Accessing Nearby Copies of Replicated Objects in a Distributed Environment

C. Greg Plaxton[1] *Rajmohan Rajaraman*[1] *Andréa W. Richa*[2]

Abstract

Consider a set of shared objects in a distributed network, where several copies of each object may exist at any given time. To ensure both fast access to the objects as well as efficient utilization of network resources, it is desirable that each access request be satisfied by a copy "close" to the requesting node. Unfortunately, it is not clear how to efficiently achieve this goal in a dynamic, distributed environment in which large numbers of objects are continuously being created, replicated, and destroyed.

In this paper, we design a simple randomized algorithm for accessing shared objects that tends to satisfy each access request with a nearby copy. The algorithm is based on a novel mechanism to maintain and distribute information about object locations, and requires only a small amount of additional memory at each node. We analyze our access scheme for a class of cost functions that captures the hierarchical nature of wide-area networks. We show that under the particular cost model considered: (i) the expected cost of an individual access is asymptotically optimal, and (ii) if objects are sufficiently large, the memory used for objects dominates the additional memory used by our algorithm with high probability. We also address dynamic changes in both the network as well as the set of object copies.

[1]Department of Computer Science, University of Texas at Austin, Austin, TX 78712. Supported by the National Science Foundation under Grant No. CCR–9504145. Email: {`plaxton`, `rraj`}`@cs.utexas.edu`.

[2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. Supported by Bruce Maggs's National Young Investigator Award under Grant No. CCR-94-57766. Email: `aricha@cs.cmu.edu`.

1 Introduction

The advent of high-speed networks has made it feasible for a large number of geographically dispersed computers to cooperate and share objects (e.g, files, words of memory). This has resulted in the implementation of large distributed databases like the World Wide Web on wide-area networks. The large size of the databases and the rapidly growing demands of the users has in turn overloaded the underlying network resources. Hence, an important goal is to make efficient use of network resources when providing access to shared objects.

As one might expect, the task of designing efficient algorithms for supporting access to shared objects over wide-area networks is extremely challenging, both from a practical as well as a theoretical perspective. With respect to any interesting measure of performance (e.g., latency, throughput), the optimal bound achievable by a given network is a complex function of many parameters, including edge delays, edge capacities, buffer space, communication overhead, patterns of user communication, and so on. Ideally, we would like to take all of these factors into account when optimizing performance with respect to a given measure. However, such a task may not be feasible in general because the many network parameters interact in a fairly complex manner. For this reason, we adopt a simplified model in which the combined effect of the detailed network parameter values is assumed to be captured by a single function that specifies the cost of communicating a fixed-length message between any given pair of nodes. We anticipate that analyzing algorithms under this model will significantly aid in the design of practical algorithms for modern distributed networks.

Accessing shared objects. Consider a set $\mathcal{A}$ of m objects being shared by a network G, where several copies of each object may exist. In this paper, we consider the basic problem of *reading* objects in $\mathcal{A}$. Motivated by the need for efficient network utilization, we seek algorithms that minimize the cost of the read operation. We do not address the *write* operation, which involves the additional consideration of maintaining consistency among the various object copies. The problem of consistency, although an important one, is separate from our main concern, namely, that of studying locality. Our results for the read apply for the write in scenarios where consistency either is not required or is enforced by an independent mechanism.

We differentiate between *shared* and *unshared* copies of objects. A copy is shared if any node can read this copy; it is unshared if only the node which holds the copy may read it. We say that a node u *inserts* (resp., *deletes*) a copy of object A (that u holds) if u declares the copy shared (resp., unshared).

We refer to the set of algorithms for read, insert, and delete operations as an *access scheme*. Any access scheme that efficiently supports these operations incurs an overhead in memory. It is desirable that this overhead be small, not only because of space considerations, but also because low overhead usually implies fast adaptability to changes in the network topology or in the set of object copies.

The main difficulty in designing an access scheme that is efficient with respect to both time and space is the competing considerations of these measures. For example, consider an access scheme in which each node stores the location of each copy of each object in the network. This allows very fast read operations since a node can easily determine the location of the closest copy of any desired object. However, such an access scheme is impractical because: (i) it incurs a prohibitively large memory overhead, and (ii) every node of the network has to be informed whenever a copy of an object is inserted or deleted. At the other extreme, one might consider an access scheme using no additional memory. In this case insert and delete operations are fast, but read operations are costly since it may be necessary to search the entire network in order to locate a copy of some desired object.

Our access scheme. We design a simple randomized access scheme that exploits locality and distributes control information to achieve low overhead in memory. The central part of our access scheme is a mechanism to maintain and locate the addresses of copies of objects. For a single object, say A, we can provide such a mechanism by the following approach. We embed an n-node "virtual" height-balanced tree T one-to-one into the network. Each node u of the network maintains information associated with the copies of A residing in the set of nodes that form the subtree of T rooted at u. Given the embedding of T, the read operation may be easily defined as follows. When a node u attempts to read A, u first checks its local memory for a copy of A or information about copies of A in the subtree of T rooted at u. If u is unable to locate any copy on the basis of local information, it forwards its request to its parent in T.

Naive extensions of the above approach to account for all objects require significant overhead in memory for control information at individual nodes. We overcome this problem by designing a novel method to embed the different trees associated with different objects. Our embedding enables us to define simple algorithms for read, insert, and delete operations, and to prove their efficiency for a class of cost functions that is appropriate for modeling wide-area networks.

The cost model. As indicated above, we assume that a given function determines the cost of communication between each pair of nodes in the network. Our analysis is geared towards a restrictive class of cost functions which we believe to be of practical interest. The precise set of assumptions that we make with respect to the cost function is stated in Section 2. Our primary assumption is that for all nodes x and costs r, the ratio of the number of nodes within cost $2r$ of node x to the number of nodes within cost r of node x is bounded from above and below by constants greater than 1 (unless the entire network is within cost $2r$ of node x, in which case the ratio may be as low as 1).

There are several important observations we can make concerning this primary assumption on the cost function. First, a number of commonly studied fixed-connection network families lead naturally to cost functions satisfying this assumption. For example, fixed-dimension meshes satisfy this assumption if the cost of communication between two nodes is defined as the minimum number of hops between them. As another example, fat-tree topologies can be made to satisfy our assumption if the cost of communication between two nodes is determined by the total cost of a shortest path between them, where the cost assigned to individual edges grows at an appropriate geometric rate as we move higher in the tree. The latter example is of particular interest here, because of all the most commonly studied fixed-connection network families, the fat-tree may provide the most plausible approximation to the structure of current wide-area networks.

Even so, it is probably inappropriate to attempt to model the Internet, say, with any kind of uniform topology, including the fat-tree. Note that our assumption on the cost function is purely "local" in nature, and allows for the possibility of a network with a highly irregular global structure. This may be the most important characteristic of our cost model.

Performance bounds. We show that our access scheme achieves optimality or near-optimality in terms of several important complexity measures for the restricted class of cost functions discussed above. In particular, our scheme achieves the following bounds:

- The expected cost for any read request is asymptotically optimal.

- If the number of objects that can be stored at each node is q, then the additional memory required is $O(q \log^2 n)$ words whp[1], where a word is an $O(\log n)$-bit string. Thus, if the objects are sufficiently large, i.e., $\Omega(\log^2 n)$ words, the memory for objects dominates the additional memory.
- The number of nodes that need to be updated upon the addition or removal of a node is $O(\log n)$ expected and $O(\log^2 n)$ whp.
- The expected cost of an insert (resp., delete) operation at node u is $O(C)$ (resp., $O(C \log n)$), where C is the maximum cost of communicating a single word message between any two nodes.

An obvious shortcoming of our analysis is that it only applies to the restricted class of cost functions discussed above. While we do not expect that all existing networks fall precisely within this restricted class, we stress that: (i) our access scheme is well-defined, and functions correctly, for arbitrary networks, and (ii) we expect that our access scheme would have good practical performance on any existing network. (Although we have not attempted to formalize any results along these lines, it seems clear that our performance bounds would only degrade significantly in the presence of a large number of non-trivial violations of our cost function assumptions.)

Related work. The basic problem of sharing memory in distributed systems has been studied extensively in different forms. Most of the earlier work in this area, e.g., emulations of PRAM on completely-connected distributed-memory machines (e.g., [9, 16]) or bounded-degree networks (e.g., [14]), and algorithms for providing concurrent access to a set of shared objects [12], assume that each of the nodes of the network has knowledge of a hash function that indicates the location of any copy of any object.

The basic problem of locating an object arises in every distributed system [10], and was formalized by Mullender and Vitányi [11] as an instance of the distributed matchmaking problem. Awerbuch and Peleg [3], and subsequently Bartal et al. [4] and Awerbuch et al. [1], give near-optimal solutions in terms of cost to a related problem by defining sparse-neighborhood covers of graphs. Their studies do not address the overhead due to control information and hence, natural extensions of their results to our problem may require an additional memory of m words at some node. However, we note that their schemes are designed for arbitrary cost functions, whereas we have focused on optimizing performance for a restricted class of cost functions.

[1] We use the abbreviation "whp" throughout the paper to mean "with high probability" or, more precisely, "with probability $1 - n^{-c}$, where n is the number of nodes in the network and c is a constant that can be set arbitrarily large by appropriately adjusting other constants defined within the relevant context."

In recent work, access schemes for certain Internet applications have been described in [7, 8, 17]. Some of the ideas in our scheme are similar to those in [17]; however, the two schemes differ considerably in the details. Moreover, the schemes of [7] and [17] have not been analyzed. As in our study, the results of [8] concerning locality assume a restricted cost model. However, their cost model, which is based on the ultrametric, is different from ours. Also, their algorithms are primarily designed for problems associated with "hot spots" (i.e., popular objects).

A closely related problem is that of designing a dynamic routing scheme for networks [2, 5]. Such a scheme involves maintaining routing tables at different nodes of the network in much the same way as our additional memory. However, in routing schemes the size of additional memory is a function of network size, i.e., n, while in our problem the overhead is primarily a function of m. Straightforward generalizations of routing schemes result in access schemes that require an additional memory of m words at each node.

The remainder of this paper is organized as follows. Section 2 defines the model of computation. Section 3 formally describes our access scheme. Section 4 contains a formal statement of the main results. Section 5 analyzes the algorithm and establishes the main results. Section 6 discusses directions for future research.

2 Model of Computation

We consider a set V of n nodes, each with its own local memory, sharing a set $\mathcal{A}$ of $m = \text{poly}(n)$ objects. We define our model of computation by characterizing the following aspects of the problem: (i) objects, (ii) communication, (iii) local memory, (iv) local computation, and (v) complexity measures.

Objects. Each object A has a unique $(\log m)$-bit identification. For i in $[\log m]$, we denote the ith bit of the identification of A by A^i. (For any positive integer x, we use $[x]$ to denote the set $\{0, \ldots, x-1\}$.) Each object A consists of $\ell(A)$ words, where a word is an $O(\log n)$-bit string.

Communication. Nodes communicate with one another by means of messages; each message consists of at least one word. We assume that the underlying network supports reliable communication.

We define the cost of communication by a function $c : V^2 \mapsto \mathbf{R}$. For any two nodes u and v in V, $c(u, v)$ is the cost of transmitting a single-word message from u to v. We assume that c is symmetric and satisfies the triangle inequality. We also assume for simplicity that for u, v, and w in V, $c(u, v)$ equals $c(u, w)$ iff v equals w.

The cost of transmitting a message of length ℓ from node u to node v is given by $f(\ell)c(u,v)$, where $f : \mathbf{N} \mapsto \mathbf{R}^+$ is any non-decreasing function such that $f(1)$ equals 1.

Given any u in V and any real r, let $M(u,r)$ denote the set $\{v \in V : c(u,v) \le r\}$. We refer to $M(u,r)$ as the *ball* of *radius* r around u. We assume that there exist real constants $\delta > 8$ and Δ such that for any node u in V and any real $r \ge 1$, we have:

$$\min\{\delta|M(u,r)|, n\} \le |M(u,2r)| \le \Delta|M(u,r)| \qquad (1)$$

Local Memory. We partition the local memory of each node u into two parts. The first part, the *main memory*, stores objects. The second part, the *auxiliary memory*, is for storing possible control information.

Local Computation. There is no cost associated with local computation. (Although the model allows an arbitrary amount of local computation at zero cost, our algorithm does not perform any particularly complex local operations.)

Complexity measures. We evaluate any solution on the basis of four different complexity measures. The first measure is the cost of reading an object. The second measure is the size of the auxiliary memory at any node. The remaining two measures concern the dynamic nature of the problem, where we address the complexity of inserting or deleting a copy of an object and adding or removing a network node. The third measure is the cost of inserting or deleting a copy of an object. The fourth measure is *adaptability*, which is defined as the number of nodes whose auxiliary memory is updated upon the addition or removal of a node. (Our notion of adaptability is analogous to that of [5].)

3 The Access Scheme

In this section, we present our access scheme for shared objects. We assume that n is a power of 2^b, where b is a fixed positive integer to be specified later. For each node x in V, we assign a label independently and uniformly at random from $[n]$. For i in $[\log n]$, let x^i denote the ith bit of the label of x. Note that the label of a node x is independent of the $(\log n)$-bit unique identification of the node. For all x in V (resp., A in $\mathcal{A}$), we define $x[i] = x^{(i+1)b-1} \cdots x^{ib}$ (resp., $A[i] = A^{(i+1)b-1} \cdots A^{ib}$), for i in $[(\log n)/b]$. We also assign a total order to the nodes in V, given by the bijection $\beta : V \rightarrow [n]$. We partition the auxiliary memory of each node into two parts, namely the *neighbor table* and the *pointer list* of the node.

Neighbor table. For each node x, the neighbor table of x consists of $(\log n)/b$ levels. The ith level of the table, i in $[(\log n)/b]$, consists of *primary*, *secondary*, and *reverse* (i,j)*-neighbors*, for all j in $[2^b]$. The *primary* (i,j)*-neighbor* y of x is such that $y[k] = x[k]$ for all k in $[i]$, and either: (i) $i < (\log n)/b - 1$ and y is the node of minimum $c(x,y)$ such that $y[i] = j$, if such a node exists, or (ii) y is the node with largest $\beta(y)$ among all nodes z such that $z[i]$ matches j in the largest number of rightmost bits. Let d be a fixed positive integer, to be specified later. Let y be the primary (i,j)-neighbor of x. If $y[i] = j$, then let $W_{i,j}$ denote the set of nodes w in $V \setminus \{y\}$ such that $w[k] = x[k]$, for k in $[i]$, $w[i] = j$, and $c(x,w)$ is at most $d \cdot c(x,y)$. Otherwise, let $W_{i,j}$ be the empty set. The set of *secondary* (i,j)*-neighbors* of x is the subset U of $\min\{d, |W_{i,j}|\}$ nodes u with minimum $c(x,u)$ in $W_{i,j}$; that is, $c(x,u)$ is at most $c(x,w)$, for all w in $W_{i,j}$, and for all u in U. A node w is a *reverse* (i,j)*-neighbor* of x iff x is a primary (i,j)-neighbor of w.

Pointer list. Each node x also maintains a pointer list $Ptr(x)$ with pointers to copies of some objects in the network. Formally, $Ptr(x)$ is a set of triples (A,y,k), where A is in $\mathcal{A}$, y is a node that holds a copy of A, and k is an upper bound on the cost $c(x,y)$. We maintain the invariant that there is at most one triple associated with any object in $Ptr(x)$. The pointer list of x may only be updated as a result of insert and delete operations. All the pointer lists can be initialized by inserting each shared copy in the network at the start of the computation. We do not address the cost of initializing the auxiliary memories of the nodes.

Let r be the node with highest $\beta(r)$ such that there exists i in $[(\log n)/b]$ satisfying: (i) $r[k] = A[k]$ for all k in $[i]$, (ii) $r[i]$ matches $A[i]$ in the largest number of rightmost bits, and (iii) if $i < (\log n)/b - 1$, there is no node y with $y[k] = A[k]$ for all k in $[i+1]$. We call r the *root* node for object A. The uniqueness of the root node for each A in $\mathcal{A}$ is crucial to guarantee the success of every read operation.

In this section and throughout the paper, we use the notation $\langle\alpha\rangle_k$ to denote the sequence (of length $k+1$) $\alpha_0, \alpha_1, \ldots, \alpha_k$ (of length $k+1$). When clear from the context, k will be omitted. In particular, a *primary neighbor sequence* for A is a maximal sequence $\langle u\rangle_k$ such that u_0 is in V, u_k is the root node for A, and u_{i+1} is the primary $(i, A[i])$-neighbor of u_i, for all i. It is worth noting that the sequence $\langle u\rangle$ is such that the label of node u_i satisfies $(u_i[i-1], \ldots, u_i[0]) = (A[i-1], \ldots, A[0])$, for all i. We now give an overview of the read, insert, and delete operations.

Read. Consider a node x attempting to read an object A. The read operation proceeds by successively forwarding the *read request* for object A originating at node x along the primary neighbor sequence $\langle x\rangle$ for A with $x_0 = x$. When forwarding the read request, node x_{i-1} also informs x_i of the current best upper bound k on the cost of sending a copy of A to x. On receiving the read request with associated upper bound k, node x_i proceeds as follows. If x_i is the root node for A, then x_i requests that the copy of A associated with k be sent to x. Otherwise, x_i communicates with its primary and secondary $(i, A[i])$-neighbors to check whether the pointer list of any of these neighbors has an entry (A, z, k_1) such that k_1 is at most k. Then, x_i updates k to be minimum of k and the smallest value of k_1 thus obtained (if any). If k is within a constant factor of the cost of following $\langle x \rangle$ up to x_i, that is, k is $O(\sum_{j=0}^{i-1} c(x_j, x_{j+1}))$, then x_i requests that the copy of A associated with the upper bound k be sent to x. Otherwise, x_i forwards the read request to x_{i+1}.

Insert. An *insert request* for object A generated by node y updates the pointer lists of some nodes that form a prefix subsequence of the primary neighbor sequence $\langle y \rangle$ for A with $y_0 = y$. When such an update arrives at a node y_i by means of an insert message, y_i updates its pointer list if the upper bound $\sum_{j=0}^{i-1} c(y_j, y_{j+1})$ on the cost of getting object A from y, is smaller than the current upper bound associated with A in this list. In other words, y_i updates $Ptr(y_i)$ if $(A, \cdot, \cdot)$ is not in this list, or if $(A, \cdot, k)$ is in $Ptr(y_i)$ and k is greater than $\sum_{j=0}^{i-1} c(y_j, y_{j+1})$. Node y_i forwards the insert request to node y_{i+1} only if $Ptr(y_i)$ is updated.

Delete. A *delete request* for object A generated by node y eventually removes all triples of the form $(A, y, \cdot)$ from the pointer lists $Ptr(y_i)$, where $\langle y \rangle$ is the primary neighbor sequence for A with $y_0 = y$, making the copy of A at y unavailable to other nodes in the network. Upon receiving such a request by means of a delete message, node y_i checks whether the entry associated with A in its pointer list is of the form $(A, y, \cdot)$. In case it is not, the delete procedure is completed and we need to proceed no further in updating the pointer lists in $\langle y \rangle$. Otherwise, y_i deletes this entry from its pointer list, and checks for entries associated with A in the pointer lists of its reverse $(i-1, A[i-1])$-neighbors. If an entry is found, y_i updates $Ptr(y_i)$ by adding the entry $(A, w, k + c(w, y_i))$, where w is the reverse $(i-1, A[i-1])$-neighbor of y_i with minimum upper bound k associated with A in its pointer list. A delete message is then forwarded to y_{i+1}.

The read, insert, and delete procedures for an object A are formally described in Figure 1. The messages and requests in the figure are all with respect to object A. A *read request* is generated by node x when x $(= x_0)$ sends a message $Read(x, \infty, \cdot)$ to itself, if x does not hold a copy of A. A read message $Read(x, k, y)$ indicates a read request for object A generated at node x, and that the current best upper bound on the cost of reading $A$is k and such a copy resides at y. An *insert* (resp., *delete*) *request* is generated when node y $(= y_0)$ sends a message $Insert(y, 0)$ (resp., $Delete(y)$) to itself. An insert message $Insert(y, k)$ indicates to its recipient node z that the best known upper bound on the cost incurred by z to read the copy of A located at y is k. We assume that y holds a copy of A and that this copy is unshared (resp., shared) when an insert (resp., delete) request for A is generated at y.

The correctness of our access scheme follows from the two points below:

(1) The insert and delete procedures maintain the following invariants. For any A in $\mathcal{A}$ and any y in V, there is at most one entry associated with A in the pointer list of y. If y holds a shared copy of A and $\langle y \rangle$ is the primary neighbor sequence for A with $y_0 = y$, then: (i) there is an entry associated with A in the pointer list of every node in $\langle y \rangle$, and (ii) the nodes that have a pointer list entry associated with the copy of A at y form a prefix subsequence of $\langle y \rangle$. The preceding claims follow directly from the insert and delete procedures as described.

(2) Every read request for any object A by any node x is successful. That is, it locates and brings to x a shared copy of A, if such a copy is currently available. The read operation proceeds by following the primary neighbor sequence $\langle x \rangle$ for A with $x_0 = x$, until either a copy of A is located or the root for A is reached. By point (1), there exists a shared copy of A in the network if and only if the root for A has a pointer to it.

4 Results

In this section, we formally state the main results of our access scheme. In Theorems 1, 2, 3, and 4, we prove bounds on the cost of a read, the cost of an insert or delete, the size of auxiliary memory, and the adaptability of our access scheme. Let C denote $\max\{c(u, v) : u, v \in V\}$.

Theorem 1 *Let x be any node in V and let A be any object in $\mathcal{A}$. If y is the nearest node to x that holds a shared copy of A, then the expected cost of a read operation is $O(f(\ell(A))c(x, y))$.*

When a node x tries to read an object A which has currently no shared copy in the network, then the expected cost of the associated operation is $O(C)$.

Theorem 2 *The expected cost of an insert operation is $O(C)$, and that of a delete operation is $O(C \log n)$.*

Theorem 3 *Let q be the number of objects that can be stored in the main memory of each node. The size of the auxiliary memory at each node is $O(q \log^2 n)$ words whp.*

Theorem 4 *The adaptability of our scheme is $O(\log n)$ expected and $O(\log^2 n)$ whp.*

Action of x_i on receiving a message Read(x,k,y):

If $i > 0$ and $x_i[i{-}1] \neq A[i{-}1]$, or $i = (\log n)/b - 1$ (that is, x_i is the root for A) then:

- Node x_i sends a message *Satisfy*(x) to node v such that $(A, v, \cdot)$ is in *Ptr*(x_i), requesting it to send a copy of A to x. If *Ptr*(x_i) has no such entry, then there are no shared copies of A.

Otherwise:

- Let U be the set of secondary $(i, A[i])$-neighbors of x_i. Node x_i requests a copy of A with associated upper bound at most k from each node in $U \cup \{x_{i+1}\}$.
- Each node u in $U \cup \{x_{i+1}\}$ responds to the request message received from x_i as follows: if there exists an entry (A, v, q_v) in *Ptr*(u) and if $q'_v = q_v + c(x_i, u) + \sum_{j=0}^{i-1} c(x_j, x_{j+1})$ is at most k, then u sends a success message *Success*(v, q'_v) to x_i.
- Let U' be the set of nodes u from which x_i receives a response message *Success*(u, k_u). If U' is not empty, then x_i updates (k, y) to be (k_z, z), where z is a node with minimum k_u over all u in U'.
- If $k = O(\sum_{j=0}^{i-1} c(x_j, x_{j+1}))$ then x_i sends a message *Satisfy*(x) to node y, requesting y to send a copy of A to x. Otherwise, x_i forwards a message *Read*(x, k, y) to x_{i+1}.

Action of y_i on receiving a message Delete(y):

If $(A, y, \cdot)$ is in *Ptr*(y_i), then:

- Let U be the set of reverse $(i{-}1, A[i{-}1])$-neighbors of y_i. Node y_i removes $(A, y, \cdot)$ from *Ptr*(y_i), and requests a copy of A from each u in U.
- Each u in U responds to the request message from y_i by sending a message *Success*$(v, q_v + c(y_i, u))$ to y_i iff (A, v, q_v) is in *Ptr*(u).
- Let U' be the set of nodes u such that y_i receives a message *Success*(u, k_u) in response to the request message it sent. If $|U'| > 0$ then y_i inserts (A, w, k_w) into *Ptr*(y_i), where w is the node in U' such that $k_w \leq k_u$, for all u in U'.
- If $y_i[i{-}1] = A[i{-}1]$ then y_i sends a message *Delete*(y) to y_{i+1}.

Action of y_i on receiving a message Insert(y,k):

If $(A, \cdot, \cdot)$ is not in *Ptr*(y_i), or $(A, \cdot, k')$ is in *Ptr*(y_i) and $k' > k$, then:

- Node y_i accordingly creates or replaces the entry associated with A in *Ptr*(y_i) by inserting (A, y, k) into this list.
- If $y_i[i{-}1] = A[i{-}1]$ then y_i sends a message *Insert*$(y, k + c(y_i, y_{i+1}))$ to y_{i+1}.

Figure 1: Actions on receiving messages *Read, Insert*, and *Delete* for object A.

5 Analysis

In this section, we analyze the access scheme described in Section 3, and establish the main results described in Section 4. Section 5.1 presents some useful properties of balls. Section 5.2 presents properties of primary and secondary neighbors. Section 5.3 presents the proofs of Theorems 1 and 2. Sections 5.4 and 5.5 present the proofs of Theorems 3 and 4, respectively. Due to space constraints, we omit most of the proofs in this abstract. We refer the reader to the full version of the paper [13] for complete proofs of the results stated in Section 4.

Several constants appear in the model, the algorithms, and the analysis: δ and Δ appear in the model, b and d appear in the algorithms, γ and ε appear in the analysis. We set b, d, γ, and ε such that: $d, \gamma \ll 2^b$, $\varepsilon < 1/(10 \cdot 2^{b \log_\delta 2})$, and b sufficiently large to obtain the desired results. We refer the reader to the full version of the paper [13] for the precise relationships among the constants.

5.1 Properties of Balls

Given any u in V and any integer k in $[1, n]$, let $N(u, k)$ denote the unique set of k nodes such that for any v in $N(u, k)$ and w not in $N(u, k)$, $c(u, v)$ is less than $c(u, w)$. (For integers a and b, we let $[a, b]$ denote the set $\{k \in \mathbf{Z} : a \leq k \leq b\}$.) We refer to $N(u, k)$ as the *ball of size* k around u. For convenience, if k is greater than n, we let $N(u, k)$ be V.

Lemma 5.1 *Let u, v, and w be in V and let k_0 and k_1 be positive integers. If v is in $N(u, k_0)$ and w is in $N(v, k_1)$, then w is in $N(u, \Delta k_0 + \Delta^2 k_1)$.* ∎

Given any subset S of V and some node u in S, let $q(u, S)$ (resp., $r(u, S)$) denote the largest (resp., smallest) integer k such that $N(u, k)$ is a subset (resp., superset) of S. Let $Q(u, S)$ and $R(u, S)$ denote $N(u, q(u, S))$ and $N(u, r(u, S))$, respectively.

Lemma 5.2 *Let u be in V, let S be a subset of V, and let k be in $[1, n]$. Then $N(u, k)$ is a subset (resp., superset) of S iff $N(u, k)$ is a subset of $Q(u, S)$ (resp., superset of $R(u, S)$).* ∎

Lemma 5.3 *Let u belong to V, and let k_0 and k_1 denote positive integers such that $k_1 \geq \Delta^2 k_0$. For any v in $N(u, k_0)$, $q(v, N(u, k_1))$ is at least k_1/Δ and $R(v, N(u, k_1))$ is a subset of $N(u, \Delta k_1)$.* ∎

We refer to any predicate on V that only depends on the label of v as a *label predicate*. Given any node u

in V and a label predicate $\mathcal{P}$ on V, let $p(u, \mathcal{P})$ denote the node v such that: (i) $\mathcal{P}(v)$ holds, and (ii) for any node w such that $\mathcal{P}(w)$ holds, $c(u, v)$ is at most $c(u, w)$. (We let $p(u, \mathcal{P})$ be null if such a v is not defined.) Let $P(u, \mathcal{P})$ be $M(u, c(u, p(u, \mathcal{P})))$, if $p(u, \mathcal{P})$ is not null, and V, otherwise.

For u in V and i in $[(\log n)/b]$, let $\lambda_{\geq i}(u)$ denote the string of $(\log n - ib)$ bits given by $u[(\log n)/b-1] \cdots u[i+1]u[i]$. For convenience, we let $\lambda_{>i}(u)$ denote $\lambda_{\geq i+1}(u)$. For all i and all u in V, let $\mathcal{P}_i(u)$ hold iff $u[i] = A[i]$. For all i and all u in V, let $\mathcal{P}_{<i}(u)$ denote $\wedge_{j \in [i]} \mathcal{P}_j(u)$. Let $\mathcal{P}_{\leq i}(u)$, $\mathcal{P}_{>i}(u)$, and $\mathcal{P}_{\geq i}(u)$ be defined similarly. (We note that for u and v in V and nonnegative integers i and j, if $(u \neq v) \vee ((u = v) \wedge (i \neq j))$, then $\mathcal{P}_i(u)$ and $\mathcal{P}_j(v)$ are independent random variables. Also, each of the predicates defined above is a label predicate.)

Lemma 5.4 *Let S and S' be subsets of V and let u belong to S. Let $\mathcal{P}$ be a label predicate on V and for each v in S', let $\lambda_{\geq 0}(v)$ be chosen independently and uniformly at random.*

1. *Given that $P(u, \mathcal{P}) \subseteq S$, we have: (i) for each node v in $S' \setminus P(u, \mathcal{P})$, $\lambda_{\geq 0}(v)$ is independently and uniformly random, and (ii) for each node v in $P(u, \mathcal{P}) \setminus \{p(u, \mathcal{P})\}$, $\mathcal{P}(v)$ is false.*
2. *Given that $P(u, \mathcal{P}) \not\subseteq S$, we have: (i) for each node v in $S' \setminus Q(u, S)$, $\lambda_{\geq 0}(v)$ is independently and uniformly random, and (ii) for each node v in $Q(u, S)$, $\mathcal{P}(v)$ is false.*
3. *Given that $P(u, \mathcal{P}) \supseteq S$, we have: (i) for each node v in $S' \setminus R(u, S)$, $\lambda_{\geq 0}(v)$ is independently and uniformly random, and (ii) for each node v in $R(u, S) \setminus \{p(u, \mathcal{P})\}$, $\mathcal{P}(v)$ is false.*

The following claim follows from repeated application of Part 1 of Lemma 5.4.

Corollary 5.4.1 *Let S be an arbitrary subset of V, let i be in $[(\log n)/b-1]$, and let S' be a subset of V such that $\lambda_{\geq 0}(u)$ is independently and uniformly random for each u in S'. Given a sequence of nodes $u_0, u_1, \ldots, u_i$ such that for all j in $[i]$, $u_{j+1} = p(u_j, \mathcal{P}_{\leq j})$ and $P(u_j, \mathcal{P}_{\leq j}) \subseteq S$, we have:*

1. *For each node u in $S' \setminus \cup_{j \in [i]} P(u, \mathcal{P}_{\leq j})$, $\lambda_{\geq 0}(u)$ is independently and uniformly random.*
2. *The random variable $\lambda_{>i}(u_i)$ is independently and uniformly random and for each node u in $\cup_{j \in [i]} P(u_j, \mathcal{P}_{\leq j}) \setminus \{u_i\}$, $\mathcal{P}_{\leq i}(u)$ is false;* ■

5.2 Properties of Neighbors

In this section, we establish certain claims concerning the different types of neighbors that are defined in Section 3. We differentiate between root and nonroot primary (i, j)-neighbors. A root primary (i, j)-neighbor w of v is a primary (i, j)-neighbor w of v such that $w[i] \neq j$ or $i = (\log n)/b - 1$. A primary neighbor that is not a root primary neighbor is a nonroot primary neighbor.

Lemma 5.5 *Let u and v be in V, and let k denote $|M(u, c(u, v))|$. For any j in $[2^b]$, we have: (i) for any i in $[(\log n)/b - 1]$, the probability that u is a primary (i, j)-neighbor of v is at most $e^{-((k/\Delta)-2)/2^{(i+1)b}}$, and (ii) for any i in $[(\log n)/b]$, the probability that u is a root primary (i, j)-neighbor of v is at most $e^{-n/2^{(i+1)b}}$.* ■

Corollary 5.5.1 *Let u and v be in V, let i be in $[(\log n)/b]$, and let j be in $[2^b]$. If u is a primary (i, j)-neighbor of v, then v is in $N(u, O(2^{ib} \log n))$ whp.* ■

For any u in V, let a_u denote the total number of triples (i, j, v) such that i belongs to $[(\log n)/b]$, j belongs to $[2^b]$, v belongs to V, and u is a primary or secondary (i, j)-neighbor of v. Lemma 5.6 is used in the proof of Theorem 4, while Corollary 5.6.1 is used in the proofs of Theorems 2 and 3.

Lemma 5.6 *Let u be in V and let i be in $[(\log n)/b]$. Then, the number of nodes of which u is an ith level primary neighbor is $O(\log n)$ whp. Also, $E[a_u] = O(\log n)$ and a_u is $O(\log^2 n)$ whp.* ■

Corollary 5.6.1 *For any u in V, the total number of reverse neighbors of u is $O(\log^2 n)$ whp, and expected $O(\log n)$.* ■

For any u and v in V and i in $[(\log n)/b]$, v is said to be an *i-leaf* of u if there exists a sequence $v = v_0, v_1, \ldots, v_{i-1}, v_i = u$, such that for all j in $[i]$, v_{j+1} is a primary $(j, v_{j+1}[j])$-neighbor of v_j. Lemma 5.7 is used in the proof of Theorem 3.

Lemma 5.7 *Let u belong to V, and let i be in $[(\log n)/b]$. Then the number of i-leaves of u is $O(2^{ib} \log n)$ whp.* ■

5.3 Cost of operations

Consider a read request originating at node x for an object A. Let y denote a node that has a copy of A. In the following, we show that the expected cost of a read operation is $O(f(\ell(A))c(x, y))$. Letting y to be the node with minimum $c(x, y)$ among the set of nodes that have a copy of A, this bound implies that the expected cost is asymptotically optimal.

Let $\langle x \rangle$ and $\langle y \rangle$ be the primary neighbor sequences for A with $x_0 = x$ and $y_0 = y$, respectively. For any nonnegative integer i, let A_i (resp., D_i) denote the ball of smallest radius around x_i (resp., y_i) that contains x_{i+1} (resp., y_{i+1}). Let B_i (resp., E_i) denote the set $\cup_{0\le j\le i}A_j$ (resp., $\cup_{0\le j\le i}D_j$). Let C_i denote the ball of smallest radius around x_i that contains all of the secondary $(i,A[i])$-neighbors of x_i. For convenience, we define $B_{-1}=E_{-1}=\emptyset$.

It is useful to consider an alternative view of x_i, y_i, A_i, and D_i. For any nonnegative i, if x_{i+1} (resp., y_{i+1}) is not the root node for A, then x_{i+1} (resp., y_{i+1}) is $p(x_i,\mathcal{P}_{\le i})$ (resp., $p(y_i,\mathcal{P}_{\le i})$) and A_i (resp., D_i) is $P(x_i,\mathcal{P}_{\le i})$ (resp., $P(y_i,\mathcal{P}_{\le i})$).

Let γ be an integer constant that is chosen later appropriately. For any nonnegative integer i and any integer j, let X_i^j (resp., Y_i^j) denote the ball $N(x,\gamma^j 2^{(i+1)b})$ (resp., $N(y,\gamma^j 2^{(i+1)b})$). Let i^* denote the least integer such that the radius of $X_{i^*}^1$ is at least $c(x,y)$. Let a_i (resp., b_i) denote the radius of X_i^1 (resp., Y_i^1).

Lemma 5.8 *For all i such that $i\ge i^*$, X_i^2 is a superset of Y_i^1.* ■

Lemma 5.9 *For all i in $[(\log n)/b-2]$, we have $2^{b\log_\Delta 2}a_i \le a_{i+1} \le 2^{b\log_\delta 2}a_i$ and $2^{b\log_\Delta 2}b_i \le b_{i+1} \le 2^{b\log_\delta 2}b_i$. For $i=(\log n)/b-2$, we have $a_{i+1}\le 2^{b\log_\delta 2}a_i$ and $b_{i+1}\le 2^{b\log_\delta 2}b_i$. Also, a_{i^*} and b_{i^*} are both $O(c(x,y))$.* ■

We define two sequences $\langle s_i\rangle$ and $\langle t_i\rangle$ of nonnegative integers as follows:

$$s_i = \begin{cases} 0 & \text{if } B_i\subseteq X_i^1,\ A_i\supseteq X_i^{-1},\ C_i\supseteq X_i^2,\\ 1 & \text{if } B_i\subseteq X_i^1,\ A_i\supseteq X_i^{-1},\ C_i\not\supseteq X_i^2,\\ 2 & \text{if } B_i\subseteq X_i^1,\ A_i\not\supseteq X_i^{-1}, \text{ and}\\ 3+j & \text{if } 0\le j\le i,\ B_{i-j}\not\subseteq X_i^1,\ B_{i-j-1}\subseteq X_i^1.\end{cases}$$

$$t_i = \begin{cases} 0 & \text{if } E_i\subseteq Y_i^1, \text{ and}\\ 1+j & \text{if } 0\le j\le i,\ E_{i-j}\not\subseteq Y_i^1,\ E_{i-j-1}\subseteq Y_i^1.\end{cases}$$

Lemma 5.10 *If s_i is in $\{0,1,2\}$, then $c(x_i,x_{i+1})$ is $O(a_i)$. If t_i is 0, then $c(y_i,y_{i+1})$ is $O(b_i)$.* ■

We now determine an upper bound on the cost of read for A as follows. Let τ be the smallest integer $i\ge i^*$ such that $(s_i,t_i)=(0,0)$. By Lemma 5.8, C_τ is a superset of D_τ, implying that a copy of A is located within τ forwarding steps along $\langle x\rangle$. By the definition of the primary and secondary neighbors, the cost of any request (resp., forward) message sent by node x_i is at most $d\cdot c(x_i,x_{i+1})$ (resp., $c(x_i,x_{i+1})$). Since a copy of A is located within τ forwarding steps, by the definition of the algorithm, the cost of all messages needed in locating the particular copy of A that is read is at most $O(\sum_{0\le j<\tau}(d^2c(x_j,x_{j+1})+c(y_j,y_{j+1})))$. The cost of reading the copy is at most $f(\ell(A))$ times the preceding cost. Since d is a constant, the cost of reading A is at most:

$$\sum_{0\le j<\tau} O(f(\ell(A))(c(x_j,x_{j+1})+c(y_j,y_{j+1}))) \qquad (2)$$

The remainder of the proof concerns the task of showing that $E[\sum_{0\le j<\tau}(c(x_j,x_{j+1})+c(y_j,y_{j+1}))]$ is $O(c(x,y))$. A key idea is to establish that the sequence $\langle s_i,t_i\rangle$ corresponds to a two-dimensional random walk that is biased towards $(0,0)$. Lemmas 5.11 and 5.12 provide the important first step towards formalizing this notion.

Lemma 5.11 *Let i be in $[(\log n)/b-1]$. Given arbitrary well defined values for s_j and t_j for all j in $[i]$ such that s_{i-1} is at least 3, the probability that s_i is less than s_{i-1} is at least $1-\varepsilon^2$. Given arbitrary values for s_j and t_j for all j in $[i]$ such that t_{i-1} is at least 1, the probability that t_i is less than t_{i-1} is at least $1-\varepsilon^2$.* ■

Lemma 5.12 *Let i be in $[(\log n)/b-1]$. Given arbitrary well defined values for s_j and t_j for all j in $[i]$ such that s_{i-1} is at most 3, the probability that s_i is 0 is at least $1-\varepsilon$. Given arbitrary values for s_j and t_j for all j in $[i]$ such that t_{i-1} is at most 1, the probability that t_i is 0 is at least $1-\varepsilon$.* ■

By the definitions of s_i and t_i, it follows that $0\le s_{i+1}\le 3$ if $s_i\le 2$, and $0\le s_{i+1}\le s_i+1$ otherwise. In addition, $0\le t_{i+1}\le t_i+1$, for all i. Let s'_i equal 0 if $s_i=0$, equal 1 if $s_i\in\{1,2,3\}$, and equal s_i-2 otherwise. Hence $0\le\max\{s'_{i+1},t_{i+1}\}\le\max\{s'_i,t_i\}+1$, for all i. We now analyze the random walk corresponding to the sequence $\langle\max\{s',t\}\rangle$.

Random Walks. Let $W(U,F)$ be a directed graph in which U is the set of nodes and F is the set of edges. For all u in U, let $\mathcal{D}_u$ be a probability distribution over the set $\{(u,v)\in F\}$ (let $\Pr_{\mathcal{D}_u}[(u,v):(u,v)\notin F]=0$, for convenience). A *random walk* on W starting at v_0 and according to $\{\mathcal{D}_u : u\in U\}$ is a random sequence $\langle v\rangle$ such that: (i) v_i is in U and (v_i,v_{i+1}) is in F, for all i, and (ii) given any fixed (not necessarily simple) path $u_0,\ldots,u_i$ in W and any fixed u_{i+1} in U, $\Pr[v_{i+1}=u_{i+1}\mid(v_0,\ldots,v_i)=(u_0,\ldots,u_i)]=\Pr[v_{i+1}=u_{i+1}\mid v_i=u_i]=\Pr_{\mathcal{D}_{u_i}}[(u_i,u_{i+1})]$.

Let H be the directed graph with node set $\mathbf{N}$ and edge set $\{(i,j): i\in\mathbf{N}, 0\le j\le i+1\}$. Let H' be the subgraph of H induced by the edges $\{(i+1,i),(i,i+1): i\in\mathbf{N}\}\cup\{(0,0),(1,1)\}$.

Let p and q be reals in $(0,1]$. We now define two random walks, $\omega_{p,q}$ and $\omega'_{p,q}$, on graphs H and H', respectively. The walk $\omega_{p,q}=\langle w\rangle$ is characterized by: (i) $\Pr[w_{i+1}\le j-1\mid w_i=j]\ge p$, for any integer $j>1$, (ii) $\Pr[w_{i+1}=0\mid w_i=j]\ge q$, for j equal 0 or 1, and (iii) $\Pr[w_{i+1}=2\mid w_i=1]\le 1-p$. The walk $\omega'_{p,q}=\langle w'\rangle$ is characterized by: (i) $\Pr[w'_{i+1}=j-1\mid w'_i=j]=p$, for all integer $j>1$, (ii) $\Pr[w'_{i+1}=0\mid w'_i=j]=q$, for j equal 0 or 1, and (iii) $\Pr[w'_{i+1}=2\mid w'_i=1]=1-p$. We note that the sequence $\langle\max\{s',t\}\rangle$ represents the random walk $\omega_{p,q}$ with appropriate values for p and q, as determined by Lemmas 5.11 and 5.12. We analyze random walk $\omega_{p,q}$ by first showing that $\omega_{p,q}$ "dominates" $\omega'_{p,q}$ with respect to the properties of interest. The random walk $\omega'_{p,q}$ is easier to analyze as it is exactly characterized by p and q. Lemmas 5.13 and 5.14 show that the bias of $\omega_{p,q}$ towards 0 is more than that of $\omega'_{p,q}$.

Lemma 5.13 *For all i and k in* $\mathbf{N}$, *for random walks $\omega_{p,q}$ and $\omega'_{p,q}$, we have* $\Pr[w_i \le k] \ge \Pr[w'_i \le k]$. ■

Let $z_i(\omega)$ be the random variable denoting the number of steps taken to reach node 0 starting from node i, for a random walk ω. An *excursion* of length ℓ in a graph W with node set $\mathbf{N}$ is a walk that starts at node 0 and first returns to the start node at time ℓ, for all ℓ in $\mathbf{N}$. For all i such that $w_i = 0$, let $\ell_i(\omega)$ be the random variable that gives the length of the excursion in ω starting at time i. We note that for all i, $\ell_i(\omega)$ equals $z_0(\omega)$.

Lemma 5.14 *For all i and ℓ in* $\mathbf{N}$, *we have* $\Pr[z_i(\omega_{p,q}) \le \ell] \ge \Pr[z_i(\omega'_{p,q}) \le \ell]$. ■

We now formalize the notion of the domination of $\omega_{p,q}$ over $\omega'_{p,q}$. For any i, let τ_i (resp., τ'_i) denote the smallest $j \ge 0$ such that $w_{i+j} = 0$ (resp., $w'_{i+j} = 0$). We note that by letting $\langle w \rangle$ represent $\langle \max\{s', t\} \rangle$, the terminating step τ is given by $i^* + \tau_{i^*}$.

Lemma 5.15 *For any i and $j \ge i$, we have* $\Pr[\tau_i \le j] \ge \Pr[\tau'_i \le j]$. ■

The following claim is proved using Raney's lemma [6, 15].

Lemma 5.16 *For all i and ℓ in* $\mathbf{N}$, *we have* $\Pr[\ell_i(\omega'_{p,q}) = \ell + 1 \mid w'_i = 0] \le \max\{1 - q, 5(p - q)\} \Pr[\ell_i(\omega'_{p,q}) = \ell \mid w'_i = 0]$. ■

We now let ω and ω' denote the random walks $\omega_{p,q}$ and $\omega'_{p,q}$, respectively, where $p = 1 - 2\varepsilon^2$ and $q = 1 - 2\varepsilon$. Lemmas 5.12 and 5.11 imply that ω characterizes the random walk corresponding to the sequence $\langle \max\{s', t\} \rangle$. Consider the random walk ω'. Assume that at each step we only reveal whether $w'_i = 0$ or not. We can define a sequence $\langle v \rangle$ associated with $\langle w' \rangle$ as follows: $v_j = G$ iff $w'_j = 0$, and $v_j = B$ otherwise.

Lemma 5.17 *Let i be in $[(\log n/b) - 1]$. Given any fixed sequence $\langle v \rangle_{i-1}$ of B, G values, the probability that w'_i is 0 is at least $1 - 10\varepsilon$.* ■

Our main claim about the random walk ω follows from Lemmas 5.15 and 5.17.

Lemma 5.18 *For any i in $[(\log n)/b - 1]$ and any non-negative integer j, the probability that τ_i is at least j is at most $(10\varepsilon)^j$.*

Using Lemma 5.18, we derive an upper bound on $E[c(x_i, x_{i+1})]$ and $E[c(y_i, y_{i+1})]$ for all i.

Lemma 5.19 *For any i in $[(\log n)/b - 1]$, $E[c(x_i, x_{i+1})]$ and $E[c(y_i, y_{i+1})]$ are both $O(a_i)$.* ■

We now use Lemmas 5.9, 5.18, and 5.19 to establish Theorem 1.

Proof of Theorem 1: By Equation 2, the expected cost of the read operation is bounded by the expected value of $f(\ell(A)) \sum_{0 \le i < \tau} O(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))$. We separately place bounds on $E[\sum_{0 \le i < i^*}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$ and $E[\sum_{i^* \le i < \tau}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$. By Lemmas 5.9 and 5.19, the first term is $O(a_{i^*} + b_{i^*})$.

We place a bound on $E[\sum_{i^* \le i < \tau}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$ as follows. Since τ is $i^* + \tau_{i^*}$, by Lemma 5.18, we obtain that for any $j \ge 0$, the probability that $\tau \ge i^* + j$ is at most $(10\varepsilon)^j$. Therefore, $E[\sum_{i^* \le i < \tau}(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$ is at most:

$$
\begin{aligned}
& \sum_{j \ge 0} j(10\varepsilon)^j (a_{i^*+j} + b_{i^*+j}) \\
\le \quad & \sum_{j \ge 0} j(10\varepsilon)^j 2^{jb \log_\delta 2}(a_{i^*} + b_{i^*}) \\
= \quad & O(a_{i^*} + b_{i^*}),
\end{aligned}
$$

since $10\varepsilon 2^{b \log_\delta 2} < 1$. By Lemma 5.9, the claim of the theorem follows. ■

Proof of Theorem 2: Consider an insert operation executed by x for any object. The expected cost of the operation is bounded by $E[\sum_{0 \le i < \log n/b} c(x_i, x_{i+1})]$, which by Lemmas 5.9 and 5.19 is $O(a_{(\log n)/b-1}) = O(C)$.

We now consider the cost of the delete operation. By Lemma 5.6, for each i, the number of reverse (i,j)-neighbors of x_i for any j is $O(\log n)$ whp, where x_i is the ith node in the primary neighbor sequence of x. Therefore, the expected cost of the delete operation executed by x is bounded by the product of $E[\sum_{0 \le i < \log n/b} c(x_i, x_{i+1})]$ and $O(\log n)$. By Lemma 5.19, it follows that the expected cost of a delete operation is $O(C \log n)$. ■

5.4 Auxiliary Memory

Proof of Theorem 3: We first place an upper bound on the size of the neighbor table of any u in V. By definition, the number of primary and secondary neighbors of u is at most $(d+1)2^b(\log n)/b$, which is $O(\log n)$. By Corollary 5.6.1, the number of reverse neighbors of u is $O(\log^2 n)$ whp.

We next place an upper bound on the size of the pointer list of any u in V. The size of $Ptr(u)$ is at most the number of triples of the form $(A, v, \cdot)$, where A is in

$\mathcal{A}$ and v is in V such that: (i) there exists i in $[(\log n)/b]$ such that v is an i-leaf of u, (ii) $A[j] = u[j]$ for all j in $[i]$, and (iii) A is in the main memory of v.

By Lemma 5.7, the number of i-leaves of u is $O(2^{ib} \log n)$ whp. The probability that $A[j] = u[j]$, for all j in $[i]$, is at most $1/2^{ib}$. Since the number of objects in the main memory of any node is at most ℓ, it follows that whp, $|Ptr(u)|$ is at most $\sum_{i \in [\log n/b]} O(\ell \log n)$ which is $O(\ell \log^2 n)$.

Combining the bounds on the sizes of the neighbor table and pointer list, we obtain that the size of the auxiliary memory of u is $O(\ell \log^2 n)$ whp. ■

5.5 Adaptability

Proof of Theorem 4: By Lemma 5.6, for any node u, the number of nodes of which u is a primary or secondary neighbor is $O(\log n)$ expected and $O(\log^2 n)$ whp. Moreover, u is a reverse neighbor of $O(\log n)$ nodes since u has $O(\log n)$ primary neighbors. Therefore, the adaptability of our scheme is $O(\log n)$ expected and $O(\log^2 n)$ whp. ■

6 Future Work

We would like to extend our study to more general classes of cost functions and determine tradeoffs among the various complexity measures. It would also be interesting to consider models that allow faults in the network. We believe that our access scheme can be extended to perform well in the presence of faults, as the distribution of control information in our scheme is balanced among the nodes of the network.

Acknowledgments

The authors would like to thank Madhukar Korupolu and Satish Rao for several helpful discussions.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed paging for general networks. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 574–583, January 1996.

[2] B. Awerbuch and D. Peleg. Routing with polynomial communication space tradeoff. *SIAM Journal on Discrete Mathematics*, 5:151–162, 1990.

[3] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[4] 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 39–47, May 1992.

[5] S. Dolev, E. Kranakis, D. Krizanc, and D. Peleg. Bubbles: Adaptative routing scheme for high-speed dynamic networks. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 528–537, 1995.

[6] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1989.

[7] J. D. Guyton and M. F. Schwartz. Locating nearby copies of replicated Internet servers. In *Proceedings of ACM SIGCOMM*, pages 288–298, 1995.

[8] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 654–663, May 1997.

[9] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 318–326, May 1992.

[10] S. J. Mullender, editor. *Distributed Systems*. Addison-Wesley, 1993.

[11] S. J. Mullender and P. M. B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[12] C. G. Plaxton and R. Rajaraman. Fast fault-tolerant concurrent access to shared objects. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 570–579, October 1996.

[13] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. Technical Report TR–97–11, Department of Computer Science, University of Texas at Austin, April 1997.

[14] A. G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[15] G. N. Raney. Functional composition patterns and power series reversion. *Transactions American Mathematical Society*, 94:441–451, 1960.

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

[17] M. Van Steen, F. J. Hauck, and A. S. Tanenbaum. A model for worldwide tracking of distributed objects. In *Proceedings of TINA'96*, pages 203–212, September 1996.