Competitive On-Line Algorithms for Distributed Data Management*

Carsten Lund†, Nick Reingold†, Jeffery Westbrook†‡, AND Dicky Yan§

Abstract. Competitive on-line algorithms for data management in a network of processors are studied in this paper. A data object such as a file or a page of virtual memory is to be read and updated by various processors in the network. The goal is to minimize the communication costs incurred in serving a sequence of such requests. Distributed data management on important classes of networks—trees and bus based networks, are studied. Optimal algorithms with constant competitive ratios and matching lower bounds are obtained. Our algorithms use different interesting techniques such as work functions [9] and "factoring."

Key words. on-line algorithms, competitive analysis, memory management, data management.

AMS subject classifications. 68Q20, 68Q25.

1. Introduction. The management of data in a distributed network is an important and much studied problem in management science, engineering, computer systems and theory [3, 11]. Dowdy and Foster [11] give a comprehensive survey of research in this area, listing eighteen different models and many papers. A data object, F, such as a file, or a page of virtual memory, is to be read and updated by a network of processors. Each processor may store a copy of F in its local memory, so as to reduce the time required to read the data object. All copies must be kept consistent, however; so having multiple copies increases the time required to write to the object. As read and write requests occur at the processors, an on-line algorithm has to decide whether to replicate, move, or discard copies of F after serving each request, while trying to minimize the total cost incurred in processing the requests. The on-line algorithm has no knowledge of future requests, and no assumptions are made about the pattern of requests. We apply competitive analysis [6] to such an algorithm.

Let σ denote a sequence of read and write requests. A deterministic on-line algorithm A is said to be *c-competitive*, if, for all σ, $C_A(\sigma) \leq c \cdot OPT(\sigma) + B$ holds, where $C_A(\sigma)$ and $OPT(\sigma)$ are the costs incurred by A and the optimal off-line solution respectively, and c and B are functions which are independent of σ, but which may depend upon the input network and file size. If A is a randomized algorithm, we replace $C_A(\sigma)$ by its *expected* cost and consider two types of adversaries: the *oblivious* adversary chooses σ in advance, and the more powerful *adaptive* on-line adversary builds σ on-line, choosing each request with knowledge of the random moves made by A on the previous requests. The oblivious adversary is charged the optimal off-line cost, while the adaptive on-line adversary has to serve σ and be charged on-line. (See Ben-David *et al.* [6] for a full discussion of different types of adversaries.) An algorithm is *strongly* competitive if it achieves the best possible competitive ratio.

In this paper, we focus on two important classes of networks: trees and the uniform network. A tree is a connected acyclic graph on n nodes and $(n-1)$ edges;

* A preliminary version of this paper has appeared in [19].

† Research, AT&T Labs, 600-700 Mountain Avenue, Murray Hill, NJ 07974-0636, U.S.A. e-mail: lund@research.att.com, reingold@research.att.edu, and westbrook@research.att.com

‡ This work was performed while the author was at Yale University. Research partially supported by NSF Grant CCR-9009753.

§ Department of Operations Research, AT&T Labs, Room 3J-314, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030, U.S.A. This work was performed while the author was at Yale University. Research partially supported by Fellowships from Yale University. e-mail:yan@att.com

the uniform network is a complete graph on n nodes with unit edge weights. We obtain strongly competitive deterministic and randomized on-line algorithms for these classes.

Our algorithms use different interesting techniques such as offset functions and "factoring." Competitive on-line algorithms based on offset functions have been found for the 3-server [9] and the migration problems [10]. An advantage of these algorithms is they do not need to record the entire history of requests and the on-line algorithm, since decisions are based on the current offset values which can be updated easily. Factoring is first observed in [7] and used in [10, 17]. The idea is to break down an on-line problem on a tree into single edge problems. Thus strongly competitive strategies for a single edge is generalized to a tree. Our algorithms are strongly competitive for specific applications and networks, and also illustrate these two useful techniques. Our randomized algorithm for file allocation is *barely random* [20], *i.e.*, it uses a bounded number of random bits, independent of the number of requests. A random choice is made only at the initialization of the algorithm, after which it runs deterministically.

1.1. Problem Description. We study three variants of distributed data management: *replication* [1, 7, 17], *migration* [7, 10, 22] and *file allocation* (**FAP**) [2, 5]. They can be can be described under the same framework. We are given an undirected graph $G = (V, E)$ with non-negative edge weights and $|V| = n$, where each node represents a processor. Let F represent a data file or a page of memory to be stored in the processors. At any time, let $R \subseteq V$, the residence set, represent the set of nodes that contain a copy of F. We always require $R \neq \emptyset$. Initially, only a single node v contains a copy of F and $R = \{v\}$.

A sequence of read and write requests occur at the processors. A *read at processor* p requests an examination of the contents of some data location in F ; a *write at processor* p requests a change to the contents of some location in F. The location identifies a single word or record in F. A read can be satisfied by sending a message to any processor holding a copy of F ; that processor then returns the information stored in the requested location. A write is satisfied by sending an update message to each processor holding a copy F, telling it how to modify the desired location. After a request is served, the on-line server can decide how to reallocate the multiple copies of F.

Let $D \in \mathcal{Z}^+$ be an integer constant, $D \geq 1$, which represents the number of records in F.[1] The costs for serving the requests and redistributing the files are as follows.

> *Service Cost:* Suppose a request occurs at a node v. If it is a read request, it is served at a cost equal to the shortest path distance from v to a nearest node in R; if it is a write request, it is served at a cost equal to the size of the minimum Steiner tree[2] that contains all the nodes in $R \cup \{v\}$.
> *Movement Cost:* The algorithm can replicate a copy of F to a node v at a cost D times the shortest path distance between v and the nearest node with a copy of F ; it can discard a copy of F at no cost.

A file reallocation consists of a sequence of zero or more replications and discards of

[1] $\mathcal{R}$, $\mathcal{Z}^+$, and $\mathcal{Z}_0^+$ represent the sets of reals, positive integers and non-negative integers, respectively.

[2] See section 2 for a definition.

copies of F . The replications and allocations can be done in any order as long as the residence set has size at least 1. The movement cost incurred during a reallocation is equal to the total sum of all replication costs.

The replication and migration problems are special cases of file allocation. For migration, we require $|R| = 1$. For replication, all the requests are reads, and it can be assumed that all replicated copies of F are not discarded. The (off-line) optimization problem is to specify R after each new request is served so that the total cost incurred is minimized. For on-line replication, we only consider competitive algorithms that have $B = 0$ in the inequality above; otherwise a trivial 0-competitive algorithm exists [7].

Following previous papers on allocation and related problems, we adopt a "lookahead-0" model. In this model, once a request is revealed, the on-line algorithm must immediately pay the service cost before making any changes to the residence set. One may contrast lookahead-0 with a lookahead-1 model, in which the algorithm may change the residence set before paying the service cost. We discuss the lookahead issue further below, together with some implementation issues.

1.2. Previous and Related Results. Black and Sleator [7] were the first to use competitive analysis to study any of these problems, giving strongly 3-competitive deterministic algorithms for file migration on trees and uniform networks, and strongly 2-competitive deterministic algorithms for replication on trees and uniform networks.
Replication: Imase and Waxman [14] showed that a greedy algorithm for building Steiner trees on-line is $\Theta(\log n)$-competitive, where n is the number of nodes, and that this ratio is optimal within constant factors for general networks. This algorithm is the basis of a solution for on-line replication in general networks. Koga [17] gave randomized algorithms that are 2-competitive and 4-competitive against an adaptive on-line adversary on trees and circles, respectively. He also obtained a randomized algorithm with a competitive ratio that depends only on D and approaches $(1+1/\sqrt{2})$ as D grows large, against an oblivious adversary on trees.
Migration: Westbrook [22] obtained a randomized algorithm for uniform networks with a competitive ratio that depends only on D and approaches $((5+\sqrt{17})/4)$ as D grows large, against an oblivious adversary, For general networks, Westbrook [22] obtained a strongly 3-competitive randomized algorithm against an adaptive on-line adversary. He also obtained an algorithm against an oblivious adversary with a competitive ratio that depends only on D and approaches $(1+\phi)$-competitive as D grows large, where $\phi \approx 1.62$ is the golden ratio. Chrobak *et al.* [10] studied migration on various classes of metric spaces including trees, hypercubes, meshes, real vector spaces, and general products of trees. They gave strongly $(2+1/2D)$-competitive randomized algorithms for these spaces, $(2+1/2D)$-competitive deterministic algorithms for some of these spaces, and a general lower bound for deterministic algorithms of $(85/27)$. Recently, Bartal *et al.* [4] obtained a 4.086-competitive deterministic algorithm.
File Allocation: For general networks, Awerbuch *et al.* [2] and Bartal *et al.* [5] give $O(\log n)$-competitive deterministic and randomized algorithms against an adaptive on-line adversary, respectively. Westbrook and Yan [23] show that Bartal *et al.*'s algorithm is $O(\log d(G))$-competitive on an unweighted graph with diameter $d(G)$, and there exists a $O(\log^2 d(G))$-competitive deterministic algorithm. Bartal *et al.* also find a $(3+O(1/D))$-competitive deterministic algorithm on a tree, and strongly 3-competitive randomized algorithms against an adaptive on-line adversary on a tree and uniform network. Since replication is a special case of file allocation, these upper bounds are also valid for replication when the additive constant B is zero.

		Replication	Migration	File Allocation
Deterministic	uniform	2 [7]	3 [7]	3 [5]
	tree	2 [7]	3 [7]	3*
Randomized	uniform	$e_D/(e_D-1)$*	$2+1/(2D)$*	?
	tree	$e_D/(e_D-1)$*	$2+1/(2D)$ [10]	$2+1/D$*
* this paper				

TABLE 1
The State of the Art: Trees and Uniform Networks. Note $e_D = (1+1/D)^D$.

1.3. New Results. This paper contributes the following results.

- For on-line file allocation on a tree, we give a strongly 3-competitive deterministic algorithm and a $(2+1/D)$-competitive randomized algorithm against an oblivious adversary, and show that this is optimal even if G is an edge.
- For uniform networks, we show that the off-line file allocation problem can be solved in polynomial time. We give a strongly $(2+1/(2D))$-competitive randomized on-line algorithm for migration against an oblivious adversary on the uniform network.
- For the replication problem, we show that the off-line problem is NP-hard; this implies the file allocation problem is also NP-hard. We obtain randomized algorithms that are $(e_D/(e_D-1))$-competitive against an oblivious adversary on a tree and a uniform network; this is optimal even if G is a an edge. (Albers and Koga [1] have independently obtained the same results for on-line replication using a different method.)
- We show that no randomized algorithm for replication on a single edge can be better than 2-competitive against an adaptive on-line adversary. Thus Koga's [17] algorithm for replication on a tree is strongly competitive.

Table 1 summarizes the competitive ratios of the best known deterministic and randomized algorithms against an oblivious adversary for replication, migration, and file allocation on trees and uniform networks. They are all optimal.

1.4. Lookahead and Implementation Issues. As stated above, we adopt the lookahead-0 model that has been used in all previous work on allocation and its variants. Studies of some other on-line problems, however, have used a lookahead-1 model, and in this subsection we comment briefly on the distinction.

In a lookahead-1 model of allocation, some request sequences could be served by an on-line algorithm at a lower cost than would be possible in the lookahead-0 model. For example, if a write request occurs, a lookahead-1 algorithm can drop all but one copies of F before servicing the request, thereby reducing the service cost. The lookahead-0 model is more appropriate for file allocation, however, because the service cost models both the message cost of satisfying a request, which includes the cost of transmitting an answer back to a read request or passing an update on to all copies, and the message cost of the control messages that must be transmitted in order for the algorithm to learn of new requests and to implement its replication and drop decisions. Specifically, we assume that a new replication will not occur unless at least one member of the replication set has been told of a new request, and a processor will not discard a copy unless it has been told of a new write request.

We claim that for large values of D the optimal competitive ratio in a lookahead-1 model is not materially different than the optimal competitive ratio in a lookahead-0 model. In particular, if there is a c-competitive algorithm using lookahead-1, there is a $(c+2/D)$-competitive algorithm using lookahead-0. The lookahead-0 algorithm

simulates the lookahead-1 algorithm by keeping the same residence set. When the lookahead-1 algorithm saves service cost on a read, the amount saved can be no more than the distance it replicates files just prior to the satisfying the request. Similarly, when the lookahead-1 algorithm saves service cost on a write, the amount saved can be no more than the weight of a minimum Steiner tree which connects the dropped copies to an undropped copy. But at some point in the past, at least one of the dropped copies must have been replicated over each edge in that Steiner tree. Hence for each unit of distance saved on reads by the lookahead-1 algorithm, one file was moved one unit of distance. The same holds for writes. The the total cost saved by the lookahead-1 algorithm is $\frac{2}{D}$ times the total movement cost. Both algorithms incur the same movement cost, however.

One may ask whether our service cost is too optimistic: could our algorithms actually be implemented using only the control messages accounted for in the service cost. Although we do not directly address this issue, our algorithms are essentially distributed in nature and can be implemented with only constant message overhead in the special case of uniform and tree networks.

2. Preliminaries. We use the technique of work functions and offset functions introduced by Chrobak and Larmore [9]. Let S be a set of states, one for each legal residence set. Thus S is isomorphic to $2^V \setminus \{\emptyset\}$. Let $R(s)$ denote the residence set corresponding to state $s \in S$. We say the *file system is in state* s if the current residence set is $R(s), s \in S$. Let $Y = \{v^r, v^w | v \in V\}$ be the set of possible requests, where v^r and v^w represent read and write requests at node v, respectively. A request sequence $\sigma = (\sigma_1, \ldots, \sigma_p)$ is revealed to the on-line algorithm, with each $\sigma_i \in Y$. Suppose the network is in state s when σ_i arrives. The algorithm will be charged a service cost of $ser(s, \sigma_i)$, where $ser(s, \sigma_i) : S \times Y \longrightarrow \mathcal{R}$ is as described in Section 1.1. After serving σ_i, the algorithm can move to a different state t at a cost $tran(s, t)$, where $tran : S \times S \longrightarrow \mathcal{R}$ is the minimum cost of moving between the two residence sets.

The work function $W_i(s)$ is the minimum cost of serving requests 1 to i, terminating in state s. Given σ, a minimum cost solution can be found by a dynamic programming algorithm with the following functional equation:

$$\forall\ s \in S,\ i \in \mathcal{Z}^+,\quad W_i(s) = \min_{t \in S}\{W_{i-1}(t) + ser(t, \sigma_i) + tran(t, s)\}$$

with suitable initializations. Let $opt_i = \min_{s \in S} W_i(s), i \geq 1$, be the optimal cost of serving the first i requests. We call $\omega_i(s) = W_i(s) - opt_i$ the *offset function* value at state s after request i has been revealed. Define $\Delta opt_i = opt_i - opt_{i-1}$; it is the increase in the optimal off-line cost due to σ_i.

Our on-line algorithms make decisions based on the current offset values, $\omega_i(s), s \in S$. Note that to compute the $\omega_i(s)$'s and Δopt_i's, it suffices to know only the $\omega_{i-1}(s)$'s. Since $OPT(\sigma) = \sum_{i=1}^{|\sigma|} \Delta opt_i$, to show that an algorithm A is c-competitive, we need only show that for each reachable combination of offset function, request, and file system state, the inequality $\Delta C_A + \Delta\Phi \leq c \cdot \Delta opt_i$ holds, where ΔC_A is the cost incurred by A and $\Delta\Phi$ is the change in some defined potential function. If the total change in Φ is always bounded or non-negative, summing up the above inequality over σ, we have $C_A(\sigma) \leq c \cdot OPT(\sigma) + B$ where B is some bounded value.

The Steiner Tree Problem:

We shall refer to a network design problem called the Steiner tree problem (STP) [24] which can be stated as follows. An instance of STP is given by a weighted undirected

graph $G = (V, E)$, a weight function on the edges $w : E \longrightarrow \mathcal{Z}_0^+$, a subset $Z \subseteq V$ of *regular nodes* or *terminals*, and a constant $B' \in \mathcal{Z}^+$. The decision problem is to ask if there exists a Steiner tree in G that includes all nodes in Z and has a total edge weight no more than B'. STP is NP-complete even when G is restricted to bipartite graphs with unit edge weights, or to planar graphs [12, 16]. Surveys on STP can be found in [13, 24]. On a tree network, the union of paths between all pairs of terminals gives the optimal Steiner tree.

3. Deterministic Algorithms for FAP on a Tree. We begin by introducing some concepts that will be used in building both deterministic and randomized algorithms for file allocation on trees.

We say a residence set is *connected* if it induces a connected subgraph in G. On a tree, if the residence set is always connected, each node without a copy of F can easily keep track of R, and hence the nearest copy of F, by using a pointer. In fact, when G is a tree we can limit our attention to algorithms that maintain a connected R at all times..

THEOREM 3.1. *On a tree, there exists an optimal algorithm that always maintains a connected residence set, i.e., given any (on-line or off-line) algorithm A, there exists an algorithm A' that maintains a connected R and $C_{A'}(\sigma) \leq C_A(\sigma)$ for all σ. If A is on-line, so is A'.*

Proof. Let $R(A)$ and $R(A')$ be the residence sets maintained by A and A', respectively. We simulate A on σ and let A' be such that at any time, $R(A')$ is the minimum connected set that satisfies $R(A) \subseteq R(A')$. Given $R(A)$, on a tree, $R(A')$ is defined and unique.

Since $R(A) \subseteq R(A')$, the reading cost incurred by A' cannot be greater than that by A. The same holds true for the writing cost issued at any node v, since $R(A') \cup \{v\}$ spans the unique minimum length Steiner tree for $R(A) \cup \{v\}$. So A' does not incur a greater read or write cost than A.

Algorithm A' does not need to carry out any replication unless A does, and only to nodes that are not already in $R(A')$. To maintain $R(A) \subseteq R(A')$, A' should leave a copy of F along any replication path, this can be done without incurring any extra cost. As $R(A) \subseteq R(A')$, A' never needs to traverse a replication path longer than that by A for the same replication. Hence, A' cannot incur a greater replication cost. Since a reallocation is a sequence of replications and discards of F, A' maintains a connected set at all times and does not incur a great cost than A in the reallocation. □

Henceforth we shall only consider algorithms that maintain a connected residence set R at all times. When we say that an algorithm replicates to node v, we shall mean it leaves a copy of F at all nodes along the shortest path from the residence set to v. In a tree network we can make some additional simplifying assumptions. Suppose an algorithm A decides to move to residence set R' from set R. This reallocation involves a some sequence of replications and drops.

LEMMA 3.2. *All replications can be performed before all drops without increasing the total cost of the reallocation.*

Proof. Dropping a copy can only increase the cost of subsequent replications. □

Henceforth we assume that all algorithms comply with Lemma 3.2.

LEMMA 3.3. *Let $S = R' \setminus R$ be the nodes that gain a copy of F. Then F can be replicated to the nodes of S in any order at total cost $D \cdot |T(R') \setminus T(R)|$, where $T(R)$ is the subtree induced by node set R.*

Proof. A copy of F must be sent across each edge in $T(R') \setminus T(R)$ at least once.

But in any order of replication, a copy cannot be sent across an edge more than once, because then both endpoints contain a copy of F. □

Henceforth we assume that all algorithms comply with Lemma 3.3.

A useful tool in handling on-line optimization on trees is *factoring* [7, 10]. It makes use of the fact that any sequence of requests σ and any tree algorithm can be "factored" into $(n-1)$ individual algorithms, one for each edge. The total cost in the tree algorithm is equal to the sum of the costs in each individual edge game. For edge (a,b) we construct an instance of two-processor file allocation as follows. The removal of edge (a,b) divides T into two subtrees T_a and T_b, containing a and b, respectively. A read or write request from a node in T_a is replaced by the same kind of request from a, and a request from a node in T_b is replaced by the same request from b. Let A be an algorithm with residence set $R(A)$. Algorithm A induces an algorithm on edge (a,b) as follows: if $R(A)$ falls entirely in T_a or T_b then the edge algorithm is in state a or b, respectively; otherwise, the edge algorithm is in state ab. When the edge algorithm changes state, it does so in the minimum cost way (*i.e. at most one replication*). This factoring approach is used in our algorithms for file allocation on a tree. For the rest of this paper, given an edge (a,b), we use T_a and T_b to represent the subtrees described above, s to denote the state the edge is in, and let the offset functions triplet be $\omega_i = (\omega_i(a), \omega_i(b), \omega_i(ab))$, where $\omega_i(s)$ is the offset function value of state s after σ_i has arrived.

LEMMA 3.4. *For algorithm A and request sequence σ, let $A_{(a,b)}$ be the algorithm induced on edge (a,b), and σ_{ab} be the request sequence induced on edge (a,b). Then*

$$C_A(\sigma) = \sum_{(a,b)\in E} C_{A_{(a,b)}}(\sigma_{ab}).$$

Proof. We show that the cost incurred by any event contributes the same amount to both sides of the equation.

For a write request at a node v, $C_A(\sigma)$ increases by the weight of the unique Steiner tree, T', containing nodes in $R(A)\cup\{v\}$. In the induced problem of any edge e on T', the residence set and the request node are on opposite sides of e, and a write cost equal to e's weight is incurred. For other edges, v and the residence set lie on the same side of e, and no cost is incurred in their induced problems. So both sides of the equation increase by the same amount.

For a read request at a node v, the same argument as in the write case can be used, replacing T' by the unique path from v to the nearest node with a copy of F. Both sides of the equation increase by the same amount.

Suppose A moves from a residence set of R to R', and consider the sequence of replications and discards that make up the reallocation process. We show by induction on the length of this sequence that the movement cost to A is exactly equal to the sum of movement costs in the induced edge problems. Suppose that the first action in the sequence is to replicate F to node v. The cost to A is D times the sum of the lengths of the edges on the shortest path from R to v. Since R is connected, the edges on this path are exactly the edges that must replicate in their induced problems. Thus both sides of the equation increase by the same amount. If the first action is a discard, then no costs are incurred by A or any of the induced edge algorithms. □

LEMMA 3.5. *Let $OPT(\sigma_{ab})$ be the cost incurred by an optimal edge algorithm for (a,b) on sequence σ_{ab}. Then $\sum_{(a,b)\in E} OPT(\sigma_{ab}) \le OPT(\sigma)$.*

Proof. The Lemma follows by letting A in Lemma 3.4 be the optimal off-line algorithm for FAP on a tree, and noting $C_{A_{(a,b)}}(\sigma_{ab}) \ge OPT(\sigma_{ab})$ for any A and edge

(a,b). □

It follows from Lemmas 3.4 and 3.5 that if A is an on-line algorithm such that on any σ, and for each edge (a,b), $C_{A_{(a,b)}}(\sigma_{ab}) \leq c \cdot OPT(\sigma_{ab})$ holds, then A is c-competitive.

To construct a deterministic algorithm for the tree, we first construct a suitable optimal algorithm for a single edge We then design the tree algorithm so that it induces this optimal edge algorithm in each edge, thereby guaranteeing competitiveness.

3.1. An Optimal Deterministic Edge Algorithm. Let $G = (a,b)$ be an edge, and $S = \{a, b, ab\}$ the set of states the file system can be in—only node a has a copy, only node b has a copy, and both a and b have a copy, respectively. We can assume G is of unit length, otherwise the offsets and cost functions can be scaled to obtain the same results. We write the offset functions as a triplet $\omega_i = (\omega_i(a), \omega_i(b), \omega_i(ab))$ and similarly for the work functions. Suppose the starting state is a. Then $W_0 = (0, D, D)$. The *ser* and *tran* functions are given in Table 2. By the definition of the offset functions, and since it is free to discard a copy of F, we always have $\omega_i(ab) \geq \omega_i(a), \omega_i(b)$, and at least one of $\omega_i(a)$ and $\omega_i(b)$ is zero. Without loss of generality, we assume a starting offset function vector of $\omega_i = (0, k, l)$, $0 \leq k \leq l \leq D$, after σ_i has arrived. Table 3 gives the changes in offsets for different combinations of requests and offsets in response to the new request σ_{i+1}.

$tran(t,s)$		s		
		a	b	ab
t	a	0	D	D
	b	D	0	D
	ab	0	0	0

$ser(t,\sigma_i)$		σ_i			
		a^r	a^w	b^r	b^w
t	a	0	0	1	1
	b	1	1	0	0
	ab	0	1	0	1

TABLE 2
Transition and Service Costs

Case 1: $k \geq 1$:

σ_{i+1}	$\omega_{i+1}(a)$	$\omega_{i+1}(b)$	$\omega_{i+1}(ab)$	Δopt_{i+1}
a^r	0	$\min(k+1, l)$	l	0
a^w	0	$\min(k+1, D)$	$\min(l+1, D)$	0
b^r	0	$k-1$	$l-1$	1
b^w	0	$k-1$	l	1

Case 2: $k = 0$:

σ_{i+1}	$\omega_{i+1}(a)$	$\omega_{i+1}(b)$	$\omega_{i+1}(ab)$	Δopt_{i+1}
a^r	0	$\min(1, l)$	l	0
a^w	0	1	$\min(l+1, D)$	0
b^r	$\min(1, l)$	0	l	0
b^w	1	0	$\min(l+1, D)$	0

TABLE 3
Changes in Offsets

Let s be the current state of R. Our algorithm specifies the new required residence set, R, after σ_{i+1} has arrived and the offsets have been updated; it assumes state a is a zero-offset state.

Algorithm Edge:

(1) **If** $s \neq ab$ and $\omega_{i+1}(s) = \omega_{i+1}(ab)$, replicate, *i.e.*, set $s = ab$.

(2) **If** $s = ab$ and $\omega_{i+1}(b) = D$, drop at b, *i.e.*, set $s = a$.

THEOREM 3.6. *Algorithm* **DetEdge** *is strongly 3-competitive.*

Proof. We first show that for each request σ_j, $\Delta C_{Edge} + \Delta\Phi \leq 3 \cdot \Delta opt_j$ $(*)$ holds, for some function $\Phi(\cdot)$ defined below. Let a be a zero-offset state and we have $\omega_i = (0, k, l)$. At any time, we define the potential function:

$$\Phi(s,k) = \begin{cases} 2 \cdot D - 2 \cdot k & \text{if } s = a \\ 2 \cdot D - k & \text{if } s = b \\ D - k & \text{if } s = ab \end{cases}$$

Initially, $\Phi = 0$, and we always have $\Phi \geq 0$. When $\omega_i = (0, 0, l)$ and $s \neq ab$, s can be considered to be in state a or b, and $\Phi(a, 0) = \Phi(b, 0) = 2D$. Note that $\omega_i = \omega_{i+1}$ and $\Delta\Phi = \Delta opt_i = 0$ hold in the following cases:
(i) $\omega_i = (0, D, D)$ and $\sigma_{i+1} = a^r$ or a^w,
(ii) $\omega_i = (0, l, l), l \geq 1$ and $\sigma_{i+1} = a^r$, and
(iii) $\omega_i = (0, 0, 0)$ and $\sigma_{i+1} = a^r$ or b^r.
Our algorithm ensures that $\Delta C_{Edge} = 0$ in these cases. Let use show that $(*)$ holds for all possible combinations of state, request, and offset. The offsets and state variable below are the ones *before* the new request σ_{i+1} arrives. We consider the $k \geq 1$ cases; the $k = 0$ cases are similar to that when $k \geq 1$ and $\sigma_{i+1} = a^r$ or a^w.
Case 1: $\sigma_{i+1} = a^r$
We have $\Delta opt_{i+1} = 0$. If $s = a$ or ab, then $L.H.S.(*) \leq 0$ and $(*)$ holds. If $s = b$, by the last execution of the algorithm, we must have $k < l$. Then $\Delta C_{Edge} = -\Delta\Phi = 1$, and $(*)$ holds.
Case 2: $\sigma_{i+1} = a^w$
We have $\Delta opt_{i+1} = 0$. If $s = a$, then $L.H.S.(*) \leq 0$ and $(*)$ holds. If $s = b$ or ab, we must have $k < D$. Then $\Delta C_{Edge} = -\Delta\Phi = 1$, and $(*)$ holds.
Case 3: $\sigma_{i+1} = b^r$ or b^w
We have $\Delta opt_{i+1} = 1$. In this case $\Delta C_{Edge} \leq 1$, $\Delta\Phi \leq 2$, and $L.H.S.(*) \leq 3$ hold.

Inequality $(*)$ also holds when **DetEdge** changes state: when **DetEdge** moves from state ab to state a, $\omega_i = (0, D, D)$ and $\Delta\Phi = \Delta C_{Edge} = 0$; when **DetEdge** moves to state ab $\Delta\Phi = -\Delta C_{Edge} = -D$. Hence, $(*)$ holds for all possible combinations of offsets, requests, and residence set.

We claim that no deterministic algorithm is better than 3-competitive for FAP on an edge. For migration, it is known that no deterministic algorithm can be better than 3-competitive on a single edge [7]. We show that given any on-line algorithm A for FAP there exists another on-line algorithm A' such that (i) $C_{A'}(\sigma) \leq C_A(\sigma)$ for any σ with only write requests, and (ii) A' always keeps only one copy of F, at a node in A's residence set, and (iii) whenever A has only one copy of F, A' has a copy at the same node. Since A' is a legal algorithm for any instance of the migration problem, and the optimal cost to process σ without using replications is no less than the optimal cost with replications, A is c-competitive on write-only sequences only if A' is a c-competitive migration algorithm. This implies the claim.

Algorithm A' is obtained from A as follows. Initially, both A and A' have a copy of F at the same node. The following rules are applied whenever A changes state.
(1) If A replicates, A' does not change state.
(2) If A migrates, A' follows.
(3) If A drops a page, A' follows to the same node.
It follows from the rules above that (ii) and (iii) hold, and A' cannot incur a write cost higher than that of A. Each movement of A' in (1) or (2) corresponds to a distinct migration or earlier replication by A, respectively. So A' cannot incur a higher movement cost than A. The claim follows. □

3.2. An Optimal Deterministic Tree Algorithm. Recall that for each edge $e = (a, b)$ on the tree, request sequence σ induces a sequence σ_{ab} on (a, b). The tree algorithm is based on factoring into individual edge subproblems and simulating **DetEdge** on each subproblem. After $r \in \sigma$ is served, for each edge $e = (a, b)$ the induced request r_{ab} is computed and the offset vector for the induced subproblem is updated. The following algorithm is then executed, updating the residence set, $R(Tree)$. Initially $R(Tree)$ consists of the single node containing F.

Algorithm Tree:

(1) Examine each edge (u, v) in any order, and simulate the first step of Algorithm **DetEdge** in the induced subproblem. If **DetEdge** replicates to one of the nodes, say v, in the induced subproblem, then add v to $R(Tree)$ and replicate to v.

(2) Simulate step 2 of **DetEdge** for all edges. For any node v, if the edge algorithm for an incident edge $e = (u, v)$ requires deleting node v from e's residence set in e's induced problem, mark v.

(3) Drop at all marked nodes.

To show that **DetTree** is 3-competitive, we will show that it chooses a connected residence set and for each edge, it induces the state required by **DetEdge**. This is not immediately obvious, because the requirements of **DetEdge** on one edge might conflict with those on another edge. For example, one edge might want to drop a copy that another edge has just replicated.

We begin by analyzing the structure of the offset functions in the induced edge problems. For the rest of this subsection, the offset values and functions for each edge (a, b) refers to that results from the induced sequence σ_{ab}. The next lemma characterizes the offset distribution between two adjacent edges.

LEMMA 3.7. *The following properties hold:*

(A) *At any time, there exists a root node r, such that $R = \{r\}$ corresponds to a zero offset state in the induced problems of all edges.*

(B) *For any edge (x, y) on the tree, define $S_i(x, y) = \omega_i(xy) - \omega_i(x)$. Then for any adjacent edges (x, y) and (y, z), the inequality $S_i(x, y) \leq S_i(y, z)$ holds, $\forall\ i$.*

Following from the earlier definitions (see the beginning of Section 3.2), the claim **(A)** above states that there is a node r such that for any edge (a, b) where a is nearer to r than b, state a is a zero offset state for the edge. Note that the location of the root node r may not be unique, and its location changes with requests. The lemma implies the following conditions.

COROLLARY 3.8.

(C) *Let (x, y) be an edge in T such that a root r is in T_x. Let $z \neq x$ be a neighbor of y, and edges (x, y) and (y, z) have offsets $(0, k_{xy}, l_{xy})$ and $(0, k_{yz}, l_{yz})$, respectively. Then*

- **(C.1)** *$l_{xy} \leq l_{yz}$;*
- **(C.2)** *$l_{xy} - k_{xy} \geq l_{yz} - k_{yz}$;*
- **(C.3)** *$k_{xy} \leq k_{yz}$, and*
- **(C.4)** *if $k_{yz} = 0$, then $k_{xy} = 0$ and $l_{xy} = l_{yz}$ hold.*

(D) *Let (x, y) and (y, z) be adjacent edges with a root r in the subtree that is rooted at y and formed from removing the two edges from T. Let the offsets in the edges be $(k_{xy}, 0, l_{xy})$ and $(0, k_{yz}, l_{yz})$, respectively. Then $l_{xy} \geq (l_{yz} - k_{yz})$ holds.*

Proof. (of Lemma 3.7) We use induction on the number of requests. Initially, let r be the node holding the single copy of F ; all the edges have offset vectors $(0, D, D)$

and the lemma holds trivially. We assume the lemma holds for $t \in \mathcal{Z}_0^+$ revealed requests and show that it remains valid after σ_{t+1} has arrived at a node w. We first show how to locate a new root. Let P represent the path from r to w. Unless specified otherwise, the offsets referred to below are the ones *before* σ_{t+1} arrives. We choose the new root, r', using the following procedure.

Procedure FindRoot

(1) **If** (i) $w = r$ or (ii) $w \neq r$ and all the edges along P have offsets of the form $(0, k, l)$, $k \geq 1$, **Then** $r' = r$,

(2) **Otherwise**, move along P from r toward w, and cross an edge if it has offset vector of the form $(0, 0, l)$ until we cannot go any further or when w is reached. Pick the node where we stop as r'.

Let us show that r' is a valid root for the new offsets. We picture P as a chain of edges starting from r, going from left to right, ending in w. If the condition in step (1) of the algorithm is satisfied, σ_{t+1} corresponds to a request at the zero offset state for all edges. By Table 3, r remains a valid root node. Suppose (2) above is executed. For any edge that is not on P, or is on P but is to the right of r', its zero-offset state remains the same. Node r' is a valid root node for these edges. By **(C.4)**, edges along P with offsets of the form $(0, 0, l)$ must form a connected subpath of P, starting from r and ending in r'. They have the same value for the parameter l. By Table 3, their offsets change from $(0, 0, l)$ to $(1, 0, \min\{l + 1, D\})$ or $(\min(1, l), 0, l)$, and r' is a valid root node for them. Hence **(A)** holds for our choice of r' above.

To show that **(B)** holds, we consider any two adjacent edges (x, y) and (y, z) whose removal will divide T into three disjoint subtrees: T_x, T_y, and T_z, with roots x, y, and z, respectively. We show that for different possible positions of r and w, **(B)** remains valid after σ_{t+1} has arrived, *i.e.*, $S_{t+1}(x, y) \leq S_{t+1}(y, z)$ holds when σ_{t+1} is a *write* or a *read*, when $r \in T_x, T_y$, or T_z, and when $w \in T_x, T_y$, or T_z. We assume **(B)** holds before σ_{t+1} arrives.

Suppose σ_{t+1} is a read request, $r \in T_x$, and $w \in T_x$. For edge (x, y), $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy} + 1, l_{xy}), l_{xy})$. For edge (y, z), $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz} + 1, l_{yz}), l_{yz})$. Inequality $S_{t+1}(x, y) \leq S_{t+1}(y, z)$ follows from $S_t(x, y) \leq S_t(y, z)$ or **(C.1)**. Condition **(B)** can be shown to hold in other situations by a similar case analysis. Please refer to the Appendix for the complete case analysis. Thus **(B)** holds for request $(t + 1)$ and the lemma follows. □

THEOREM 3.9. *Algorithm* **DetTree** *is strongly 3-competitive.*

Proof. We show that **DetTree** induces **DetEdge** on each tree edge. The theorem then follows from Lemmas 3.4 and 3.5 and Theorem 3.6.

We proceed by induction on the number of requests. Initially, $R(Tree)$ consists a single node. Suppose $R(Tree)$ is connected after the first $t \in \mathcal{Z}_0^+$ requests, and for each edge (a, b), the state induced by $R(Tree)$ is equal to the state desired by **DetEdge** when run on σ_{ab}. Consider the processing of request $t + 1$.

Step (1): Replication. We do a subinduction on the number of replications done in Step (1), and show that no replication is in conflict with the state desired by any edge.

Suppose that processing edge (a, b) in Step (1) causes F to be replicated to a. Then $r \in T_a$, $R(Tree)$ lies in T_b, inducing state $s = b$, and $\omega_{t+1} = (0, l, l)$. This follows from the definition of **DetEdge**, the definition of the induced subproblem, and the inductive hypothesis. Let Q be the path from b to the nearest node in $R(Tree)$. If $Q \neq \{b\}$, then, to avoid conflict, each edge along Q must also require replication across it. From **(A)** and **(C.2)** in Theorem 3.7, we see that each edge (x, y) in Q has

an offset of the form $\omega_{t+1} = (0, l', l')$ where x is nearer to b than y, and requires a replication. A similar argument holds for the case when $s = a$ and $\omega_{t+1} = (0, 0, 0)$.

Step (2): Marking Nodes to Drop: Again we perform a subinduction on the number of markings done in Step (2), and show that no marking is in conflict with the state desired by any edge and that a connected residence set results.

Suppose that processing edge (a, b) in Step (2) causes b to be marked. This occurs because (a, b) has $\omega_{t+1} = (0, D, D)$, $r \in T_a$, r a write, and $s = ab$.

Since $R(Tree)$ is connected by hypothesis, both a and b are in $R(Tree)$, and the nodes in T_b with a copy of F span a connected subtree of T_b, with b as its root. Let us call it T'_b. If $T'_b \neq \{b\}$, each edge (x, y) in T'_b is in state $s = xy$. By **(A)** and **(C.3)** in Theorem 3.7, (x, y) must have offset $\omega_{t+1} = (0, D, D)$, with x nearer to b than y is. Under **DetEdge**, (x, y) needs to drop the copy of F in node y. Hence all the nodes in T'_b are required to be removed from $R(Tree)$, the new $R(Tree)$ remains connected, and no edges are in conflict.

Thus $R(Tree)$ is connected, all induced edge algorithms match **DetEdge**, and **DetTree** is 3-competitive. □

4. Randomized Algorithms for FAP on a Tree. Our approach to building a randomized tree algorithm is the same as our approach in the deterministic case. We give a randomized algorithm for a two-point space, **RandEdge**, that is based on counter values assigned at the nodes. By factoring, we obtain from **RandEdge** a $(2+1/D)$-competitive algorithm, **RandTree**, for file allocation on a tree. **RandTree** requires the generation of only $O(\log D)$ random bits at the beginning of the algorithm, after which it runs completely deterministically. It is simpler than the tree algorithm in [19], which can require the generation of $\Omega(\log D)$ random bits after each request is served.

4.1. An Optimal Randomized Edge Algorithm, RandEdge. Let edge $e = (a, b)$. We maintain counters c_a and c_b on nodes a and b, respectively. They satisfy $0 \leq c_a, c_b \leq D$ and $(c_a + c_b) \geq D$. Our algorithm maintains a distribution of R dependent on the counter values. Initially, the node with a copy of F has counter value D, and the other node has counter value 0. The counter values change according to the following rules. On a read request at a, we increment c_a if $c_a < D$. On a write request at a, if $(c_a + c_b) > D$, we decrement c_b; if $(c_a + c_b) = D$ and $c_a < D$, we increment c_a. The counters change similarly for a request at b. There is no change in the counter values in other cases.

Algorithm **RandEdge** always maintains a distribution of R such that

$$p_e[a] = 1 - \frac{c_b}{D}, \tag{1a}$$

$$p_e[b] = 1 - \frac{c_a}{D}, \text{ and} \tag{1b}$$

$$p_e[ab] = \frac{c_a + c_b}{D} - 1 \tag{1c}$$

Observe that the probability of having a copy of F at node $v \in \{a, b\}$ is c_v/D.

In order to maintain this distribution, **RandEdge** simulates D deterministic algorithms, numbered from 1 to D. The moves of each deterministic algorithm are constructed (deterministically) on-line, according to rules given below. Before the first request, one of the D algorithms is picked at random. **RandEdge** then makes

the same moves as the chosen deterministic algorithm. Thus $p_e[s]$, $s \in \{a, b, ab\}$, is the proportion of algorithms in state s, and the expected cost incurred by **RandEdge** is the average of the costs incurred by the D algorithms.

We define the D algorithms that achieve the probability distribution in (1). Suppose that initially, only node a has a copy of F. Then initially the D algorithms are placed in state a. The following changes are made after a new request, σ_i, has arrived. Without loss of generality, we assume the request arises at node a. (The c_a and c_b values below refer to the counter values just *before* σ_i arrives.)

- There is no change in the algorithms if there is no change in the counter values.
- Case 1: if $\sigma_i = a^r$ and $c_a < D$, the lowest-numbered algorithm in state b moves to state ab.
- Case 2: if $\sigma_i = a^w$, $(c_a + c_b) > D$, the lowest-numbered algorithm in state ab moves to state a.
- Case 3: if $\sigma_i = a^w$, $(c_a + c_b) = D$, and $c_a < D$, the lowest-numbered algorithm in state b moves to state ab.

LEMMA 4.1. **RandEdge** *is feasible and maintains the probability distribution in (1).*

Proof. By feasible we mean that whenever a move must be made in Cases 1, 2, and 3, there is some algorithm available to make the move. The choice of lowest-numbered algorithm is only to emphasize that the choice must be independent of which algorithm **RandEdge** is actually emulating.

The lemma holds initially with $c_a = D$ and $c_b = 0$. We prove the lemma by induction on the requests, and assume it holds before σ_i arrives. If there is no change in counter values after σ_i has arrived, the lemma holds trivially. By the induction hypothesis, in case 1 above, since $c_a < D$ and $p_e[b] > 0$, at least one of the D algorithms is in state b; in case 2, since $(c_a + c_b) > D$ and $p_e[ab] > 0$, there is an algorithm in state ab; in case 3, since $c_a < D$, there is an algorithm in state b. Hence, **RandEdge** is feasible. It can be verified that the changes in the algorithms implement the probability distribution in (1) for the new counter values. □

THEOREM 4.2. **RandEdge** *is strongly* $(2 + 1/D)$*-competitive.*

Proof. For each node $v \in \{a, b\}$, we maintain the potential function:

$$\phi_v = \begin{cases} \frac{D+1}{2} + \sum_{j=c_v}^{D-1}(2 - \frac{j}{D}) & \mathcal{OPT} \text{ has a copy of } F \text{ at } v \\ \sum_{j=1}^{c_v} \frac{j}{D} & \text{otherwise} \end{cases}$$

where $\mathcal{OPT}$ represents the adversary. Let the overall potential function $\Phi = \phi_a + \phi_b - (D+1)/2$. Initially, $\Phi = 0$; at any time, $\Phi \geq 0$. We show that in response to each request and change of state,

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) + \mathbf{E}(\Delta M_i) + \Delta\Phi \leq (2 + 1/D) \cdot \Delta OPT. \tag{2}$$

holds, where ΔOPT, $\mathbf{E}[\Delta C_{\mathbf{RandEdge}}]$ and $\mathbf{E}(\Delta M_i)$ are the cost incurred by the event on $\mathcal{OPT}$, and the service and movement costs incurred on **RandEdge**, respectively. The c_a and c_b values below are the counter values just before the new request σ_i arrives.

Case 1: Request $\sigma_i = a^r$.

If $c_a = D$, inequality (2) holds trivially. Suppose $c_a < D$. We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = 1 - \frac{c_a}{D}, \qquad \mathbf{E}(\Delta M_i) = 1,$$

$$\Delta\Phi = \begin{cases} -2 + \frac{c_a}{D} & \mathcal{OPT} \text{ has a copy of } F \text{ at } a \\ \frac{c_a+1}{D} & \text{otherwise} \end{cases}$$

It follows that if $\mathcal{OPT}$ has a copy of F at a when σ_i arrives, $L.H.S.(2) = \Delta OPT = 0$; otherwise, $L.H.S.(2) = (2 + 1/D) = (2 + 1/D) \cdot \Delta OPT$. Inequality (2) holds.
Case 2: Request $\sigma_i = a^w$ and $(c_a + c_b) > D$.
We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = \frac{c_b}{D}, \quad \mathbf{E}(\Delta M_i) = 0,$$

$$\Delta\Phi = \begin{cases} 2 - \frac{c_b-1}{D} & \mathcal{OPT} \text{ has a copy of } F \text{ at } b \\ \frac{-c_b}{D} & \text{otherwise} \end{cases}, \text{ and}$$

$$L.H.S.(2) = \begin{cases} 2 + \frac{1}{D} & \mathcal{OPT} \text{ has a copy of } F \text{ at } b \\ 0 & \text{otherwise} \end{cases}$$

Inequality (2) holds.
Case 3: Request $\sigma_i = a^w$ and $(c_a + c_b) = D$.
If $c_a = D$, L.H.S.(2)=0 and (2) holds trivially. Suppose $c_a < D$. We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = 1 - \frac{c_a}{D}, \quad \mathbf{E}(\Delta M_i) = 1,$$

$$\Delta\Phi = \begin{cases} -2 + \frac{c_a}{D} & \mathcal{OPT} \text{ has a copy of } F \text{ at } a \\ \frac{c_a+1}{D} & \text{otherwise} \end{cases}, \text{ and}$$

$$L.H.S.(2) = \begin{cases} 0 & \mathcal{OPT} \text{ has a copy of } F \text{ at } a \\ 2 + \frac{1}{D} & \text{otherwise} \end{cases}$$

Hence, (2) holds.
Case 4: $\mathcal{OPT}$ changes state.
When $\mathcal{OPT}$ changes state, $\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = \mathbf{E}(\Delta M_i) = 0$. It can be checked from the definition of Φ that when $\mathcal{OPT}$ replicates, $\Delta OPT = D$ and $\Delta\Phi \leq (2D + 1)$ hold; when $\mathcal{OPT}$ discards a copy of F, $\Delta\Phi \leq 0$.

Since (2) holds for all possible events, by Theorem 4.6 **RandEdge** is strongly $(2 + 1/D)$-competitive. □

4.2. An Optimal Randomized Tree Algorithm—RandTree. We extend **RandEdge** to a randomized algorithm for FAP on a tree, T, by means of factoring. Our algorithm, **RandTree**, induces **RandEdge** on each edge for the induced request sequence for the edge.

Description of Algorithm RandTree. **RandTree** internally simulates D deterministic algorithms. Each of them maintains a residence set that spans a subtree of T. Initially, the residence set for each of them is the single node that contains F. One of the D simulated algorithms is picked uniformly at random at the beginning, and **RandTree** behaves exactly the same as the particular algorithm chosen.

We maintain counters c_a and c_b for each edge (a, b) in the tree. Using the factoring approach (see section 3.2), we obtain an induced request sequence σ_{ab} for (a, b). The counter values change according to the same rules as described in the single edge case (section 4.1), using σ_{ab}. **RandTree** responds to each request and maintains an (induced) distribution as required by **RandEdge** in (1) for each of the edges.

Read Request. Suppose the new request, σ_i, is a *read* request at a node g. Let T be rooted at g, and $e = (a, b)$ be an edge with a nearer to g than b is. The c_a and c_b values described below are the counter values before σ_i arrives. The edges can be classified into three types:

- type 1: edges with $c_a = D$ and $c_b = 0$;
- type 2: edges with $c_a = D$ and $c_b > 0$, and
- type 3: edges with $c_a < D$.

RandEdge requires no change in probability values for the first two types of edges; for type 3 edges, it requires $p_e[b]$ decreases by $1/D$ and $p_e[ab]$ goes up by $1/D$. For any node v, we use $T(v)$ to denote the subtree of T rooted at v. **RandTree** changes the subtree configurations maintained by the D algorithms by using the following procedure (fig. 1).

(1) Let $\mathcal{F}$ be the forest of trees formed by all the type 3 edges.
(2) **While** there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (2.1) Let x be a leaf node in T' and P be the path from x to the root node of T'.
  (The root node of T' is the node in T' that is nearest to g.)
  (2.2) Pick any one of the D algorithms that maintains a subtree, Z, that includes node x and lies entirely in $T(x)$.
  Make that algorithm replicate along P, *i.e.*, replace Z by $Z \cup P$.
  (2.3) Remove the edges in P from T' and update the forest $\mathcal{F}$.

FIG. 1. *Algorithm RandEdge (Read Requests)*

LEMMA 4.3. **RandTree** *implements the required changes for all the edges for a read request.*

Proof. We prove the lemma by induction on the requests. Suppose that **RandTree** induces **RandEdge** on all edges before σ_i arrives. Let y be the parent node of x. If **RandTree** is feasible, *i.e.*, it can be executed, it implements the changes required by **RandEdge** as described above, for all edges. We show that this is the case.

If x is a leaf node of T, since (x, y) is a type 3 edge, $p_{(x,y)}[x] > 0$ and one of the D algorithms must have the single node $\{x\}$ as its tree configuration.

Otherwise, suppose all the descending edges of x are of type 1. Let (x, w) be one of them. Then $p_{(x,w)}[xw] = p_{(x,w)}[w] = 0$; none of the algorithms maintains a subtree with any edge in $T(x)$. Since $p_{(x,y)}[x] > 0$, one of the algorithms must have $\{x\}$ as its subtree.

Otherwise, suppose x has descending type 2 edges. Let (x, w) be any one of them. Then $p_{(x,w)}[w] = 0$ and $p_{(x,w)}[xw] > 0$. Thus each of these edges is contained in the subtree of at least one of the algorithms, and none of the algorithms has its subtree in $T(w)$. Since $p_{(x,y)}[x] > 0$, at least one of these subtrees must lie in $T(x)$ and contains node x.

Hence, our algorithm is feasible and the lemma holds. □

Write Request. Suppose $\sigma_i = r^w$. We use the same notation as in the read request case. The edges can be classified into three types:

- type 1: edges with $(c_a + c_b) > D$;
- type 2: edges with $(c_a + c_b) = D$ and $c_a < D$, and
- type 3: edges with $(c_a + c_b) = D$ and $c_a = D$.

RandEdge requires no change in probability values for the type 3 edges; for type 1 edges, it requires $p_e[ab]$ decreases by $1/D$ and $p_e[a]$ increases by the same amount; for type 2 edges, it requires $p_e[b]$ decreases by $1/D$ and $p_e[ab]$ increases by the same amount. **RandTree** performs the following (fig. 2).

(1) Let $\mathcal{F}$ be the forest of trees formed by all the type 1 edges.
(2) **While** there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (2.1) Let g' be the root node of T' and x be one of its children nodes in T'.
  (2.2) Pick an algorithm that has a subtree Z that includes edge (g', x).
  (2.3) If Z is contained in T', make the algorithm replace Z by the single-node subtree $\{g'\}$; otherwise, replace Z by the tree formed by edges in $(Z - T')$.
  (2.4) Replace T' in $\mathcal{F}$ by the subtrees formed by $T' - Z$.
(3) Let $\mathcal{F}$ be the forest of trees formed by all the type 2 edges.
(4) **While** there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (4.1) Let x be a leaf node of T' and P be the path from x to the root node of T'.
  (4.2) Pick any one of the D algorithms that maintains a subtree, Z, that includes node x and lies entirely in $T(x)$.
  Make that algorithm replicates along P, *i.e.*, extends Z to $Z \cup P$.
  (4.3) Remove the edges in P from T' and update the forest $\mathcal{F}$.

FIG. 2. *Algorithm RandEdge (Write Requests)*

LEMMA 4.4. **RandTree** *implements the required changes for all the edges for a write request.*

Proof. We prove by induction and assume **RandEdge** is induced on all the edges before σ_i arrives. If **RandTree** is feasible, it implements the required changes for all the edges. We show that this is the case.

Consider the first loop of the algorithm (in step (2)). Since $p_{(g',x)}[g'x] > 0$, subtree Z must exist. **RandTree** removes edges from Z that are contained in T'. Note that the $p_e[ab]$ values for type 2 and 3 edges are zero; **RandTree** processes edges in T' in a top-down fashion, and configuration $(Z - T')$ is always a connected subtree. Thus the first loop can be executed.

Consider the second loop of the algorithm (in step (4)). Let y be a parent node of x. Then $p_{(x,y)}[xy] = 0$ and $p_{(x,y)}[x] > 0$ hold. If x is a leaf node in T, one of the D algorithms must have $\{x\}$ as its subtree. If x has a descending type 1 edge, by the first part of the algorithm, one of the D algorithms must have $\{x\}$ as its subtree after running the first loop of the algorithm. Suppose all the descending edges of x are type 3 edges. Let (x, w) be a type 3 edge; then $p_{(x,w)}[w] = p_{(x,w)}[xw] = 0$ and $p_{(x,w)}[x] = 1$. Since $p_{(x,y)}[x] > 0$, one of the algorithms must have $\{x\}$ as its subtree.

Hence, the algorithm is feasible and the lemma follows. □

Lemmas 4.3 and 4.4 imply that **RandTree** induces **RandEdge** on all the edges. Theorem 4.5 follows from the above lemmas and Theorem 4.2.

THEOREM 4.5. *Algorithm* **RandTree** *is strongly* $(2 + 1/D)$*-competitive for FAP on a tree against an oblivious adversary.*

4.3. Lower Bound. We show that the competitive ratio, $(2 + 1/D)$, obtained above is the best possible for file allocation against an oblivious adversary, even if G is a single edge.

THEOREM 4.6. *No on-line algorithm for the file allocation problem on two points* (a, b) *is* c*-competitive, for any* $c < (2 + 1/D)$.

Proof. Let A be any randomized algorithm for the file allocation problem on two points. We define a potential function Ψ, and give a strategy for generating adversary request sequences such that:
(i) for any C there is a request sequence σ with optimum cost $\geq C$;
(ii) the cost to **RandEdge** on σ is at least $(2 + 1/D)OPT(\sigma) - B''$, for B'' bounded independent of σ;
(iii) Ψ is bounded; and
(iv) for each request generated by this adversary,

$$\Delta C_A + \Delta\Psi \;\geq\; \Delta C_{\mathbf{RandEdge}} \tag{3}$$

If conditions (ii), (iii), and (iv) hold for an adversary sequence σ, then summing (3) over the sequence gives

$$C_A(\sigma) \geq (2 + 1/D) \cdot OPT(\sigma) - B$$

where B is bounded. By condition (i), the adversary can make $OPT(\sigma)$ arbitrarily large, so there is no constant B' independent of σ such that $C_A(s) < (2 + 1/D) \cdot OPT(\sigma) + B'$. Hence A cannot be c-competitive for $c < 2 + 1/D$.

We now define the adversary's strategy. We assume that both the on-line and off-line algorithms start with a single copy of F at a. Our adversary will only generate requests that result in offset functions of the form $(0, i, i)$, where $0 \leq i \leq D$. A *zero-cost self-loop* is a request such that the offset function is unchanged and $\Delta opt = 0$. By a theorem of [18], there is always an optimal on-line algorithm that incurs 0 expected cost on a zero-cost self loop. We assume A has this property. This simplifies the adversary's strategy, although the result can still be proved without this assumption.

Suppose that the current offset function is $(0, i, i)$, and let p_i be probability that **RandEdge** is in state a. Suppose A is in state a with probability q. If $q < p_i$ the adversary requests a^w, otherwise the adversary requests b^r. When $i = D$ we will have $q = 1$ (a^w is a zero-cost self-loop if $i = D$) and so the adversary will request b^r. Similarly, when $i = 0$, $q = 0$ (a^r is a zero-cost self-loop) and the adversary requests a^w. Therefore the adversary can always generate a next request using the above rules, and the request sequence can be made arbitrarily long. Since there are only D offset functions that can be generated by this strategy, an arbitrarily long sequence of requests must cycle through the offset functions arbitrarily often. Notice, however, that the only cycles which cost $\mathcal{OPT}$ nothing are zero-cost self-loops. Since the adversary never uses these requests, all cycles have non-zero cost, so by continuing long enough the adversary can generate request sequences of arbitrarily large optimum costs. Hence condition (i) hold.

Next we consider condition (ii). Recall c_a and c_b, the counter values maintained by **RandEdge**. We claim that if the offset function is $(0, i, i)$, then $c_a = D$ and $c_b = D - i$. This is true initially, when F is located only at a, and $i = D$. By inspection of **RandEdge** one can verify that the whenever the adversary generates request b^r, c_b increases by 1, and that whenever the adversary generates a^w, c_b decreases by 1. Hence $p_i = i/D$ and the expected movement cost incurred by **RandEdge** is 1 on b^r and 0 on a^w. With reference to the proof of Theorem 4.2, note that the amortized cost to **RandEdge** is exactly $2 + 1/D$ times the cost to $\mathcal{OPT}$ on any request that the adversary might generate, assuming $\mathcal{OPT}$ does not move following the request. (The adversary never generates b^r if $c_b = D$ or a^w if $c_b = 0$.) The amortized cost incurred

by **RandEdge** is therefore exactly $2+1/D$ times the cost incurred by an "optimum" algorithm that only ever has a copy of F at a. It is possible to show that this cost really is optimum for our sequences, but in any case it is certainly lower-bounded by the true optimum cost, and so (ii) holds.

Now define Ψ to be $D \cdot \max\{0, q - p_i\}$. This is trivially bounded by 1, so (iii) holds. Finally, we must verify (3).

Case 1: The adversary requests a^w.
In this case the new offset function must be $(0, i+1, i+1)$. Suppose that after the request A has mass q' at a. Then $\Delta C_A = 1 - q + D \cdot \max\{0, q - q'\}$, $\Delta\Psi = D \cdot \max\{0, q' - p_{i+1}\} - D \cdot \max\{0, q - p_i\}$, and $\Delta C_{\mathbf{RandEdge}} = 1 - p_i$. Since $q < p_i < p_{i+1}$,

$$\begin{aligned} \Delta C_A + \Delta\Psi &= 1 - q + D \cdot \max\{0, q - q'\} + D \cdot \max\{0, q' - p_{i+1}\} - D \cdot \max\{0, q - p_i\} \\ &\geq 1 - q \\ &\geq 1 - p_i \\ &= \Delta C_{\mathbf{RandEdge}} \end{aligned}$$

Case 2: The adversary requests b^r.
In this case the new offset function must be $(0, i-1, i-1)$. Suppose that after the request A has mass q' at a. Then $\Delta C_A = q + D \cdot \max\{0, q - q'\}$, $\Delta\Psi = D \cdot \max\{0, q' - p_{i-1}\} - D \cdot \max\{0, q - p_i\}$, and $\Delta C_{\mathbf{RandEdge}} = p_i + 1$. Since $q \geq p_i > p_{i-1}$,

$$\begin{aligned} \Delta C_A + \Delta\Psi &= q + D \cdot \max\{0, q - q'\} + D \cdot \max\{0, q' - p_{i-1}\} - D \cdot \max\{0, q - p_i\} \\ &\geq q + D(q - p_{i-1}) - D \cdot \max\{0, q - p_i\} \\ &= q + D\,(p_i - p_{i-1}) \\ &= q + 1 \\ &\geq p_i + 1 \\ &= \Delta C_{\mathbf{RandEdge}} \end{aligned}$$

□

5. Migration on a Uniform Network. We give a $(2 + 1/(2D))$-competitive randomized algorithm against an oblivious adversary for migration on a uniform network. This competitive ratio is optimal even for a single edge [10]. Let G be a complete graph on n nodes labeled 1 to n. Initially, only node 1 has a copy of F. Our algorithm is based on the offsets calculated on-line. Let $S = \{1, \ldots, n\}$ and the algorithm is in state s if the single copy of F is at node s. We have the cost functions

$$ser(t, \sigma_i) = \begin{cases} 0 & \text{if } t = \sigma_i \\ 1 & \text{otherwise} \end{cases} \qquad tran(t, s) = \begin{cases} 0 & \text{if } t = s \\ D & \text{otherwise} \end{cases}$$

and initially $W_0 = (0, D, \ldots, D)$.
Suppose the ith request is served and the new offset for each node, s, is calculated. Let $v_s = D - \omega_i(s)$, $k = max\{j \in S | \sum_{m=1}^{j} v_m < 2D\}$, and $\delta = 2D - \sum_{m=1}^{k} v_m$.
Algorithm Migrate: The algorithm maintains a probability distribution such that the probability, $p[s]$, that a node, s, contains F is as follows:

$$\text{If } k < n, \quad p[s] = \begin{cases} v_s/2D & \text{if } s \leq k \\ \delta/(2D) & \text{if } s = k+1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{If } k = n, \quad p[s] = \begin{cases} v_s/2D & \text{if } s \le k \text{ and } v_s < D \\ (v_s + \delta)/(2D) & \text{if } s \le k \text{ and } v_s = D \\ 0 & \text{otherwise} \end{cases}$$

After a new request has arrived, our algorithm moves to different states with transition probabilities that minimize the total expected movement cost, while maintaining the new required distribution.

THEOREM 5.1. *Given any σ, the expected cost incurred by* **Migrate**, $\mathbf{E}[C_{mig}(\sigma)]$, *satisfies*

$$\mathbf{E}[C_{mig}(\sigma)] \le [2 + 1/(2D)] \cdot OPT(\sigma)$$

Proof. We show that after a request has arrived,

$$\mathbf{E}[\Delta C_{mig}] + \mathbf{E}[\Delta M] + \Delta\Phi \le \left(2 + \frac{1}{2D}\right) \cdot \Delta opt_i \tag{4}$$

holds, where $\mathbf{E}[\Delta C_{mig}]$ is the expected cost incurred by **Migrate**, $\mathbf{E}[\Delta M]$ is the expected movement cost, $\Delta\Phi$ is the change in the potential function:

$$\Phi = \sum_{s=1}^{n} \sum_{j=1}^{v_s} \left(\frac{1}{2} + \frac{j}{2D}\right) - (3D+1)/4.$$

Initially, $\Phi = 0$. At any time, since at least one $v_s = D$, we have $\Phi \ge 0$.

An offset table similar to Table 3 for file allocation can be constructed. Since migration is equivalent to FAP with only write requests, it can be seen that if request σ_{i+1} is at a node s with $\omega_i(s) = 0$, we have $\omega_{i+1}(s) = 0$, the offsets for all other states will increase by one, subject to a maximum of D, and $\Delta opt_{i+1} = 0$. If σ_{i+1} is at a node s with $\omega_i(s) > 0$, the offset for state s will decrease by one, all other offsets remain the same, and $\Delta opt_{i+1} = 1$.

Case 1: A request at s such that $v_s < D$.

In this case, we have $\Delta opt_{i+1} = 1$, and v_s increases by one.

If $s \le k$ then $\mathbf{E}[\Delta C_{mig}] \le [1 - v_s/(2D)]$. It can be verified that $\mathbf{E}[\Delta M] \le 1/2$, $\Delta\Phi \le 1/2 + (v_s + 1)/(2D)$, and $L.H.S.(4) \le 2 + 1/(2D) = R.H.S.(4)$.

If $s > k$ then the movement cost is zero. We also have $\mathbf{E}[\Delta C_{mig}] \le 1$, $\Delta\Phi = 1/2 + (v_s + 1)/(2D) \le 1 + 1/(2D)$; inequality (4) also holds.

Case 2: A request at s such that $v_s = D$.

We have $\Delta opt_{i+1} = 0$. For each $j \in S - \{s\}$ such that $v_j > 0$, v_j decreases by 1. Each such v_j contributes $-[1/2 + v_j/(2D)]$ to $\Delta\Phi$, and no more than $1/2$ to $\mathbf{E}[\Delta M]$. We have $\mathbf{E}[\Delta Cost] = (1 - p_s)$, where p_s is the probability mass at s, and

$$1 - p_s = \sum_{j \ne s} p_j \le \sum_{j \ne s, v_j > 0} \frac{v_j}{2D}$$

Hence $L.H.S.(4) \le 0$. ∎

6. Replication. We give upper and lower bounds on the performance of randomized on-line algorithms for the replication problem.

6.1. Randomized On-Line Algorithms. Let $e_D = (1 + 1/D)^D$ and $\beta_D = e_D/(e_D - 1)$. So $\beta_D \longrightarrow e/(e-1) \approx 1.58$ as $D \longrightarrow \infty$. We describe randomized algorithms that are β_D-competitive against an oblivious adversary on the uniform network and trees. First, consider a single edge (r, b) of unit length. Initially, only r contains a copy of F. Suppose algorithm A is α-competitive, and it replicates F to node b with probability p_i after the ith request at b, where $\sum_i p_i = 1$. The p_i's and α must satisfy, for each $k \in \mathcal{Z}_0^+$,

$$\mathbf{E}[C_A(\sigma) \mid k] \leq \begin{cases} \alpha \cdot k & k \leq D \\ \alpha \cdot D & k > D \end{cases},$$

$$\text{where } \mathbf{E}[C_A(\sigma) \mid k] = \sum_{i=1}^{k} p_i \cdot (D + i) + \left(1 - \sum_{i=1}^{k} p_i\right) \cdot k$$

is the expected cost incurred by A when σ contains k requests at b. The optimal off-line strategy is to replicate a copy of F to b before the first request arrives if $k \geq D$, and does not replicate otherwise. Algorithm A incurs a cost of $(D + i)$ if it replicates right after serving the ith request, $i \leq k$; otherwise it incurs a cost of k. An optimal randomized algorithm is given by a set of p_i values that satisfy the above inequalities for all k, such that α is minimized. We note that the conditions above are identical to those for the on-line block snoopy caching problem on two caches in [15]. Karlin *et al.* [15] showed that the optimal α value is β_D. This is achieved when $p_i = [(D+1)/D]^{i-1}/[D(e_D - 1)]$, $1 \leq i \leq D$, and other p_i's are zero. The above single edge algorithm can be applied to a uniform network by replicating F to each node v after the ith request at v, with a probability of p_i — another example of factoring.

THEOREM 6.1. *There exists a randomized algorithm that is strongly β_D-competitive against an oblivious adversary for replication on a uniform network.*

We can extend the single edge algorithm to a tree, T, rooted at r, the node that contains F initially. The algorithm only responds to requests at nodes not in the current residence set.

> **Algorithm TREE:** Keep a counter c_i on each node $i \neq r$. Initially, all $c_i = 0$. Suppose a request arrives at a node x, and w is the node nearest to x that contains a copy of F. After serving the request, the counters for all the nodes along the path from w to x are increased by one. Let the nodes along the path be $w = i_0, i_1, \ldots, i_q = x$, $q \geq 1$. Perform the following procedure each time after a request has been served:
>
> (1) Let $p_{c_{i_0}} = 1$.
>
> (2) **For** $j = 1, \ldots, q$, **Do**
>
> (2.1) With probability $p_{c_{i_j}}/p_{c_{i_{j-1}}}$, replicate to node i_j from node i_{j-1}.
>
> (2.2) **If** F is not replicated to i_j, **STOP**.

THEOREM 6.2. *Algorithm TREE is strongly β_D-competitive against an oblivious adversary for replication on a tree network.*

Proof. Without loss of generality, we assume that a *connected* R is always maintained by any solution. Let $x \neq r$ be any node and y its parent. Before x obtains a copy of F, a cost equal to the weight of (x, y) is incurred on the edge for each request at a node in the subtree rooted at x. (This follows because R is connected and the unique path from x to r passes through (x, y).) By our single edge algorithm, the

algorithm TREE is β_D-competitive if, for each node $x \neq r$, a copy of F is replicated to x after the ith request at the subtree rooted at x, with a probability p_i. It can be shown by induction on the requests that before x acquires a copy of F, the counter on x records the number of requests that have arrived at the subtree rooted at x, and these counters form a non-increasing sequence on any path moving away from r. Hence, the values $p_{c_{i_j}}/p_{c_{i_{j-1}}} = (1+1/D)^{c_{i_j}-c_{i_{j-1}}} \leq 1$, $j = 2, \ldots, q$, are defined probability values. It is simple to verify that each time a node at the subtree rooted at x receives a request, a copy of F is replicated to x with probability p_{c_x}, where c_x is x's new counter value. The theorem follows. □

6.2. Lower Bound. We show that no randomized algorithm can be better than 2-competitive against an adaptive on-line adversary. We use n to denote the number of nodes in G.

THEOREM 6.3. *Let ϵ be any positive function of n and D, taking values between 0 and 1. No on-line algorithm is better than $(2-\epsilon(D,n))$-competitive for replication against an adaptive on-line adversary.*

Proof. Let node a have the initial copy of F and let (a,b) be any edge in G. Let A be any on-line algorithm which replicates to b after the jth request at b with probability p_j, $j \in \mathcal{Z}^+$. The adversary issues requests at b until A replicates or until N_ε requests have been issued, whichever first happens. Algorithm A incurs an expected cost of

$$\mathbf{E}[C_A(\sigma)] = \sum_{j=1}^{N_\varepsilon} p_j \cdot (j+D) + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon \tag{5}$$

We choose different N_ε's for two different cases.

Suppose $\sum_{j=1}^{\infty} p_j \cdot j > D$. Let N_ε be the minimum positive integer that is greater than D and such that $\sum_{j=1}^{N_\varepsilon} p_j \cdot j \geq D$. Given D, parameter N_ε is a finite and unique constant. The adversary replicates to b before the first request arrives, incurring a cost of D. From equation (5), we have $\mathbf{E}[C_A(\sigma)] \geq D + \sum_{j=1}^{N_\varepsilon} p_j \cdot j \geq 2D$, giving a ratio of at least 2.

Suppose $\sum_{j=1}^{\infty} p_j \cdot j \leq D$. Let $\varepsilon = \epsilon(n,D)$. The adversary does not replicate and incurs an expected cost of

$$\sum_{j=1}^{N_\varepsilon} p_j \cdot j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon \tag{6}$$

¿From (5), we have

$$\mathbf{E}[C_A(\sigma)] = \sum_{j=1}^{N_\varepsilon} p_j \cdot j + D \cdot \sum_{j=1}^{N_\varepsilon} p_j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon \tag{7a}$$

$$\geq (1+\sum_{j=1}^{N_\varepsilon} p_j) \cdot \sum_{j=1}^{N_\varepsilon} p_j \cdot j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon \tag{7b}$$

Given the p_j's, one can choose N_ε so that $\sum_{j=1}^{N_\varepsilon} p_j$ is arbitrarily close to 1. Since the series $\sum_{j=1}^{\infty} j \cdot p_j$ is bounded, one can also choose N_ε so that $\sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon$ is arbitrarily close to zero. Thus, by comparing (6) and (7b), we see that given any ε, one can choose a finite N_ε so that the ratio is as close to $(2-\varepsilon)$ as desired. □

7. Off-Line Replication and File Allocation. We show that the off-line replication problem is NP-hard, and the off-line file allocation problem on the uniform network can be solved in polynomial time.

7.1. The Off-Line Replication Problem. Awerbuch *et al.* find interesting relationships between the on-line Steiner tree problem [14, 23] and on-line FAP. We show that the (off-line) replication problem is NP-hard by using a straight-forward reduction from the Steiner tree problem [12, 16] (see Section 2 for the definition). The proof involves creating an instance for the replication problem in which $(D+1)$ requests are issued at each of the terminal nodes for the Steiner tree problem instance, forcing the optimal algorithm to replicate to these nodes.

THEOREM 7.1. *The replication problem is NP-hard, even if G is is bipartite and unweighted, or if G is planar.*

7.2. Off-line Solution For File Allocation on a Uniform Network. We show that the off-line file allocation problem on a uniform network can be solved in polynomial time by reducing it to a min-cost max-flow problem. A similar reduction was obtained by Chrobak *et al.* [8] for the k-server problem. We convert an instance of the FAP on a uniform network on nodes $1, \ldots, n$, to a min-cost max-flow problem on an acyclic layered network, N, with $O(n \cdot |\sigma|)$ nodes and $O(n^2|\sigma|)$ arcs. Initially node 1 has a copy of F. An integral maximum flow in N defines a dynamic allocation of F in the uniform network. The arcs costs in N are chosen so that the min-cost max-flow in N incurs a cost differs from the minimum cost for FAP on the uniform network by a constant. Network N is constructed as follows:

Nodes: Network N has $(|\sigma|+1)$ layers of nodes, $(2n-1)$ nodes in each layer, a source node s and a sink node t. Layer k, $0 \le k \le |\sigma|$, has nodes $\{v_1^k, \ldots, v_n^k\}$ and $\{u_1^k, \ldots, u_{(n-1)}^k\}$. Each node allows a maximum flow of one unit into and out of it. The v_j^k nodes correspond to the nodes in the uniform network. Layer k of N corresponds to the state of the uniform network after σ_k has been served.

Arcs: There is an arc going from each layer k node to each layer $(k+1)$ node, $0 \le k \le (|\sigma|-1)$; there is an arc from each layer $|\sigma|$ node to t, arc (s, v_1^0), and arcs $(s, u_j^0), 1 \le j \le (n-1)$. All the arcs have unit capacity.

A Flow: A maximum flow in N has a value of n. Given integer arc costs, there is a min-cost max-flow solution with only an integral flow of either 0 or 1 in each arc. A flow of 1 into a v_j^k represents the presence of a copy of F at node j just before request k arrives. If the flow comes from a $v_i^{(k-1)}$, it represents a copy of F being moved from node i to node j after serving the $(k-1)$st request; if the flow comes from a $u_i^{(k-1)}$, it represents a replication to node j. A flow from a $v_j^{(k-1)}$ to a u_w^k represents the copy of F at j is dropped after $\sigma_{(k-1)}$ is served. Thus an integral flow in N defines a strategy for relocating copies of F. Since there are $(n-1)$ u nodes in each layer, an integral max-flow must include a flow of 1 unit into at least one of the v nodes in each layer. This corresponds to the requirement that there is always at least a copy of F in the uniform network.

Edge Costs: Edge costs are chosen so that the optimal flow has cost equal to the optimal off-line cost for FAP minus the number of read requests, J, in σ. Arcs with one end point at s or t have zero costs. Let (a, b) be any other arc, going between layer k and $(k+1)$. Its cost is equal to the sum of its associated *movement* and *service* costs. Suppose b is $v_i^{(k+1)}$. Then (a, b)'s associated movement cost is D unless a is v_i^k. If $\sigma_{(k+1)}$ is a write at some node other than node i, the service cost is 1; if $\sigma_{(k+1)}$ is a read at node i, the service cost is -1. The costs for all other cases are zero. The

movement and service costs account for the cost for replication and serving requests, except a node is charged -1 when a read request arrives and it has a copy of F. If we add J to the cost of the optimal flow, thus charging each read request 1 in advance, the sum is equal to the cost of an optimal dynamic allocation of F.

Using the algorithm in [21] for solving the min-cost max-flow problem on acyclic networks, FAP on a uniform network can be solved in polynomial time.

THEOREM 7.2. *An optimal (off-line) file allocation on a uniform network can be found in* $O(n^3 \cdot |\sigma| \cdot (1 + \log_n |\sigma|))$ *time.*

8. Open Problems. Interesting open problems include finding a strongly competitive randomized algorithm for FAP on a uniform network. Awerbuch *et al.* [2] conjecture that if there exists a c_n-competitive algorithm for the on-line Steiner tree problem [14, 23], then there exists a $O(c_n)$-competitive deterministic algorithm for FAP. This conjecture is still open. For the migration problem, there is a gap between the best known bounds [4, 10].

REFERENCES

[1] S. ALBERS AND H. KOGA, *New On-Line Algorithms for the Page Replication Problem*, in Proceedings of the Fourth Scandinavian Workshop on Algorithmic Theory, vol. 824 of Lecture Notes in Computer Science, Aarhus, Denmark, July 1994, Springer-Verlag, pp. 25–36.

[2] B. AWERBUCH, Y. BARTAL, AND A. FIAT, *Competitive Distributed File Allocation*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 164–173.

[3] B. GAVISH AND O. R. L. SHENG, *Dynamic File Migration in Distributed Computer Systems*, Communications of the ACM, 33 (1990), pp. 177–189.

[4] Y. BARTAL, M. CHARIKAR, AND P. INDYK, *On Page Migration and Other Relaxed Task Systems*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '97, New Orleans, Louisiana, U.S.A., 1997.

[5] Y. BARTAL, A. FIAT, AND Y. RABANI, *Competitive Algorithms for Distributed Data Management*, in Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, 1992, pp. 39–50.

[6] S. BEN DAVID, A. BORODIN, R. KARP, G. TARDOS, AND A. WIGDERSON, *On the Power of Randomization in Online Algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[7] D. L. BLACK AND D. D. SLEATOR, *Competitive Algorithms for Replication and Migration Problems*, Tech. Report CMU-CS-89-201, Department of Computer Science, Carnegie Mellon University, 1989.

[8] M. CHROBAK, H. KARLOFF, T. PAYNE, AND S. VISHWANATHAN, *New Results on Server Problems*, SIAM J. Disc. Math., 4 (1991), pp. 172–181.

[9] M. CHROBAK AND L. L. LARMORE, *The Server Problem and On-line Games*, in Proceedings of the DIMACS Workshop on On-Line Algorithms, American Mathematical Society, February, 1991.

[10] M. CHROBAK, L. L. LARMORE, N. REINGOLD, AND J. WESTBROOK, *Page Migration Algorithms Using Work Functions*, in Proceedings of the 4th International Symposium on Algorithms and Computation, ISAAC '93, vol. 762 of Lecture Notes in Computer Science, Hong Kong, 1993, Springer-Verlag, pp. 406–415.

[11] D. DOWDY AND D. FOSTER, *Comparative Models of The File Assignment Problem*, Computing Surveys, 14 (1982).

[12] M. R. GAREY AND D. S. JOHNSON, *The Rectilinear Steiner Tree Problem is NP-complete*, SIAM J. Appl. Math., 32 (1977), pp. 826–834.

[13] F. K. HWANG AND D. RICHARDS, *Steiner Tree Problems*, Networks, 22 (1992), pp. 55–89.

[14] M. IMASE AND B. M. WAXMAN, *Dynamic Steiner Tree Problem*, SIAM J. Disc. Math., 4 (1991), pp. 369–384.

[15] A. R. KARLIN, M. S. MANASSE, L. A. MCGEOCH, AND S. OWICKI, *Competitive Randomized Algorithms for Non-Uniform Problems*, in Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 301–309.

[16] R. M. KARP, *Reducibility Among Combinatorial Problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[17] H. Koga, *Randomized On-line Algorithms for the Page Replication Problem*, in Proceedings of the 4th International Symposium on Algorithms and Computation, ISAAC '93, vol. 762 of Lecture Notes in Computer Science, Hong Kong, 1993, Springer-Verlag, pp. 436–445.
[18] C. Lund and N. Reingold, *Linear Programs for Randomized On-Line Algorithms*, in Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 382–391.
[19] C. Lund, N. Reingold, J. Westbrook, and D. Yan, *On-Line Distributed Data Management*, in Proceedings of the 2nd Annual European Symposium on Algorithms, ESA '94, vol. 855 of Lecture Notes in Computer Science, Utrecht, The Netherlands, 1994, Springer-Verlag, pp. 202–214.
[20] N. Reingold, J. Westbrook, and D. D. Sleator, *Randomized Competitive Algorithms for The List Update Problem*, Algorithmica, 11 (1994), pp. 15–32.
[21] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania, 1983.
[22] J. Westbrook, *Randomized Algorithms for Multiprocessor Page Migration*, SIAM J. Comput., 23 (1994), pp. 951–965.
[23] J. Westbrook and D. C. K. Yan, *The Performance of Greedy Algorithms for the On-Line Steiner Tree and Related Problems*, Math. Systems Theory, 28 (1995), pp. 451–468.
[24] P. Winter, *Steiner Problem in Networks: A Survey*, Networks, 17 (1987), pp. 129–167.

Appendix: Complete Proof for (B) in Lemma 3.7. The following is the complete proof for condition **(B)** in Lemma 3.7. It will be shown that

$$S_i(x,y) \le S_i(y,z) \tag{8}$$

holds for $i = (t+1)$ after request $(t+1)$ has arrived, by considering the change in offset values using table 1, for all possible locations of the root node r, request node w, and offset vectors. In each case, the required inequality follows from the property that all offsets satisfy $0 \le k \le l$ and the assumption that (8) holds initially for $i = t$. We will be referring to the conditions **(C.1)** to **(C.4)**, and **(D)**, that are implied by (8) for $i = t$. We use *L.H.S.* and *R.H.S.* to denote the left- and right-hand sides of the inequality under consideration, respectively.

Suppose the request is a **READ** request:

Case 1: $r \in T_x$ and $w \in T_x$
For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy}+1, l_{xy}), l_{xy})$.
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, l_{yz}), l_{yz})$.
$S_{t+1}(x,y) \le S_{t+1}(y,z)$ follows from $S_t(x,y) \le S_t(y,z)$ or **(C.1)**.

Case 2: $r \in T_y$ and $w \in T_y$
For $(x,y) : \omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, l_{xy}), 0, l_{xy})$.
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, l_{yz}), l_{yz})$.
$S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; l_{xy} - \min(k_{xy}+1, l_{xy}) \le l_{yz}$, follows from **(D)**.

Case 3: $r \in T_x$ and $w \in T_y$
For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}-1) & \text{if } k_{xy} \ge 1 \\ (\min(1, l_{xy}), 0, l_{xy}) & \text{if } k_{xy} = 0 \end{cases}$
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, l_{yz}), l_{yz})$.
$k_{xy} \ge 1$: $S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; l_{xy} - 1 \le l_{yz}$, follows from **(C.1)**.
$k_{xy} = 0$: $S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; l_{xy} - \min(1, l_{xy}) \le l_{yz}$, follows from **(C.1)**.

Case 4: $r \in T_x$ and $w \in T_z$
For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}-1) & \text{if } k_{xy} \ge 1 \\ (\min(1, l_{xy}), 0, l_{xy}) & \text{if } k_{xy} = 0 \end{cases}$
For $(z,y) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}-1) & \text{if } k_{yz} \ge 1 \\ (\min(1, l_{yz}), 0, l_{yz}) & \text{if } k_{yz} = 0 \end{cases}$

$k_{xy}, k_{yz} \geq 1 : S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} \leq l_{yz}$, follows from **(C.1)**.
$k_{xy} = k_{yz} = 0$: By **(C.4)**, $l_{xy} = l_{yz}$; hence $S_{t+1}(x,y) = S_{t+1}(y,z)$ holds in this case.
$k_{xy} \geq 1, k_{yz} = 0$: By **(C.3)**, this case cannot happen.
$k_{xy} = 0, k_{yz} \geq 1$: $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} - \min(1, l_{xy}) \leq l_{yz}$, follows from **(C.1)**.
Case 5: $r \in T_y$ and $w \in T_z$
For $(x,y) : \omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, l_{xy}), 0, l_{xy})$
For $(z,y) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}-1) & \text{if } k_{yz} \geq 1 \\ (\min(1, l_{yz}), 0, l_{yz}) & \text{if } k_{yz} = 0 \end{cases}$
$k_{yz} \geq 1$: $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} - \min(k_{xy}+1, l_{xy}) \leq l_{yz}$, follows from **(D)**.
$k_{yz} = 0$:
We need to show that $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} - \min(k_{xy}+1, l_{xy}) \leq l_{yz} - \min(1, l_{yz})$ holds, given $S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \leq l_{yz}$ holds initially. The inequality holds because $l_{xy} - l_{xy} = 0 = l_{yz} - l_{yz} \leq R.H.S.$, and $l_{xy} - k_{xy} - 1 \leq l_{yz} - 1 \leq R.H.S.$.

Suppose the request is a **WRITE** request:
Case 1: $r \in T_x$ and $w \in T_z$
For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}) & \text{if } k_{xy} \geq 1 \\ (1, 0, \min(l_{xy}+1, D)) & \text{if } k_{xy} = 0 \end{cases}$
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}) & \text{if } k_{yz} \geq 1 \\ (1, 0, \min(l_{yz}+1, D)) & \text{if } k_{yz} = 0 \end{cases}$
$k_{xy}, k_{yz} \geq 1 : S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} \leq l_{yz}$, follows from **(C.1)**.
$k_{xy} \geq 1, k_{yz} = 0$: By **(C.3)**, this case cannot happen.
$k_{xy} = k_{yz} = 0$: By **(C.4)**, $l_{xy} = l_{yz}$; hence $S_{t+1}(x,y) = S_{t+1}(y,z)$ holds in this case.
$k_{xy} = 0, k_{yz} \geq 1 : S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D) - 1 \leq l_{yz}$, follows from $S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} \leq l_{yz}$.
Case 2: $r \in T_y$ and $w \in T_z$
For $(x,y) : \omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, D), 0, \min(l_{xy}+1, D))$
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}) & \text{if } k_{yz} \geq 1 \\ (1, 0, \min(l_{yz}+1, D)) & \text{if } k_{yz} = 0 \end{cases}$
$k_{yz} \geq 1$: We need to show

$$S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \tag{9}$$

$$\min(l_{xy}+1, D) - \min(k_{xy}+1, D) \leq l_{yz} \text{ holds, given}$$

$$S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \leq l_{yz} \text{ holds initially.}$$

It can be proved as follows. If $k_{xy} = D$ then $L.H.S. = 0 \leq R.H.S.$.
If $k_{xy} < D$, then $L.H.S. \leq l_{xy} + 1 - (k_{xy}+1) \leq l_{yz} = R.H.S.$ Hence (9) holds.
$k_{yz} = 0$: We need to show that

$$S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \tag{10}$$

$$\min(l_{xy}+1, D) - \min(k_{xy}+1, D) \leq \min(l_{yz}+1, D) - 1, \text{ holds, given}$$

$$S_t(x,y) \le S_t(y,z) \;\Leftrightarrow\; l_{xy} - k_{xy} \le l_{yz} \text{ holds initially.}$$

If $l_{yz} < D$, (10) follows from (9). If $k_{xy} = D$, then $L.H.S. = 0 \le R.H.S.$. Suppose $l_{yz} \le (D-1)$ and $(k_{xy}+1) \le D$. In this case $L.H.S. \le l_{xy} - k_{xy} \le l_{yz} = R.H.S.$. Hence (10) holds.

Case 3: $r \in T_y$ and $w \in T_y$

For $(x,y) : \omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, D), 0, \min(l_{xy}+1, D))$.
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$.
We need to show that $S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; \min(l_{xy}+1, D) - \min(k_{xy}+1, D) \le \min(l_{yz}+1, D)$ holds, given $S_t(x,y) \le S_t(y,z) \;\Leftrightarrow\; l_{xy} - k_{xy} \le l_{yz}$ holds initially, which follows from (10).

Case 4: $r \in T_x$ and $w \in T_y$

For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}) & \text{if } k_{xy} \ge 1 \\ (1, 0, \min(l_{xy}+1, D)) & \text{if } k_{xy} = 0 \end{cases}$
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$
$k_{xy} \ge 1$: $S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; l_{xy} \le \min(l_{yz}+1, D)$, follows from $S_t(x,y) \le S_t(y,z) \;\Leftrightarrow\; l_{xy} \le l_{yz}$
$k_{xy} = 0$:
$S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; \min(l_{xy}+1, D) - 1 \le \min(l_{yz}+1, D)$, follows from $S_t(x,y) \le S_t(y,z) \;\Leftrightarrow\; l_{xy} \le l_{yz}$

Case 5: $r \in T_x$ and $w \in T_x$

For $(x,y) : \omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy}+1, D), \min(l_{xy}+1, D))$
For $(y,z) : \omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$
$S_{t+1}(x,y) \le S_{t+1}(y,z) \;\Leftrightarrow\; \min(l_{xy}+1, D) \le \min(l_{yz}+1, D)$, follows from $S_t(x,y) \le S_t(y,z) \;\Leftrightarrow\; l_{xy} \le l_{yz}$.

Thus we have shown that **(B)** holds after σ_{t+1} has arrived.