during the execution of the bridging algorithm; these lemmas are straightforward and are stated without proof.

Lemma 6.4 *(Insert) Let S be an arbitrary set of objects such that B S-dominates D and D is a (B, C)-emulator, let ψ be any object in $Missing(B)$ such that either $\psi \in S$ or $count(C, \psi) = 0$, let p denote the copy (α, ψ), and let B' denote the placement $(\alpha, Copies(B) \cup \{p\})$. Then D is a (B', C)-emulator, and if $\psi \notin S$ (resp., $\psi \in S$) then B' S-dominates (resp., $(S - \{\psi\})$-dominates) D.* ■

Lemma 6.5 *(Delete) Let S be an arbitrary set of objects such that B S-dominates D and D is a (B, C)-emulator. For any object ψ such that there is at least one (B, ψ)-copy, let p denote $quasivictim(B, C, \psi)$ and let B' denote $(\alpha, Copies(B) - \{p\})$. If p is in $Unmatched(B, C)$, then let $D' = D$; otherwise, let*

D' denote the new value of D after a call to $Prune(D, B, p)$. Then D' is a (B', C)-emulator and B' S-dominates D'. Furthermore, if p belongs to $Matched(B, C)$, then $|Copies(D')| \leq |Copies(D)| - 1$ and $cost(D') \leq cost(D) + quasibenefit(B, C, p)$. ■

Lemma 6.6 *(Combine) Assume that α is non-singleton and let B_i, C_i, and D_i denote three α_i-placements, $0 \leq i < k$, such that D_i is a (B_i, C_i)-emulator and B_i dominates D_i. Assume that B, C, and D are equal to the placements $(\alpha, \cup_{0 \leq i < k} Copies(B_i))$, $(\alpha, \cup_{0 \leq i < k} Copies(C_i))$, and $(\alpha, \cup_{0 \leq i < k} Copies(D_i))$, respectively. Further, let D' denote the placement $(\alpha, Copies(D) \cup P)$ where P is the set of copies $\{(\alpha, \psi) : count(D, \psi) = 0 \wedge count(C, \psi) > 0\}$. Then, D' is a (B, C)-emulator and B S-dominates D', where $S = \{\psi : count(B, \psi) = 0 \wedge count(D', \psi) > 0\}$.* ■

6.2 State variables

In this section we specify the variables that are used to capture the state of the computation. In addition to the program variables that appear in the pseudocode of Section 5.1, we also define a number of auxiliary variables. Each auxiliary variable is classified as either independent or dependent. We modify the values of the independent variables explicitly, in effect augmenting the pseudocode. The value of each dependent auxiliary variable is determined by the values of the program variables and the independent auxiliary variables. Below is a list of all state variables.

1. **Program variables**: Placements A, B, and C; potential Φ; change in potential Δ; the set of objects Υ.

2. **Independent auxiliary variables**: (i) the placement D; (ii) the nonnegative reals *deficit*, *surplus*, *newdeficit*, and *newsurplus*; (iii) the nonnegative integers *numdead* and *numlift*; (iv) a color, either red or blue, for each copy in $Copies(B)$.

3. **Dependent auxiliary variables**: (i) *numred*, the number of copies that are colored red; (ii) *threshold*, defined as $\min(\max(0, x - \Phi), y)$ where x is $benefit(A, secondary\text{-}victim(A))$ and y is $benefit(A, victim(A))$; (iii) the two sets of objects $\Upsilon_U = \Upsilon \cap \{\psi : count(C, \psi) = 0\}$ and $\Upsilon_M = \Upsilon - \Upsilon_U$; (iv) the six sets of copies $\hat{P}$, $\hat{Q}$, $\hat{R}$, $\check{P}$, $\check{Q}$, and $\check{R}$.

   The last six sets partition the set $Copies(B)$ as follows: $\hat{P}$ (resp., $\check{P}$) is the set of all blue copies in $Matched(B, C)$ that belong to $Quasiprimary(B, C)$ (resp., $Quasisecondary(B, C)$); $\hat{Q}$ (resp., $\check{Q}$) is the set of all blue copies in $Unmatched(B, C)$ that belong to $Quasiprimary(B, C)$ (resp., $Quasisecondary(B, C)$); $\hat{R}$ (resp., $\check{R}$) is the set of all red copies in $Unmatched(B, C)$ that belong to $Quasiprimary(B, C)$ (resp., $Quasisecondary(B, C)$). It is worth remarking that our coloring mechanism guarantees that none of the copies in $Matched(B, C)$ are colored red, and hence the above six sets alone partition the set $Copies(B)$.

6.3 The invariant properties

In this section, we list certain properties that are claimed to hold at various points in the execution of the bridging algorithm.

Invariant 6.1 *The following properties hold after the combining step of the bridging algorithm: (i) B S-dominates D, where $S = \{\psi : count(B, \psi) = 0 \wedge count(C, \psi) > 0\}$; (ii) D is a (B, C)-emulator; (iii) $|Copies(D)| \leq |Copies(B)| - numdead$; (iv) $numred \leq numdead$; (v) $(1 + \frac{3\lambda}{\lambda - 1}) cost(C) \geq cost(D) - deficit + surplus + \sum_{r \in \check{Q} \cup \hat{Q}} 3 \cdot quasibenefit(B, C, r) + \sum_{r \in \hat{R}} 2 \cdot quasibenefit(B, C, r) + \sum_{\psi \in T} 3 \cdot value(\alpha, \psi)$, where $T = \{\psi : count(B, \psi) = 0 \wedge count(C, \psi) = 0\}$; (vi) $deficit \leq \min(\Phi, \sum_{r \in \check{Q}} quasibenefit(B, C, r))$; (vii) $surplus \geq \Phi - deficit$.*

Invariant 6.2 *The following properties hold after the local initialization step, after each iteration of the amortized swapping loop, and after each iteration of the accounting loop of the bridging algorithm: (i) B Υ_M-dominates D; (ii) D is a (B, C)-emulator; (iii) $|Copies(D)| \leq |Copies(B)| - (numdead + numlift)$; (iv) $numred \leq numdead$; (v) $(1 + \frac{3\lambda}{\lambda-1}) cost(C) \geq cost(D) - deficit + surplus - newdeficit + newsurplus + \sum_{r \in \check{Q} \cup \hat{Q}} 3 \cdot quasibenefit(B, C, r) + \sum_{r \in \hat{R}} 2 \cdot quasibenefit(B, C, r) + \sum_{\psi \in \Upsilon_U} 3 \cdot value(\alpha, \psi)$; (vi) $deficit \leq \min(\Phi, \sum_{r \in \check{Q}} quasibenefit(B, C, r))$; (vii) $surplus \geq \Phi - deficit$; (viii) $numlift = |\{\psi : count(B, \psi) = 0 \wedge count(D, \psi) = 0 \wedge count(C, \psi) > 0\}|$; (ix) $newdeficit \leq \Delta$; (x) $newdeficit \leq numlift \cdot threshold$; (xi) $newsurplus \geq \Delta - newdeficit$.*

Invariant 6.3 *The following properties hold after the potential update step of the bridging algorithm: (i) B dominates D; (ii) D is a (B, C)-emulator; (iii) $|Copies(D)| \leq |Copies(B)| - (numdead + numlift)$; (iv) $numred \leq numdead$; (v) $(1 + \frac{3\lambda}{\lambda-1}) cost(C) \geq cost(D) - deficit + surplus - newdeficit + newsurplus + \sum_{r \in \check{Q} \cup \hat{Q}} 3 \cdot quasibenefit(B, C, r)$; (vi) $deficit \leq \min(\Phi - \Delta, \sum_{r \in \check{Q}} quasibenefit(B, C, r))$; (vii) $surplus \geq \Phi - \Delta - deficit$; (viii) $numlift = |\{\psi : count(D, \psi) = 0 \wedge count(C, \psi) > 0\}|$; (ix) $newdeficit \leq \Delta$; (x) $newdeficit \leq numlift \cdot w$, where w is such that each $r \in \hat{Q}$ satisfies $quasibenefit(B, C, r) \geq w$ and each $r \in \check{Q}$ satisfies $quasibenefit(B, C, r) - deficit \geq w$; (xi) $newsurplus \geq \Delta - newdeficit$.*

The above invariants are established in Sections 6.4 through 6.9. Our main technical lemma follows from Invariant 6.3.

Proof of Lemma 5.18: Property (i) of Invariant 6.3 along with Lemma 6.3 implies that $cost(B) \leq cost(D)$. Moreover the fact that $cost(D) \leq (1 + \frac{3\lambda}{\lambda-1}) cost(C)$ follows from property (v) of Invariant 6.3 along with the following two inequalities: (a) $deficit \leq \sum_{r \in \check{Q}} quasibenefit(B, C, r)$ and (b) $newdeficit \leq \sum_{r \in \check{Q} \cup \hat{Q}} quasibenefit(B, C, r)$.

Inequality (a) above follows directly from property (vi). For inequality (b), we use Lemma 6.2 to deduce that $|Unmatched(B, C)|$ is at least $|Copies(B)| - |Copies(D)|$, which in turn is at least $numred + numlift$ by properties (iii) and (iv). Therefore, $|\check{Q} \cup \hat{Q}|$, the number of blue copies in $Unmatched(B, C)$ is at least $numlift$. Moreover property (x) implies that for each $r \in \check{Q} \cup \hat{Q}$, $quasibenefit(B, C, r) \geq w$, and hence $\sum_{r \in \check{Q} \cup \hat{Q}} quasibenefit(B, C, r) \geq numlift \cdot w \geq newdeficit$. ■

6.4 Local initialization step

In this section, we assume that Invariant 6.1 holds before the local initialization step, and prove that Invariant 6.2 holds after the local initialization step.

- *State change.* Initialize Υ to $Missing(B)$, and set $\Delta = newdeficit = newsurplus = numlift = 0$.
- *Dependent variables.* $\Upsilon_U = \Upsilon \cap \{\psi : count(C, \psi) = 0\}$ and $\Upsilon_M = \Upsilon \cap \{\psi : count(C, \psi) > 0\}$.
- *Analysis.* All properties of Invariant 6.2 either follow from the corresponding properties of Invariant 6.1 or are trivially satisfied. Note that Υ_M is equal to the set S in property (i) of Invariant 6.1 and Υ_U is equal to the set T in property (v) of Invariant 6.1.

Thus Invariant 6.2 holds after the local initialization step.

6.5 Amortized swapping loop

In this section, we assume that Invariant 6.2 holds before an iteration of the amortized swapping loop, and prove that it holds after the iteration. We treat each swap as an insertion followed by a deletion. To facilitate this decomposition, we introduce a slightly stronger version of property (iii) which we call property (iii)′: $|Copies(D)| \leq |Copies(B)| - 1 - (numdead + numlift)$. We first show in Section 6.5.1 that Invariant 6.2 and property (iii)′ hold after the insertion. Then in Section 6.5.2 we show that Invariant 6.2 holds after the deletion.

6.5.1 Insertion

In this section, we assume that Invariant 6.2 holds before an insertion, and prove that Invariant 6.2 and property (iii)$'$ hold after the insertion. Let ψ denote the incoming object $\mathit{candidate}(B)$. For notational convenience, we use unprimed (resp., primed) symbols to denote the values of variables before (resp., after) the insertion. If the value of a variable does not change then we use the unprimed symbol throughout.

- *State change.* Let p denote the copy (α, ψ). Set $\mathit{Copies}(A') = \mathit{Copies}(A) \cup \{p\}$, $\mathit{Copies}(B') = \mathit{Copies}(B) \cup \{p\}$, and $\Upsilon' = \Upsilon - \{\psi\}$. Set the color of p to blue.
- *Dependent variables.* If $\mathit{count}(C, \psi) > 0$ then $\Upsilon'_M = \Upsilon_M - \{\psi\}$ and $\hat{P}' = \hat{P} \cup \{p\}$. Otherwise $\Upsilon'_U = \Upsilon_U - \{\psi\}$ and $\hat{Q}' = \hat{Q} \cup \{p\}$.
- *Inequalities.* By Lemma 5.5, $\mathit{quasibenefit}(B', C, p) = \mathit{value}(\alpha, \psi)$ and for all copies q in $\mathit{Copies}(B)$, $\mathit{quasibenefit}(B', C, q) = \mathit{quasibenefit}(B, C, q)$.
- *Analysis.* Properties (i) and (ii) follow from Lemma 6.4. Property (iii)$'$ follows since $|\mathit{Copies}(B)| = |\mathit{Copies}(B')| - 1$. Properties (iv) and (vii) through (xi) are not affected. Properties (v) and (vi) follow from the above inequalities.

Thus Invariant 6.2 and property (iii)$'$ hold after the insertion.

6.5.2 Deletion

In this section, we assume that Invariant 6.2 and property (iii)$'$ hold before a deletion, and prove that Invariant 6.2 holds after the deletion. Throughout this section, let q denote the copy deleted from A, let ψ denote the object associated with q, and let p denote $\mathit{quasivictim}(B, C, \psi)$. Note that q is either $\mathit{secondary\text{-}victim}(A)$ or $\mathit{victim}(A)$ and $q = \mathit{mate}(A, B, C, p)$. The copy p either belongs to $\mathit{Quasiprimary}(B, C)$ or $\mathit{Quasisecondary}(B, C)$, and accordingly q either belongs to $\mathit{Primary}(A)$ or $\mathit{Secondary}(A)$. The deletion is handled differently in each case. Each case is further split into three different subcases depending on the color of p and whether p is in $\mathit{Matched}(B, C)$ or not.

For notational convenience, we use unprimed (resp., primed) symbols to denote the values of variables before (resp., after) the deletion. If the value of a variable does not change then we use the unprimed symbol throughout. In all cases below we use the result of Lemma 5.4 that if $B' = (\alpha, \mathit{Copies}(B) - \{p\})$ then for every copy r in $\mathit{Copies}(B')$, $\mathit{quasibenefit}(B', C, r) = \mathit{quasibenefit}(B, C, r)$.

Case 1: $p \in \mathit{Quasisecondary}(B, C) = \check{R} \cup \check{Q} \cup \check{P}$

- *Remark.* In this case Δ, $\mathit{newdeficit}$, $\mathit{newsurplus}$, and $\mathit{numlift}$ do not change. Hence properties (ix), (x), and (xi) are not affected. However the placements B and D, along with the variables Φ, $\mathit{deficit}$, $\mathit{surplus}$, and $\mathit{numdead}$ may change. This affects properties (i) through (viii) and these have to be restored. Of these, properties (i) and (ii) follow from Lemma 6.5 while properties (iii), (iv), and (viii) are easy to verify. The crucial properties to establish are (iv), (v), (vi), and (vii).
- *State change.* Set $\Phi' = \max(0, \Phi - \mathit{benefit}(A, q))$.
- *Analysis.* Note that the RHS of (vii), as well as the first term in the RHS of (vi), is reduced by $\min(\Phi, \mathit{benefit}(A, q))$.
- **Case 1.1:** $p \in \check{R}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$.
  - *Dependent variables.* $\check{R}' = \check{R} - \{p\}$, and $\mathit{numred}' = \mathit{numred} - 1$.

- *Analysis.* Properties (i), (ii), (iii), and (viii) hold and are not affected further.
- **Case 1.1.1:** $\mathit{deficit} = 0$
  * *State change.* None.
  * *Analysis.* Properties (iv) and (v) are unaffected, while properties (vi) and (vii) are directly satisfied. Hence all properties hold.
- **Case 1.1.2:** $\mathit{deficit} > 0$
  * *Remark.* By property (vi) of Invariant 6.2, $\check{Q} \neq \emptyset$.
  * *State change.* Pick an arbitrary copy r from $\check{Q}$ and color it red. Set $\mathit{deficit}' = \max(0, \mathit{deficit} - \mathit{quasibenefit}(B, C, r))$ and $\mathit{surplus}' = \mathit{surplus} + \mathit{quasibenefit}(B, C, r)$.
  * *Dependent variables.* $\check{Q}' = \check{Q} - \{r\}$, $\check{R}' = \check{R} \cup \{r\}$, and $\mathit{numred}' = \mathit{numred}$.
  * *Inequalities.* By Lemma 5.17, $\mathit{quasibenefit}(B, C, r) \geq \mathit{benefit}(A, \mathit{mate}(A, B, C, r)) \geq \mathit{benefit}(A, \mathit{secondary\text{-}victim}(A)) = \mathit{benefit}(A, q)$.
  * *Analysis.* For (iv), note that the LHS and the RHS are both unchanged. For (v), note that the net change in the RHS is $-\mathit{deficit}' + \mathit{surplus}' + \mathit{deficit} - \mathit{surplus} - 3 \cdot \mathit{quasibenefit}(B, C, r)$, which is nonpositive. For (vi), note that the RHS is always nonnegative, and furthermore, if $\mathit{deficit}'$ is greater than zero then the decrease in the LHS is $\mathit{quasibenefit}(B, C, r))$ while the decrease in the RHS is no more than this. For (vii), note that the LHS increases by $\mathit{quasibenefit}(B, C, r)$, which is an upper bound on the cumulative increase in the RHS.

- **Case 1.2:** $p \in \check{Q}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$.
  - *Dependent variables.* $\check{Q}' = \check{Q} - \{p\}$.
  - *Analysis.* Properties (i), (ii), (iii), and (viii) hold and are not affected further. Property (iv) also holds since the LHS and the RHS are both unchanged.
  - **Case 1.2.1:** $\mathit{deficit} = 0$
    * *State change.* None.
    * *Analysis.* For (v) and (vii), note that the LHS is unchanged and the RHS only decreases. For (vi), note that the LHS is zero and the RHS remains nonnegative.
  - **Case 1.2.2:** $\mathit{deficit} > 0$
    * *State change.* Set $\mathit{deficit}' = \max(0, \mathit{deficit} - \mathit{quasibenefit}(B, C, p))$ and $\mathit{surplus}' = \mathit{surplus} + \mathit{quasibenefit}(B, C, p)$.
    * *Inequalities.* By Lemma 5.17, $\mathit{quasibenefit}(B, C, p) \geq \mathit{benefit}(A, q)$.
    * *Analysis.* For (v), note that the net change in the RHS is $-\mathit{deficit}' + \mathit{surplus}' + \mathit{deficit} - \mathit{surplus} - 3 \cdot \mathit{quasibenefit}(B, C, p)$, which is nonpositive. For (vi), note that the RHS is always nonnegative, and furthermore, if $\mathit{deficit}'$ is greater than zero then the decrease in the LHS is $\mathit{quasibenefit}(B, C, p))$ while the decrease in the RHS is no more than this. For (vii), note that the LHS increases by $\mathit{quasibenefit}(B, C, p)$, which is an upper bound on the cumulative increase in the RHS.

- **Case 1.3:** $p \in \check{P}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$. Set D' to the new value of D after a call to $\mathit{Prune}(D, B, p)$ and set $\mathit{numdead}' = \mathit{numdead} + 1$.
  - *Dependent variables.* $\check{P}' = \check{P} - \{p\}$.

- *Analysis.* Properties (i), (ii), and (iii) follow from Lemma 6.5 and are not affected further. Note that the RHS of (iv) increases by one while the RHS of (v) increases by $cost(D') - cost(D)$, which is at most $quasibenefit(B, C, p)$.
- **Case 1.3.1:** $deficit = 0$
  * **Case 1.3.1.1:** $surplus \geq quasibenefit(B, C, p)$
    · *State change.* Set $surplus' = surplus - quasibenefit(B, C, p)$.
    · *Inequalities.* By Lemma 5.17, $quasibenefit(B, C, p) \leq benefit(A, q)$.
    · *Analysis.* For (iv), note that the LHS does not change. For (v), note that the RHS now reduces by $quasibenefit(B, C, p)$ and hence the cumulative change is nonpositive. For (vi), note that the LHS is zero. For (vii), note that the LHS reduces by $quasibenefit(B, C, p)$ while the RHS either becomes zero or reduces by at least $benefit(A, q)$.
  * **Case 1.3.1.2:** $surplus < quasibenefit(B, C, p)$
    · *Remark.* By Lemma 6.2 and property (ii), $|Unmatched(B', C)|$ is at least $|Copies(B')| - |Copies(D')|$, which is at least $numdead' \geq numred + 1$, using properties (iii) and (iv). Hence there exists at least one blue copy in $|Unmatched(B, C)|$ and therefore $|\hat{Q} \cup \check{Q}| \geq 1$.
    · *State change.* Pick an arbitrary copy $r \in \hat{Q} \cup \check{Q}$ and color it red. Set $surplus' = 0$.
    · *Dependent variables.* If r is in $Quasiprimary(B, C)$, then $\hat{Q}' = \hat{Q} - \{r\}$ and $\hat{R}' = \hat{R} \cup \{r\}$. Otherwise $\check{Q}' = \check{Q} - \{r\}$ and $\check{R}' = \check{R} \cup \{r\}$. Moreover $numred' = numred + 1$.
    · *Inequalities.* By Lemma 5.17, $quasibenefit(B, C, r) \geq benefit(A, mate(A, B, C, r)) \geq benefit(A, victim(A)) \geq benefit(A, secondary\text{-}victim(A)) - \Phi = benefit(A, q) - \Phi \geq quasibenefit(B, C, p) - \Phi$. So $quasibenefit(B, C, r) + \Phi \geq quasibenefit(B, C, p)$, and hence by property (vii), $quasibenefit(B, C, r) + surplus \geq quasibenefit(B, C, p)$. Moreover, $\Phi' = 0$, because by property (vii) and the preconditions for case 1.3.1.2, $\Phi \leq surplus < quasibenefit(B, C, p) \leq benefit(A, q)$.
    · *Analysis.* For (iv), note that the LHS and the RHS both increase by one. For (v), note that the RHS further reduces by at least $quasibenefit(B, C, r) + surplus$ which is at least $quasibenefit(B, C, p)$ by the above inequality; it follows that the net change in the RHS of (v) is nonpositive. For (vi), note that the LHS is zero. For (vii), note that the LHS and the RHS are both zero.
- **Case 1.3.2:** $deficit > 0$
  * *Remark.* By property (vi), $\check{Q} \neq \emptyset$.
  * *State change.* Pick an arbitrary copy r from $\check{Q}$ and color it red. Set $deficit' = \max(0, deficit - quasibenefit(B, C, r))$ and $surplus' = surplus + quasibenefit(B, C, r)$.
  * *Dependent variables.* $\check{Q}' = \check{Q} - \{r\}$, $\check{R}' = \check{R} \cup \{r\}$, and $numred' = numred + 1$.
  * *Inequalities.* By Lemma 5.17, $quasibenefit(B, C, r) \geq benefit(A, mate(A, B, C, r)) \geq benefit(A, secondary\text{-}victim(A)) = benefit(A, q) \geq quasibenefit(B, C, p)$.
  * *Analysis.* For (iv), note that the LHS and the RHS both increase by one. For (v), note that the new change in the RHS is $-deficit' + surplus' + deficit - surplus - 3 \cdot quasibenefit(B, C, r)$, which is at most $-quasibenefit(B, C, p)$; it follows that the net change in the RHS is nonpositive. For (vi), note that the RHS is always nonnegative, and furthermore, if $deficit'$ is greater than zero then the decrease in the LHS is $quasibenefit(B, C, r))$ while the decrease in the RHS is no more than this. For (vii), note that the LHS increases by $quasibenefit(B, C, r)$, which is an upper bound on the cumulative increase in the RHS.

Case 2: $p \in \mathit{Quasiprimary}(B,C) = \hat{R} \cup \hat{Q} \cup \hat{P}$

- *Remark.* In this case, Φ, *deficit*, *surplus*, and *numdead* do not change and *numred* can only decrease. Hence properties (iv), (vi), and (vii) are not affected at all. However the placements B and D, along with the variables Δ, *newdeficit*, *newsurplus*, and *numlift* may change. Hence the remaining properties may change and have to be restored. Of these, (i) and (ii) will again follow from Lemma 6.5 while (iii) and (viii) are easy to verify. The crucial properties to establish are (v), (ix), (x), and (xi).
- *State change.* Set $\Delta' = \Delta + \mathit{benefit}(A,q)$.
- *Analysis.* Note that the RHS of (ix) and (xi) increases by $\mathit{benefit}(A,q)$.
- **Case 2.1:** $p \in \hat{R}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$. Set $\mathit{newsurplus}' = \mathit{newsurplus} + \mathit{quasibenefit}(B,C,p)$.
  - *Dependent variables.* $\hat{R}' = \hat{R} - \{p\}$ and $\mathit{numred}' = \mathit{numred} - 1$.
  - *Inequalities.* By Lemma 5.17, $\mathit{quasibenefit}(B,C,p) \geq \mathit{benefit}(A,q)$.
  - *Analysis.* Properties (i), (ii), (iii), (viii), and (ix) are easy to verify. For (v), note that the change in the RHS is $\mathit{newsurplus}' - \mathit{newsurplus} - 2 \cdot \mathit{quasibenefit}(B,C,p)$, which is nonpositive. For (x), note that there is no change in the LHS while the RHS can only increase. For (xi), note that the LHS increases by $\mathit{quasibenefit}(B,C,p)$ while the RHS increases by $\mathit{benefit}(A,q)$.
- **Case 2.2:** $p \in \hat{Q}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$. Set $\mathit{newsurplus}' = \mathit{newsurplus} + \mathit{quasibenefit}(B,C,p)$.
  - *Dependent variables.* $\hat{Q}' = \hat{Q} - \{p\}$.
  - *Inequalities.* By Lemma 5.17, $\mathit{quasibenefit}(B,C,p) \geq \mathit{benefit}(A,q)$.
  - *Analysis.* Properties (i), (ii), (iii), (viii), and (ix) are easy to verify. For (v), note that the change in the RHS is $\mathit{newsurplus}' - \mathit{newsurplus} - 3 \cdot \mathit{quasibenefit}(B,C,p)$, which is nonpositive. For (x), note that there is no change in LHS while the RHS can only increase. For (xi), note that the LHS increases by $\mathit{quasibenefit}(B,C,p)$ while the RHS increases by $\mathit{benefit}(A,q)$.
- **Case 2.3:** $p \in \hat{P}$
  - *State change.* Set $\mathit{Copies}(A') = \mathit{Copies}(A) - \{q\}$ and $\mathit{Copies}(B') = \mathit{Copies}(B) - \{p\}$. Set D' to the new value of D after a call to $\mathit{Prune}(D,B,p)$. Set $\mathit{numlift}' = \mathit{numlift} + 1$ and $\mathit{newdeficit}' = \mathit{newdeficit} + \mathit{benefit}(A,q)$.
  - *Dependent variables.* $\hat{P}' = \hat{P} - \{p\}$.
  - *Inequalities.* By Lemma 5.17, $\mathit{quasibenefit}(B,C,p) \leq \mathit{benefit}(A,q)$ and $\mathit{benefit}(A,q) = \mathit{threshold}$. By Lemma 6.5, $\mathit{cost}(D') - \mathit{cost}(D) \leq \mathit{quasibenefit}(B,C,p)$.
  - *Analysis.* Properties (i), (ii), (iii), and (viii) hold. For (v), note that the change in RHS is $\mathit{cost}(D') - \mathit{cost}(D) - \mathit{newdeficit}' + \mathit{newdeficit}$, which is at most $\mathit{quasibenefit}(B,C,p) - \mathit{benefit}(A,q)$ and hence is nonpositive. For (ix), note that the LHS and the RHS both change by the same amount. For (x), note that the LHS increases by $\mathit{benefit}(A,q)$ while the RHS increases by $\mathit{threshold}$. For (xi), note that the net change in the RHS is zero while the LHS is unchanged.

Thus Invariant 6.2 holds after an iteration of the amortized swapping loop.

6.6 Accounting loop

In this section, we assume that Invariant 6.2 holds before an iteration of the accounting loop, and prove that it holds after the iteration. Let ψ denote the object that is removed from the set Υ in this iteration. For notational convenience, we use unprimed (resp., primed) symbols to denote the values of variables before (resp., after) the iteration. If the value of a variable does not change then we use the unprimed symbol throughout.

- *Remark.* Since $\Upsilon = \Upsilon_U \cup \Upsilon_M$, ψ belongs to either Υ_U or Υ_M. The placement B and the variables Φ, *deficit*, *surplus*, *numdead*, and *numred* do not change here. Hence properties (iv), (vi), and (vii) are unaffected. However the placement D and the variables Δ, *newdeficit*, *newsurplus*, and *numlift* may change. This affects the other properties and they have to be restored.
- *State change.* Set $\Upsilon' = \Upsilon - \{\psi\}$ and $\Delta' = \Delta + \mathit{value}(\alpha, \psi)$.
- *Analysis.* Note that the RHS of (ix) and (xi) increase by $\mathit{value}(\alpha, \psi)$.
- **Case 1:** $\psi \in \Upsilon_U$
  - *Dependent variables.* $\Upsilon'_U = \Upsilon_U - \{\psi\}$.
  - *State change.* Set $\mathit{newsurplus}' = \mathit{newsurplus} + \mathit{value}(\alpha, \psi)$.
  - *Analysis.* Properties (i), (ii), (iii), (viii), (ix), (x), and (xii) are easy to verify. For (v), note that the change in the RHS is $\mathit{newsurplus}' - \mathit{newsurplus} - 3 \cdot \mathit{value}(\alpha, \psi)$, which is nonpositive. For (xi), note that the LHS and the RHS each increase by the same amount.
- **Case 2:** $\psi \in \Upsilon_M$
  - *Remark.* We have $\mathit{count}(B, \psi) = 0$ and, by property (i), $\mathit{count}(D, \psi) = 1$ where the corresponding copy is (α, ψ). Furthermore, the definition of Υ_M implies that $\mathit{count}(C, \psi) > 0$.
  - *Dependent variables.* $\Upsilon'_M = \Upsilon_M - \psi$.
  - *State change.* Set $\mathit{Copies}(D') = \mathit{Copies}(D) - \{p\}$, where $p = (\alpha, \psi)$. Set $\mathit{newdeficit}' = \mathit{newdeficit} + \mathit{value}(\alpha, \psi)$ and $\mathit{numlift}' = \mathit{numlift} + 1$.
  - *Inequalities.* We have $\mathit{cost}(D') - \mathit{cost}(D) = \mathit{value}(\alpha, \psi)$, and moreover since the object ψ was not swapped in, we have $\mathit{value}(\alpha, \psi) \leq \mathit{threshold}$.
  - *Analysis.* The modification of D preserves property (i) with respect to Υ'_M, while property (ii) is unaffected. Properties (iii) and (viii) hold due to the increase in *numlift*. For (v), note that the change in the RHS is $\mathit{cost}(D') - \mathit{cost}(D) - \mathit{newdeficit}' + \mathit{newdeficit}$, which is zero. For (ix), note that the LHS and the RHS both increase by the same amount. For (x), note that the LHS increases by $\mathit{value}(\alpha, \psi)$ while the RHS increases by *threshold*. For (xi), note that the net change in the RHS is zero while the LHS is unchanged.

Thus Invariant 6.2 holds after an iteration of the accounting loop.

6.7 Potential update step

In this section, we assume that Invariant 6.2 holds before the potential update step, and prove that Invariant 6.3 holds after the step. We use unprimed (resp., primed) symbols to denote the values of variables before (resp., after) the potential update step. If the value of a variable does not change then we use the unprimed symbol throughout.

- *Remark.* Note that $\Upsilon = \Upsilon_U = \Upsilon_M = \emptyset$.

- *Analysis.* Property (i) of Invariant 6.3 follows from property (i) of Invariant 6.2 since $\Upsilon_M = \emptyset$. Properties (ii), (iii), (iv), (viii), (ix), and (xi) are same as the corresponding properties of Invariant 6.2 and hold without change. Property (v) of Invariant 6.3 follows by dropping the last two summation terms of property (v) in Invariant 6.2.

- *State change.* Set $w = \mathit{threshold}$.

- *Analysis.* Property (x) of Invariant 6.3 follows from property (x) of Invariant 6.2. By Lemma 5.17, each r in $\hat{Q}$ satisfies $\mathit{quasibenefit}(B, C, r) \geq \mathit{benefit}(A, \mathit{mate}(A, B, C, r)) \geq \mathit{benefit}(A, \mathit{victim}(A)) \geq \mathit{threshold} = w$. The same lemma implies that each copy r in $\check{Q}$ satisfies $\mathit{quasibenefit}(B, C, r) - \Phi \geq \mathit{benefit}(A, \mathit{mate}(A, B, C, r)) - \Phi \geq \mathit{benefit}(A, \mathit{secondary\text{-}victim}(A)) - \Phi \geq \mathit{threshold} = w$, and then by property (vi) of Invariant 6.2, it follows that $\mathit{quasibenefit}(B, C, r) - \mathit{deficit} \geq w$.

- *State change.* Set $\Phi' = \Phi + \Delta$.

- *Analysis.* Properties (vi) and (vii) follow from the corresponding properties of Invariant 6.2.

Thus Invariant 6.3 holds after the potential update step.

6.8 Combining step at a singleton hierarchy

In this section, we establish the base case of our proof by showing that Invariant 6.1 holds after the combining step at a singleton hierarchy α.

- *State change.* Initialize A, B, C, and D to the same arbitrary α-placement and set $\Phi = \mathit{deficit} = \mathit{surplus} = \mathit{numdead} = 0$. Set the color of all the copies in B to blue.

- *Dependent variables.* Initialize the six dependent sets $\hat{R}$, $\hat{Q}$, $\hat{P}$, $\check{R}$, $\check{Q}$, $\check{P}$ using B, C, and the colors of the copies in B. Set $\mathit{numred} = 0$.

- *Inequalities.* We have $\mathit{cost}(C) = \mathit{cost}(D) \geq \sum_{\psi \in T} \mathit{value}(\alpha, \psi)$, where $T = \{\psi : \mathit{count}(B, \psi) = \mathit{count}(C, \psi) = 0\}$. Moreover, $\check{Q} = \hat{Q} = \hat{R} = \emptyset$.

- *Analysis.* Properties (i) and (ii) follow directly from the definitions of domination and emulation, while property (iii) follows directly. Properties (iv), (vi), and (vii) hold trivially. Property (v) follows from the above inequalities.

Thus Invariant 6.1 holds after the combining step at a singleton hierarchy.

6.9 Combining step at a non-singleton hierarchy

In this section, we assume that α is a non-singleton hierarchy and that Invariant 6.3 holds after the potential update step at each child α_i, and prove that Invariant 6.1 holds after the combining step at α. (Recall that α_i, $0 \leq i < k$, denote the children of α.) We use the subscript i on the state variables to denote the variable computed at the child α_i.

1. *State change.* Initialize $A = (\alpha, \cup_{0 \leq i < k} \mathit{Copies}(A_i))$, $B = (\alpha, \cup_{0 \leq i < k} \mathit{Copies}(B_i))$, and $C = (\alpha, \cup_{0 \leq i < k} \mathit{Copies}(C_i))$. The color of the copies in $\mathit{Copies}(B)$ does not change during this union (i.e., each copy p in $\mathit{Copies}(B_i)$ retains the same color when it enters B.) Initialize $\mathit{numdead} = \sum_{0 \leq i < k} \mathit{numdead}_i$.

2. *Dependent variables.* Variable $\mathit{numred} = \sum_{0 \leq i < k} \mathit{numred}_i$.

3. *State change.* Initialize $D = (\alpha, \cup_{0 \leq i < k} \mathit{Copies}(D_i))$ and then for each object ψ such that $\mathit{count}(C, \psi) > 0$ and $\mathit{count}(D, \psi) = 0$, add a copy (ψ, u) to $\mathit{Copies}(D)$.

4. *Analysis.* Lemma 6.6 guarantees that properties (i) and (ii) of Invariant 6.1 hold after this modification. Property (iii) is established in Lemma 6.7 below. Property (iv) follows by using property (iv) of Invariant 6.3, and summing over all children of α. It remains to establish properties (v), (vi), and (vii) by choosing appropriate values for *deficit* and *surplus*. Before choosing these values, we first write down expressions to relate the $cost(C)$ to $\sum_{0\le i<k} cost(C_i)$, and the $cost(D)$ to $\sum_{0\le i<k} cost(D_i)$. Let $\hat{T}$ denote the set $\{\psi : count(C,\psi)=0\}$, or equivalently the set $\{\psi : count(D,\psi)=0\}$. Clearly,

$$\begin{aligned} cost(C) &= \sum_{0\le i<k} cost(C_i) + \sum_{\psi\in\hat{T}} value(\alpha,\psi), && (1)\\ cost(D) &= \sum_{0\le i<k} cost(D_i) + \sum_{\psi\in\hat{T}} value(\alpha,\psi). && (2)\end{aligned}$$

Moreover note that this set $\hat{T}$ is precisely the union of Υ_U, and the set of objects corresponding to copies in $\hat{R}$ and $\hat{Q}$. This in conjunction with Lemma 5.3 yields

$$\sum_{\psi\in\hat{T}} value(\alpha,\psi) \ge \sum_{\psi\in\Upsilon_U} value(\alpha,\psi) + \sum_{r\in\hat{Q}\cup\hat{R}} \frac{\lambda-1}{\lambda}\cdot quasibenefit(B,C,r). \qquad (3)$$

5. *Dependent variables.* Initialize the six dependent sets $\hat{R}$, $\hat{Q}$, $\hat{P}$, $\check{R}$, $\check{Q}$, $\check{P}$ using B, C, and the colors on the copies in B.

6. *Remark.* By Lemma 5.6, $\check{Q} = \cup_{0\le i<k}\check{Q}_i + \cup_{0\le i<k}\bar{Q}_i$ where $\bar{Q}_i = \hat{Q}_i - \hat{Q}$.

7. *State change.* For each i, $0\le i<k$, set $deficit'_i = deficit_i + newdeficit_i - (|\hat{Q}_i - \bar{Q}_i|)\cdot w_i$ and $surplus'_i = surplus_i + newsurplus_i + (|\hat{Q}_i - \bar{Q}_i|)\cdot w_i$.

8. *Analysis.* Using the above values of $deficit'_i$ and $surplus'_i$, we establish in Lemma 6.8 that

$$\begin{aligned} \left(1+\frac{3\lambda}{\lambda-1}\right) cost(C_i) &\ge cost(D_i) - deficit'_i + surplus'_i + \\ &\quad \sum_{r\in\check{Q}_i\cup\bar{Q}_i} 3\cdot quasibenefit(B_i,C_i,r), && (4)\\ deficit'_i &\le \min(\Phi_i, \sum_{r\in\check{Q}_i\cup\bar{Q}_i} quasibenefit(B_i,C_i,r)), && (5)\\ surplus'_i &\ge \Phi_i - deficit'_i. && (6)\end{aligned}$$

9. *State change.* Initialize Φ, *deficit*, and *surplus* to $\sum_{0\le i<k}\Phi_i$, $\sum_{0\le i<k} deficit'_i$, and $\sum_{0\le i<k} surplus'_i$, respectively.

10. *Analysis.* Summing Equations (4), (5), and (6) over all children α_i, $0\le i<k$, we obtain

$$\begin{aligned} \sum_{0\le i<k}\left(1+\frac{3\lambda}{\lambda-1}\right) cost(C_i) &\ge \sum_{0\le i<k} cost(D_i) - deficit + surplus + \\ &\quad \sum_{r\in\check{Q}} 3\cdot quasibenefit(B,C,r), && (7)\\ deficit &\le \min(\Phi, \sum_{r\in\check{Q}} quasibenefit(B,C,r)), && (8)\\ surplus &\ge \Phi - deficit. && (9)\end{aligned}$$

(Here we used the result of Lemma 5.6 that $\check{Q} = \cup_{0\le i<k}\check{Q}_i + \cup_{0\le i<k}\bar{Q}_i$, and that for each $r \in \check{Q}_i \cup \bar{Q}_i$, $\mathit{quasibenefit}(B_i, C_i, r) = \mathit{quasibenefit}(B, C, r)$.) Properties (vi) and (vii) are essentially the latter two equations. Finally, we establish property (v) using Equations (1), (2), (3), and (7) as follows:

$$\begin{aligned}
&\left(1+\frac{3\lambda}{\lambda-1}\right) \mathit{cost}(C) \\
=\ & \left(1+\frac{3\lambda}{\lambda-1}\right) \sum_{0\le i<k} \mathit{cost}(C_i) + \sum_{\psi\in\hat{T}} \left(1+\frac{3\lambda}{\lambda-1}\right) \mathit{value}(\alpha,\psi) \\
\ge\ & \sum_{0\le i<k} \mathit{cost}(D_i) - \mathit{deficit} + \mathit{surplus} + \sum_{r\in\check{Q}} 3\cdot \mathit{quasibenefit}(B,C,r) + \sum_{\psi\in\hat{T}} \left(1+\frac{3\lambda}{\lambda-1}\right) \mathit{value}(\alpha,\psi) \\
\ge\ & \mathit{cost}(D) - \mathit{deficit} + \mathit{surplus} + \sum_{r\in\check{Q}} 3\cdot \mathit{quasibenefit}(B,C,r) + \sum_{\psi\in\hat{T}} \left(\frac{3\lambda}{\lambda-1}\right) \mathit{value}(\alpha,\psi) \\
\ge\ & \mathit{cost}(D) - \mathit{deficit} + \mathit{surplus} + \sum_{r\in\check{Q}\cup\hat{Q}\cup\hat{R}} 3\cdot \mathit{quasibenefit}(B,C,r) + \sum_{\psi\in\Upsilon_U} 3\cdot \mathit{value}(\alpha,\psi).
\end{aligned}$$

(The above four equations follow by using Equations (1), (7), (2), and (3), respectively.)

11. *Conclusion.* Thus Invariant 6.1 holds after the combining step.

Lemma 6.7 *Suppose that properties (iii) and (viii) of Invariant 6.3 hold for each child α_i of α, and that the placements B, C, and D and the variable numdead are initialized as indicated in the preceding state change descriptions. Then property (iii) of Invariant 6.1 holds after the initialization.*

Proof: Note that $|\mathit{Copies}(B)| = \sum_{0\le i<k} |\mathit{Copies}(B_i)|$ and $|\mathit{Copies}(D)| = \sum_{0\le i<k} |\mathit{Copies}(D_i)| + |\hat{T}|$ where $\hat{T} = \{\psi : \mathit{count}(D,\psi) = 0 \ \wedge\ \mathit{count}(C,\psi) > 0\}$. Using property (iii) of Invariant 6.3 to upper bound $|\mathit{Copies}(D_i)|$, it follows that $|\mathit{Copies}(D)| \le \sum_{0\le i<k}(|\mathit{Copies}(B_i)| - \mathit{numdead}_i - \mathit{numlift}_i) + |\hat{T}| = |\mathit{Copies}(B)| - \mathit{numdead} + (|\hat{T}| - \sum_{0\le i<k} \mathit{numlift}_i)$. Moreover, $|\hat{T}| \le \sum_{0\le i<k} \mathit{numlift}_i$, since $\hat{T} \subseteq \cup_{0\le i<k}\hat{T}_i$, where $\hat{T}_i = \{\psi : \mathit{count}(D_i,\psi) = 0 \ \wedge\ \mathit{count}(C_i,\psi) > 0\}$, and by property (viii) of Invariant 6.3, $|\hat{T}_i| = \mathit{numlift}_i$. Thus property (iii) holds. ■

Lemma 6.8 *Suppose that Invariant 6.3 holds and let $\bar{Q}$ be an arbitrary subset of $\hat{Q}$. Let x and y be two new variables that are set to $\mathit{deficit} + \mathit{newdeficit} - |\hat{Q} - \bar{Q}|\cdot w$ and $\mathit{surplus} + \mathit{newsurplus} + |\hat{Q} - \bar{Q}|\cdot w$, respectively. Then the following three equations hold:*

$$\left(1+\frac{3\lambda}{\lambda-1}\right) \mathit{cost}(C) \ \ge\ \mathit{cost}(D) - x + y + \sum_{r\in\check{Q}\cup\bar{Q}} 3\cdot \mathit{quasibenefit}(B,C,r), \tag{10}$$

$$x \ \le\ \min(\Phi, \sum_{r\in\check{Q}\cup\bar{Q}} \mathit{quasibenefit}(B,C,r)), \tag{11}$$

$$y \ \ge\ \Phi - x. \tag{12}$$

Proof: Let j denote $|\hat{Q}| - |\bar{Q}|$. Property (x) of Invariant 6.3 implies that each $r \in \hat{Q}$ satisfies $\mathit{quasibenefit}(B,C,r) \ge w$. Therefore

$$-x + y \le -\mathit{deficit} + \mathit{surplus} - \mathit{newdeficit} + \mathit{newsurplus} + \sum_{r\in\hat{Q}-\bar{Q}} 3\cdot \mathit{quasibenefit}(B,C,r).$$

Equation (10) now follows from property (v) of Invariant 6.3. Equation (12) follows directly from properties (vii) and (x) of Invariant 6.3, since $y = \mathit{surplus} + \mathit{newsurplus} + j\cdot w \ge (\Phi - \Delta - \mathit{deficit}) + (\Delta - \mathit{newdeficit}) + j\cdot w = \Phi - x$.

It remains to establish Equation (11). The first part (i.e., $x \leq \Phi$), follows directly from properties (vi) and (viii), which imply that $\mathit{deficit} \leq \Phi - \Delta$ and $\mathit{newdeficit} \leq \Delta$, respectively. For the second part, we first establish that $|\check{Q} + \bar{Q}| \geq (\mathit{numlift} - j)$. By Lemma 6.2, $|\mathit{Unmatched}(B, C)|$ is at least $|\mathit{Copies}(B)| - |\mathit{Copies}(D)|$ which, by properties (iii) and (v) of Invariant 6.3, is at least $\mathit{numred} + \mathit{numlift}$. Hence the number of blue copies in $\mathit{Unmatched}(B, C)$ (i.e., $|\hat{Q}| + |\check{Q}|$) is at least $\mathit{numlift}$. It follows that $|\check{Q}| + |\bar{Q}| \geq \mathit{numlift} - j$. Now by property (x) of Invariant 6.3, each $r \in \check{Q}$ has $\mathit{quasibenefit}(B, C, r) - \mathit{deficit} \geq w$. Moreover, each $r \in \hat{Q}$ satisfies $\mathit{quasibenefit}(B, C, r) \geq w$. Therefore,

$$\sum_{r \in \check{Q} \cup \bar{Q}} \mathit{quasibenefit}(B, C, r) \geq \mathit{deficit} + (\mathit{numlift} - j) \cdot w \geq \mathit{deficit} + \mathit{newdeficit} - j \cdot w = x. \qquad (13)$$

(For the second inequality, we use property (ix) of Invariant 6.3.) This establishes Equation (11), thus proving the lemma. ■

7 An efficient distributed implementation

The main strength of the amortizing algorithm is that, in contrast with the flow-based algorithm of Section 3, it admits a fast distributed implementation. In this section, we briefly sketch such an implementation. The techniques employed are not particularly novel. The main point we wish to emphasize is that while the pseudocode of Section 4.2 may appear to be inherently sequential, in fact the algorithm is highly parallelizable.

The amortizing algorithm determines a placement for the given hierarchy α in a bottom-up manner. During the computation, the current placement, along with associated control information, is distributed across the nodes of the network. We begin by describing how this information is organized.

For each object ψ and each descendant β of α, we designate a node in β as the *manager* for ψ in β, denoted $\mathit{manager}(\beta, \psi)$. For load balancing purposes, this manager is chosen at random from β, where the probability of choosing a particular node u is $\mathit{size}(u)/\mathit{size}(\beta)$. For each descendant β of α, we choose a node uniformly at random from β and designate it as the *leader* of β, denoted $\mathit{leader}(\beta)$.

The current α-placement A is distributed across the nodes in α in the following natural manner. For every object ψ, the $\mathit{manager}(\alpha, \psi)$ stores $\mathit{frequency}(\alpha, \psi)$ along with a bit indicating whether ψ belongs to $\mathit{Missing}(A)$. If ψ belongs to $\mathit{Missing}(A)$, then $\mathit{manager}(\alpha, \psi)$ also stores $\mathit{value}(\alpha, \psi)$. Otherwise, for each copy $p = (\beta, \psi)$ in $\mathit{Copies}(A)$, the $\mathit{manager}(\beta, \psi)$ maintains the $\mathit{benefit}(A, p)$ along with a bit indicating whether p belongs to $\mathit{Primary}(A)$. The variables Φ and Δ are maintained by the leader of α. We note that a suitably random selection of managers and leaders ensures that the α-placement A is distributed across the nodes in α in a balanced manner with high probability.

As mentioned earlier, the amortizing algorithm pseudocode of Section 4.2 may appear to be inherently sequential. In particular, in the amortized swapping loop, the placement is modified one swap at a time, and the number of swaps could be large. Moreover, the desired swaps, which satisfy some "global" objective, are determined from a distributed placement. Fortunately, as we discuss below, all of the steps in the algorithm can be expressed in terms of instances of a simple prefix sum operation for any given node ordering. (We stress that for the prefix sum operations used in our implementation, the particular ordering of the nodes is not important. In fact, this ordering is allowed to change from one invocation of the operation to another. Consequently, the prefix sum operation can be efficiently implemented on any spanning tree.)

Let us now consider the process of computing a α-placement A at a non-singleton hierarchy α with k children α_i, $0 \leq i < k$. Let A_i denote the α_i-placement computed by the amortizing algorithm. The amortizing algorithm proceeds in four steps: the combining step, the local initialization step, the amortized swapping loop, and the potential update step.

In the combining step, A is set to $(\alpha, \cup_{0 \leq i < k} A_i)$. This requires no movement of copies in our distributed storage; however there are three computations that are implicit in the description of the algorithm in Section 4.2 that need to be performed. These are the calculation of $\mathit{benefit}(A, p)$ for each copy p in *Primary*(A), the calculation of *value*(α, ψ) for each object ψ in *Missing*(A), and the adjustment of the bit for each copy p in *Primary*(A_i) − *Primary*(A). Lemma 4.6 describes how these benefits and values change during the combining step. The computations for each object ψ are performed by *manager*(α, ψ), *manager*(α_i, ψ) for $0 \leq i < k$, and by *manager*(β, ψ) for each copy (β, ψ) in $\cup_{0 \leq i < k}$*Primary*(A_i). The computation of the potential Φ is a simple summation involving the leader of α and the leaders of the α_i's.

The local initialization step of the amortizing algorithm involves summation over a subset of nodes in the hierarchy α. This can be implemented efficiently using prefix sums in a straightforward manner.

We now turn to the amortized swapping loop. We implement this loop as an amortization phase followed by a swapping phase. (For the rest of this discussion, we say that a copy p is a *secondary* iff it belongs to *Secondary*(A).) Let X denote the list of secondaries in A, sorted according to their benefits in nonincreasing order. Let Φ denote the value of the potential after the combining step. In the amortization phase, we determine the largest prefix X' of X such that the sum of the benefits of copies in X' is at most Φ. (The copies in X' are guaranteed to be swapped out in the amortized swapping loop.) Thus, the amortization phase corresponds to a selection problem. Similarly, the swapping phase corresponds to the following abstract selection problem. Let X and Y denote two lists of numbers sorted in nondecreasing and nonincreasing orders, respectively. (The lists X and Y correspond to the benefits of the copies in the placement, and the values of the objects missing from the placement, respectively.) Our goal is to determine a largest prefix X' of X and a prefix Y' of Y such that $|X'|$ equals $|Y'|$ and no element of X' is greater than any element of Y'.

While the amortization and swapping phases are straightforward to perform sequentially, in the distributed setting the lists are partitioned across the nodes and thus are not available in sorted order. We would like to avoid explicitly sorting these lists, since sorting would require costly large-scale movement of list elements across the network. Moreover, we would like to perform the amortization and swapping steps "in place", that is, without moving the list elements. We now briefly describe an efficient distributed recursive implementation of the amortization phase; a similar approach can be used for the swapping phase. If $|X| \leq 1$ the problem is trivial. Otherwise, we first select a splitter among the secondaries in X. Second, we determine the set Z of those secondaries in X with benefit at most that of the splitter. Third, we sum up the benefits of the secondaries in Z. Finally, in a manner that depends on whether the sum exceeds the given potential Φ, we define a new instance of the problem with fewer elements and recurse. Standard probabilistic arguments can be used to establish a logarithmic bound on the depth of recursion. Moreover, each level of the recursion can be implemented using simple distributed operations such as broadcast and sum. We remark that a more efficient implementation can be obtained by using a large number of splitters to partition the list of secondaries.

The potential update step is straightforward to implement efficiently. In the computation of the placement for a hierarchy α, the combining step requires a constant number of messages per primary stored in the hierarchy and for each leader. The amortized swapping loop and the local initialization step each require $O(\log(\mathit{size}(\alpha)))$ prefix sum operations. The top-down pass that is used to convert the placement to a concrete placement can be easily implemented within the same complexity bounds as for the bottom-up amortizing algorithm.

References

[1] T. E. Anderson, M. D. Dahlin, J. N. Neefe, D. A. Patterson, D. S. Rosselli, and R. Y. Wang. Serverless network file systems. In *Proceedings of the 15th Symposium on Operating Systems Principles*, pages 109–126, 1995.

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

[3] B. Awerbuch, Y. Bartal, and A. Fiat. Heat & Dump: Competitive distributed paging. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 22–31, November 1993.

[4] B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 37:1021–1058, 1995.

[5] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184–193, October 1996.

[6] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 161–168, May 1998.

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

[8] M. A. Blaze. Caching in large-scale distributed file systems. Technical Report TR-397-92, Department of Computer Science, Princeton University, January 1993. PhD Thesis.

[9] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proceedings of the 2nd International World Wide Web Conference*, pages 763–771, October 1994.

[10] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997.

[11] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical Internet object cache. In *Proceedings of the USENIX 1996 Technical Conference*, pages 22–26, January 1996.

[12] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 106–113, May 1998.

[13] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey. The uncapacitated facility location problem. In P. Mirchandani and R. Francis, editors, *Discrete Location Theory*, pages 119–171. Wiley, New York, NY, 1990.

[14] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 267–280, November 1994.

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

[16] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proceedings of the 1998 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 254–265, August 1998.

[17] J. S. Gwertzman and M. Seltzer. The case for geographical push-caching. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, pages 51–57, May 1995.

[18] A. Heddaya and S. Mirdad. WebWave: Globally load balanced fully distributed caching of hot published documents. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 160–168, May 1997.

[19] S. Irani. Page replacement with multi-size pages and applications to Web caching. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 701–710, May 1997.

[20] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[21] M. R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. In *Proceedings of the IEEE Workshop on Internet Applications*, July 1999. To appear.

[22] A. Leff, J. L. Wolf, and P. S. Yu. Replication algorithms in a remote caching architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4:1185–1204, 1993.

[23] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-line distributed data management. In J. van Leeuwen, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms,* Lecture Notes in Computer Science, volume 855, pages 202–214. Springer-Verlag, 1994.

[24] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 284–293, October 1997.

[25] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[26] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic replication on the Internet. Technical Report HA6177000–980305–01-TM, AT&T Labs – Research, April 1998.

[27] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, May 1997.

[28] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Beyond hierarchies: Design considerations for distributed caching on the Internet. Technical Report TR–98–04, Department of Computer Science, University of Texas at Austin, January 1998.

[29] M. van Steen, F. J. Hauck, and A. S. Tanenbaum. A model for worldwide tracking of distributed objects. In *Proceedings of the 1996 Conference on Telecommunications Information Networking Architecture (TINA 96)*, pages 203–212, September 1996.

[30] D. Wessels. Squid Internet object cache. Available at URL http://squid.nlanr.net/Squid, January 1998.

[31] D. Wessels and K. Claffy. Internet Cache Protocol (ICP), version 2, request for comments rfc–2187. Available at URL http://ds.internic.net/rfc/rfc2186.txt, September 1997.

[32] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22:255–314, 1997.

[33] N. E. Young. On-line file caching. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, January 1998.