Fast Fault-Tolerant Concurrent Access to Shared Objects*

C. Greg Plaxton   Rajmohan Rajaraman

Abstract

We consider a synchronous model of distributed computation in which n nodes communicate via point-to-point messages, subject to the following constraints: (i) in a single "step", a node can only send or receive $O(\log n)$ words, and (ii) communication is unreliable in that a constant fraction of all messages are lost at each step due to node and/or link failures. We design and analyze a simple local protocol for providing fast concurrent access to shared objects in this faulty network environment. In our protocol, clients use a hashing-based method to access shared objects. When a large number of clients attempt to read a given object at the same time, the object is rapidly replicated to an appropriate number of servers. Once the necessary level of replication has been achieved, each remaining request for the object is serviced within $O(1)$ expected steps. Our protocol has practical potential for supporting high levels of concurrency in distributed file systems over wide area networks.

1 Introduction

A basic problem in distributed memory environments is to provide efficient access to remote objects (e.g., files, words of memory). This is a complicated problem because of the large number of (often competing) considerations involved, including: object size, network topology, latency, throughput, buffer space, degree of concurrency to be supported, consistency requirements, and communication overhead.

This paper describes a hashing-based protocol for supporting fast fault-tolerant concurrent access to shared objects in a distributed network. The protocol is most suitable for applications in which: (i) reads occur much more frequently than writes, (ii) objects are not too small, and (iii) the "popularity" of any object (i.e., the number of users trying to access the object) does not constantly change by a significant amount. For example, the protocol might be appropriate for managing access to WWW pages on the Internet, since pages tend to be read far more often than they are written, the typical page size is thousands of bytes or more, and popular pages tend to remain popular over extended periods of time (e.g., for minutes, hours, or even days). In contrast, the protocol would probably be poorly-suited for use within a PRAM emulation scheme, where writes often account for a significant fraction of all accesses, the objects being accessed tend to be extremely small, and the popularity of an object may fluctuate arbitrarily.

Most of the details of our protocol are concerned with ensuring fast access to popular objects. A variety of other well-known methods have been used for solving this problem, including broadcast, combining [14], and multicast [8]. However, the class of architectures that support the efficient implementation of these methods is restricted. For example, a single-bus network can efficiently support broadcast, which enables an arbitrary subset of the processors to obtain copies of a single object at the same time. On the other hand, the cost of implementing broadcast in a distributed network with point-to-point connections is significant.

Modern distributed networks tend to have complex topologies that can support many independent point-to-point connections simultaneously. In fact, it has been widely observed that the communication capabilities of modern distributed networks are similar to those previously associated only with tightly-coupled parallel machines. For this reason, we have chosen to implement and evaluate the performance of our protocol under a model of computation that may be loosely viewed as a variant of Valiant's bulk-synchronous parallel (BSP) model [17]. As in the BSP model, we assume the existence of a simple point-to-point router with no built-in combining or multicast capability. (See [17] for a detailed justification of this assumption.) In order to demonstrate the fault-tolerance of our protocol, our model of computation incorporates both static and dynamic node faults as well as a notion of faulty communication. Most importantly, our fault model assumes that at any given time: (i) a constant fraction of the nodes may be "down" (i.e., unable to communicate with any other nodes), and (ii) each "up" node may be unable to directly communicate (i.e., via a single point-to-point message) with a constant

*Department of Computer Science, University of Texas at Austin, Austin, TX 78712. This research was supported by the National Science Foundation under Grant No. CCR–9504145, and the Texas Advanced Research Program under Grant No. 93–003658–461. E-mail: {plaxton,rraj}@cs.utexas.edu.

fraction of the other "up" nodes. See Section 2 for a precise definition of our model of computation.

How can we provide efficient concurrent access to a given popular object A in a network that supports only partially reliable point-to-point communication? In a conventional distributed file system, a single "server" process (residing on a particular physical node) is assigned the responsibility for storing the object A, and any "client" process wishing to read A sends a message to this server; the server then responds with a message containing a copy of the object A. Unfortunately, this scheme suffers from both low fault-tolerance (if a given client cannot connect to the server due to a network fault, an event that occurs with constant probability in our model, then that client cannot access the object) and high latency (since A is assumed to be a popular object, a long time is needed for the server to sequentially service each of the incoming requests for A).

Thus, to obtain either fault-tolerance or fast concurrent access we are led to consider schemes in which each object is replicated across a number of different nodes. In fact, fault-tolerance considerations alone would seem to imply that each object should be replicated $\Omega(\log n)$ times if we wish to guarantee access with a failure probability that is polynomially small in n, the number of nodes in the network (since each node can fail with constant probability). Unfortunately, this results in an $\Omega(\log n)$-fold increase in the space needed to store each object. However, the theory of erasure codes provides a convenient method for achieving fault-tolerance while paying only a constant factor space penalty. For example, using Rabin's Information Dispersal Algorithm [13] (IDA), for any $k > m$, a given b-bit string can be encoded as a set of k (b/m)-bit strings of length m, with the property that any m of the (b/m)-bit strings suffice to reconstruct the original b-bit string. Thus, IDA can be used to obtain fault-tolerance with only a constant factor space penalty by setting m to $\Theta(\log n)$ and k to $\Theta(m)$, e.g., $k = 2m$. This powerful technique is used by Aumann et al. [3] as part of an efficient scheme for emulating large-grained PRAM programs on an asynchronous parallel machine. In this paper, we use the same technique to store the "primary" copy of each object.

Of course, the IDA technique alone is not sufficient to guarantee fast (e.g., $O(\log n)$ time) concurrent access to an object that is extremely popular. For example, suppose that the popularity of some object is $\sqrt{n}$, and that IDA has been used to encode the object in $\Theta(\log n)$ "fragments", each of which is stored in a separate node. Assuming point-to-point communication, and assuming that a single node cannot send or receive more than a small number (e.g., $O(\log n)$) of messages in a single time step, it is clear that further replication (either of the individual fragments, or of the object as a whole) is needed in order to rapidly service all $\sqrt{n}$ requests. In our scheme, we choose to replicate whole copies of popular objects, as opposed to fragments, so that the encode-decode overhead associated with IDA can be avoided on retrieval of popular objects. (This may be viewed as a minor optimization since the overhead of IDA is actually quite small [13].)

At a high level, our protocol achieves fast concurrent access by enforcing the following two invariants. *Invariant 1:* While the popularity of a given object exceeds the number of "server copies" (i.e., the number of server processes holding a copy of the object), the number of server copies increases geometrically. *Invariant 2:* When the popularity of a given object does not exceed the number of server copies by more than a constant factor, each outstanding request is independently serviced with constant probability at the current step. (Thus, if the popularity of the object does not increase dramatically during subsequent steps, each of the outstanding requests is serviced in $O(1)$ expected steps, and in $O(\log n)$ steps whp[1].)

The methods used to establish these invariants, discussed in Section 3, are loosely related to Valiant's hashing-based combining mechanism for parallel computers [16]. However, the focus of our algorithm design and analysis is different, since Valiant is primarily concerned with the problem of CRCW PRAM emulation, while we have optimized our protocol to obtain fast performance (e.g., expected $O(1)$ time) on a more restricted class of access patterns. See Section 4 for a formal statement of our main results.

We remark that existing implementations for handling concurrent access to shared objects (e.g., [2, 5, 9]) do not provide fast concurrent access in the sense considered in this paper. While these schemes incorporate replication of objects, the only way for a client to determine where the copies of a given object are stored is to consult the "manager" of the object. The manager is usually implemented as a process running at a single node and thus constitutes a sequential bottleneck.

2 Model of Computation

In this section we define our model of computation. We assume a synchronous network consisting of n nodes, each with its own local memory. We specify the model by characterizing: (i) communication, (ii) faults, (iii) object size, (iv) cache size, and (v) local computation.

[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."

Communication. Nodes communicate with one another by sending messages. Each message contains at least one word, and at most $O(\log n)$ words, where a "word" is defined as an $O(\log n)$-bit string.

Sending messages. The total number of words in all messages sent by a single node in one step is required to be $O(\log n)$ (even if some or all of these messages are not successfully transmitted due to faults in the network, which are discussed below).

Receiving messages. In a BSP-like model [17], where communication is assumed to take the form of an h-relation, we might now tend to add a requirement that the total number of words in all messages destined to a single node in one step must be $O(\log n)$. We make a slight variant of this assumption which will ultimately allow us to obtain a more efficient protocol. Namely, we place no upper bound on the total number of words in all messages destined to a single node in one step; instead, we only limit to $c_0 \log n$ the number of words in all messages successfully received by a node in one step, where c_0 is some positive constant. As in the c-arbitrary crossbar model of MacKenzie et al. [11] (where all messages have unit length), we assume that a worst-case adversary determines which subset of the messages of total size $c_0 \log n$ are successfully received by a given node if the $c_0 \log n$ limit on total size would otherwise be exceeded. (The related c-arbitrary DMM model of Karp et al. [10] does not take into account contention among clients trying to access the same object and hence is not well-suited for our study.)

Message types. Our protocol makes use of a constant number of different types of messages. At times the protocol may result in, say, $O(\log n)$ messages of type α and $\Theta(\sqrt{n})$ messages of type β being sent to a particular node. In such a scenario, the adversary referred to above has the freedom to decide that none of the messages of type α get through. On the other hand, it may be important for the correctness of the protocol that the type α messages be given priority over the type β messages. One way to accomplish this is to modify the model stated above by associating a numeric priority with each message-type to resolve contention among messages of different types. Since our protocol only makes use of a constant number of different message types, we could avoid introducing such priorities by modifying the protocol to ensure that only one type of message is ever sent in a single step. We prefer the former solution since it is more compatible with an asynchronous view of the protocol.

Faults. As mentioned in Section 1, our model of computation also allows for the possibility of faults in the network. More specifically, we assume that the network is subject to following three classes of faults.

Random static node faults. After we have fixed our initial storage layout for the objects, we assume that a (sufficiently small) constant fraction ϕ_0 of the nodes are selected at random and marked as "dead". Such dead nodes cannot send or receive any messages throughout the course of the computation.

Random dynamic node faults. In each step of the computation, we assume that a (sufficiently small) constant fraction ϕ_1 of the non-dead nodes are selected at random and marked as "down". Such down nodes cannot send or receive any messages in the current step.

Dynamic link faults. For each pair of non-dead nodes i and j in the network, an oblivious adversary determines whether communication between nodes i and j is to be allowed in step t. In each step t, each non-dead node must be allowed the possibility of communicating with a (sufficiently large) constant fraction $(1 - \phi_2)$ of the other non-dead nodes.

With regard to the dynamic node faults, we should emphasize that the vectors determined by the adversary are not provided to the non-dead nodes at execution time. The only way that a non-dead node can find out whether it is possible to communicate with some other non-dead node in step t is by attempting to send a message in step t, with the hope of subsequently receiving some form of acknowledgment in a later step. (Of course, any acknowledgment message is itself subject to possible faults.)

Object size. Each object consists of $\Theta(\log n)$ words. Note that this assumption can be enforced by simply breaking up larger data items into $\Theta(\log n)$-word pieces, and padding out smaller data items to $\Theta(\log n)$ words. The main reason for assuming a uniform object length is that it simplifies our presentation and analysis. In a practical implementation, we would modify the protocol to handle messages of varying lengths; for larger objects, the associated optimizations can be expected to provide substantial constant factor savings in overhead per object-word accessed.

Cache size. We assume that each node of the network has a cache in which extra copies of objects are stored. We let Δ denote the cache capacity (in objects). In our analysis, it is convenient to assume that $\Delta = \Omega(\log n)$.

Local computation. In each step, a node is allowed to perform an arbitrary amount of local computation. (Although the model of computation allows an arbitrary amount of local computation in each step, our protocol does not perform any particularly complex local operations in a single step.)

3 Overview of the Protocol

In this section we provide an informal overview of our protocol for sharing read-only objects. Our discussion is formalized in Section 5. See Section 7 for a discussion on write operations. As mentioned in Section 1, our protocol relies on maintaining Invariants 1 and 2.

Enforcing Invariant 1. With each object we associate a number of disjoint *blocks* of servers. The ith block contains $\Theta(2^i \log n)$ servers, $0 \leq i < \log(n/\Theta(\log n))$, so that the total number of servers in all blocks is approximately n, the number of nodes in the network. A hash function is used to map these logical blocks of servers to the physical nodes of the network. The hash function is distributed to all nodes so that any node can rapidly compute the physical node corresponding to the jth server of the ith block of a given object. The $\Theta(\log n)$ servers of block 0 of an object are used to store the primary copy of that object, i.e., the $\Theta(\log n)$ fragments computed using IDA. In our protocol, a client process attempting to read a particular object A sends $\Theta(\log n)$ messages, one to each of the $\Theta(\log n)$ servers in block 0 of A, and $O(1)$ messages to a randomly chosen set of servers in each of the $\Theta(\log n)$ other blocks associated with A. If the popularity of A is low (i.e., $O(\log n)$ where the hidden constant is sufficiently small), then whp a sufficiently high constant fraction of the messages sent to block 0 are successfully transmitted, and at the next step a sufficiently high number of fragments are returned to the client, allowing the client to reconstruct a copy of the desired object using IDA. (Note that a node can send $O(\log n)$ copies of a fragment in a single step, since a fragment only consists of a constant number of words.)

If the popularity of A is high (i.e., $\Omega(\log n)$ where the hidden constant is sufficiently high), then so many clients attempt to access A that the servers in block 0 of A are "flooded" with incoming messages requesting fragments of A. As a result, most of these messages are not successfully transmitted, and few if any of the clients receive (on the next step) sufficiently many fragments to reconstruct A using IDA. On the other hand, a sufficiently high constant fraction of the servers in block 0 of A receive $\Theta(\log n)$ messages requesting a fragment of A.

One might believe that *all* of the servers in block 0 receive $\Theta(\log n)$ such messages; this is not necessarily the case, however, since some of these servers may be mapped to the same node as, for example, the servers in block 0 of one or more other popular objects, so that the messages associated with A might be "swamped out" by the messages associated with other objects. A critical part of our analysis is geared towards proving that whp a sufficiently high constant fraction of the nodes in block 0 of A is not the destination of more than a total of $O(\log n)$ messages associated with other objects at the current step; these are the nodes that whp receive $\Theta(\log n)$ requests for A.

Each server in block 0 of A that detects a high level of popularity for A at a particular step reacts by attempting to send a copy of the fragment of A that it holds to all $O(\log n)$ servers in block 1 of A. Although the servers in block 1 may all be flooded with client requests for A (since the popularity of A is assumed to be high), the fragment messages sent from servers in block 0 are not swamped out by such client requests because the fragment messages are given a higher priority. (Of course, we need to argue that these fragment messages are not swamped out by same-priority fragment messages associated with other objects; this follows by essentially the same argument as was mentioned in the preceding paragraph.) As a result of the fragment messages sent from servers in block 0 (the constant fraction detecting a high popularity for A) to servers in block 1, whp a sufficiently high constant fraction of the servers in block 1 of A reconstruct a copy of A using IDA.

Thus, if the popularity of A is sufficiently high at time t, then at time $t+1$, a constant fraction of the servers in block 1 of A hold a copy of A whp. A minor variant of the above process is used to ensure that, if a sufficiently large constant fraction of the servers in block 1 hold a copy of A at time t, and if the popularity of A is $\Omega(\log n)$, then a constant fraction of the servers in block 2 hold a copy of A at time $t+1$. The idea is that a server in block 1 "detects a high popularity" for A if it receives more than a certain constant threshold number of requests for A. Rather than sending $O(\log n)$ fragments of A to servers of block 2 (as were sent from servers of block 0 to servers of block 1 earlier), each server of block 1 detecting a high popularity for A sends $O(1)$ copies of A to a randomly chosen set of servers in block 2 of A. (Note that $O(1)$ copies of A require $O(\log n)$ words.)

More generally, suppose that at time t a sufficiently high constant fraction of the servers in each of blocks 1 through i holds a copy of A, and that the popularity of A is $\Omega(2^i \log n)$, where the hidden constant is sufficiently large. Then a constant fraction of the servers in block i receive more than a certain constant threshold number of requests for A, and react by sending $O(1)$ copies of A to randomly chosen servers in block $i+1$. As a result, at time $t+1$, a constant fraction of the servers in block $i+1$ of A hold a copy of A whp.

Enforcing Invariant 2. The total number of requests received by a server for object fragments is $O(\log n)$ per step, simply because a node cannot receive more than $O(\log n)$ messages per step. Thus, in the following step (assuming it is not subject to a dynamic node fault), a server can respond to each such request with a copy of the desired fragment. (Recall that a fragment consists of a constant number of words and so the total number of words in all of these responses is $O(\log n)$.) Of course, each of these responses may or may not be received by the associated client due to the possibility of dynamic faults in the network. On the other hand, the server may also receive as many as $O(\log n)$ requests for entire copies of objects, and since each object consists of $\Theta(\log n)$ words, only a constant number of these requests can be handled in a single step. In our protocol, the server selects an arbitrary constant-size subset of the incoming requests for entire copies of objects, and responds only to this selected subset.

Now suppose that the hypothesis of Invariant 2 holds, that is, the popularity of some object A is less than or equal to the number of server copies of the object. Because our mechanism for generating server copies fills in the blocks in ascending order of index, we can deduce that a block of servers of A with size within a constant fraction of the current popularity of A satisfies the following two conditions whp: (i) a constant fraction of the servers in the block contain a copy of A, and (ii) each client requesting a copy of A sends a constant number of messages to randomly chosen servers within the block. By a straightforward Chernoff-type argument [6], we can show that a constant fraction of the client requests for A are satisfied at the current step, establishing Invariant 2.

Cache management. In the preceding discussion, we have described the mechanism by which additional server copies of an object are created, but we have not said anything about how such server copies are destroyed. Recall that each node has a cache for holding extra object copies. When this cache becomes full, an LRU (least-recently-used) replacement policy is invoked to decide which object copy to abandon.

4 Results

In this section, we state our main performance bounds for the read-only protocol, which is formally defined in Section 5. We analyze the protocol under different access pattern models. Our time bounds are stated in terms of *rounds*; for any nonnegative integer t, round t consists of steps $2t$ and $2t+1$ (steps are numbered from 0). Let $\mathcal{A}$ be a collection of m objects, labeled A_0 through A_{m-1}. For any round t and any integer i in $[m]$, let $q_i(t)$, $r_i(t)$, and $s_i(t)$ denote the number of new requests generated, the number of requests remaining, and the number of requests attempted, respectively, for A_i at the start of round t. (For any nonnegative integer x, we use $[x]$ to denote the set $\{0,\ldots,x-1\}$.) Thus, for any round t and i in $[m]$, $s_i(t) = r_i(t) + q_i(t)$.

We measure the performance of our protocol in terms of *throughput*, *delay*, and *per-request communication*. The throughput of the protocol is the average number of access requests satisfied per round. The delay of an individual access request is the number of rounds taken to satisfy that request. The per-request communication is defined as the total number of words in all messages sent divided by the number of access requests satisfied. We say that round t is *steady* if: (i) there exists a real constant δ, $0 \le \delta < 1$, such that $r_{i+1}(t) \le \delta s_i(t)$ for all i in $[m]$, and (ii) the probability that an arbitrary access request is satisfied in round t is $\Omega(1)$. Thus, if round t is steady, for every object A, an expected constant fraction of the requests for A are satisfied in round t.

The first access model we consider is the *fixed probability distribution* model, in which each new access request is independently drawn from a fixed probability distribution $\mathcal{D}$ on $\mathcal{A}$. The distribution $\mathcal{D}$ is specified by a vector $(p_0,\ldots,p_{m-1})$; for a random variable X drawn from $\mathcal{D}$, we have $\Pr[X = A_i] = p_i$. At the start of each round t, new requests drawn from $\mathcal{D}$ are generated and "placed" by an adversary on each of the nodes that do not currently have an outstanding request. The particular assignment of new requests to "free" clients can be arbitrarily determined by the adversary.

Theorem 1 *In the fixed probability distribution model, there exists $t_0 = O(\log n)$ such that for any t, $t_0 \le t \le \mathrm{poly}(n)$, round t is steady whp.*

It follows from Theorem 1 and the protocol definitions that, given an arbitrary probability distribution $\mathcal{D}$ on the access requests, our protocol provides optimal throughput and optimal expected delay using optimal per-request communication.

Corollary 1.1 *If each new request is drawn from $\mathcal{D}$, then in t rounds, $\Omega(\log n) \le t \le O(\mathrm{poly}(n))$, whp, the number of access requests satisfied is $\Omega(nt)$ using $O(\log n)$ per-request communication. Moreover, after $O(\log n)$ rounds, each access request is satisfied in expected $O(1)$ rounds.* ■

(Note that in Corollary 1.1 the per-request communication is optimal since each object is of size $\Theta(\log n)$.)

Our analysis for the fixed probability distribution model can be easily extended to apply to a *time-varying probability distribution model*, in which the probability distribution changes every $\Omega(\log n)$ steps. The model is defined as follows. Let $\mathcal{D}_0,\ldots,\mathcal{D}_{d-1}$, be a sequence of d probability distributions over $\mathcal{A}$. We consider ds rounds of the protocol, where s is a nonnegative integer. In round $j \in [ds]$, each new access request is independently drawn from $\mathcal{D}_{\lfloor i/s \rfloor}$. As in the fixed probability distribution model, the particular assignment of new requests to free clients is determined by an adversary.

Theorem 2 *Consider ds rounds in a time-varying probability distribution model with d distributions such that d is $O(\mathrm{poly}(n))$ and s is $\Omega(\log n)$. The number of access requests satisfied by the protocol is $\Omega(nds)$ using $O(\log n)$ per-request communication.* ■

Thus far, we have considered access patterns in which each new request is drawn from a probability distribution. The bounds stated in Corollary 1.1 also hold for access patterns in which the popularity of an object can change arbitrarily, subject to the constraint that for all i in $[m]$ and $t \geq 0$, we have $q_i(t) \leq c \max\{q_i(t') : \max\{0, t - \Theta(\Delta)\} \leq t' < t\}$, where $c > 1$. Note that such an access pattern allows arbitrary decreases in the popularity of an object, and also admits large increases in certain cases.

5 The Read-Only Protocol

In this section, we formally define our protocol for accessing read-only shared objects. With every object A we associate n server processes, which provide client processes access to A. Let the servers associated with A be labeled $S_i(A)$, $0 \leq i < n$. We partition the servers into *blocks* as follows. For i in $[b]$, $b = \lceil \log(n/c_1 \log n) + 1 \rceil$, the ith block, denoted by $B_i(A)$, is the set $\{S_j(A) : (2^i - 1)c_1 \log n \leq j < (2^{i+1} - 1)c_1 \log n\}$.

Each server associated with A is mapped to a physical node by means of a hash function h_A; the function h_A is chosen such that for any i in $[b]$, block $B_i(A)$ is mapped to a subset of $|B_i(A)|$ nodes chosen independently and uniformly at random. (Note that several servers associated with different objects may be mapped to the same node.) We encode A as a set of $c_1 \log n$ *fragments* such that any $c_2 \log n$ fragments suffice to decode A. For each i in $[c_1 \log n]$, server $h_A(S_i(A))$ stores the ith fragment of A. For each j in $[b]$, and for each server $S \in B_j(A)$, $h_A(S)$ stores at most one replicated copy of the entire object. Each node has an LRU cache for storing up to Δ object copies.

We describe our access protocol in terms of the communication between the clients attempting to access a given object A and the servers associated with A. In order to simplify the presentation and analysis of the protocol, we assume that the clients send messages at even steps of the protocol and the servers send messages at odd steps of the protocol. The clients always send messages to servers; servers send messages to both clients and servers.

In our description of the protocol, we differentiate between several kinds of messages; these are listed in Table 1. In the priority-based model, any assignment of priorities that respects the following constraints can be used: (i) *obj-req* has a lower priority than each of *client-obj*, *client-frag*, *server-obj*, and *server-frag*, and (ii) *frag-req* has a lower priority than each of *client-obj* and *client-frag*.

The protocol is defined in Figure 1, where we state the actions in round t of: (i) a client C attempting to access object A, and (ii) a server S associated with A. (Recall that round t consists of steps $2t$ and $2t+1$.) It is convenient to divide each step into two phases, one in which messages are sent, and the other in which messages are received. Thus, in Figure 1, Phase 0 (resp., Phase 2) is the "sending phase" for step $2t$ (resp., $2t+1$), while Phase 1 (resp., Phase 3) is the "receiving phase" for step $2t$ (resp., $2t+1$). In Figure 1, π_0, π_1, π_2, and π_3 denote positive constants.

The terms "send", "receive", and "attempt to send" are used in the protocol definition. When we say that a client/server mapped to node u *sends* a message x, we mean that u initiates the transmission of x. When we say that a client/server mapped to node u *receives* a message x, we mean that the transmission of x is successful and x is at destination u. When we say that a client/server mapped to node u *attempts to send* a message x, we mean that transmission of x is initiated if x is in the subset of messages of total size at most $c_0 \log n$ that is selected for transmission from u.

6 Analysis

Our analysis proceeds in two parts. In the first part (Section 6.1), we define the notion of a good round and show that Invariants 1 and 2 of Section 1 hold whp in a good round. The claims in Section 6.1 hold for an arbitrary access pattern. In the second part (Section 6.2), we restrict our attention to the fixed probability distribution model and prove Therorem 1. Due to space constraints, we omit most of the proofs in this abstract. We refer the reader to the full version of the paper [12] for complete proofs of the results stated in Section 4.

6.1 Protocol Properties

Definition 6.1 *Let X denote a set of balls numbered 0 through $|X| - 1$. Let U denote a set of bins. Let ε_0 and ε_1 be reals such that $0 \leq \varepsilon_0, \varepsilon_1 < 1$, and let τ be a nonnegative integer. We say that Y is the* **outcome of an** $(X, U, \varepsilon_0, \varepsilon_1, \tau)$**-experiment** *if there exists a set $\{V_i : i \in [|X|], V_i \subseteq U$, and $|V_i| \leq \varepsilon_1|U|\}$ such that Y is the random variable denoting the set of remaining balls after the following experiment: (i) discard an arbitrary subset of at most $\varepsilon_1|X|$ balls, (ii) throw the remaining balls uniformly at random into U, (iii) for each $i \in X$, if ball i lands in a bin in V_i, discard ball i, (iv) for each bin that has more than τ balls, discard all but an arbitrary subset of τ balls.*

The experiment of Definition 6.1 can be analyzed using standard probabilistic techniques such as Chernoff bounds [6] and Azuma's inequality [1].

For any step t, object A, message-type α, and $i \in [b]$, let $M_\alpha(A, i, t)$ (resp., $M'_\alpha(A, i, t)$) denote the set of messages of type α attempted to (resp., received by) block

Message type	Source	Destination	Size	Contents
obj-req	client	server	$\Theta(1)$	request for object
frag-req	client	server	$\Theta(1)$	request for fragment
client-obj	server	client	$\Theta(\log n)$	copy of object
client-frag	server	client	$\Theta(1)$	copy of fragment
server-obj	server	server	$\Theta(\log n)$	copy of object
server-frag	server	server	$\Theta(1)$	copy of fragment

Table 1: Types of messages.

$B_i(A)$ in step t. (Here and in the rest of this section, we use the word "attempt" as a short form for "attempt to send".) Similarly, let $N_\alpha(A,i,t)$ (resp., $N'_\alpha(A,i,t)$) denote the set of messages of type α attempted (resp., sent) by block $B_i(A)$.

If we assume a fault-free model with no upper bound on the number of words a node can send/receive in a single step, then since each message is sent to a random server, $M_\alpha(A,i,t)$ and $B_i(A)$ correspond to a random throw of $|M_\alpha(A,i,t)|$ balls into $|B_i(A)|$ bins. Unfortunately, in the presence of faults and contention, this is no longer true. On the other hand, we are able to show that the process relating $M_\alpha(A,i,t)$ and $B_i(A)$ is an instance of Definition 6.1.

To simplify the presentation, we assume in this section that there is no contention among messages of distinct types. The message priorities can easily be used to remove this assumption [12]. Let $\mathrm{size}(\alpha)$ denote the number of words in a message of type α.

Definition 6.2 *Step t is* **good** *if for every object A, message-type α, and i in $[b]$, the following conditions hold: (i) $M'_\alpha(A,i,t)$ is the outcome of an $(M_\alpha(A,i,t), B_i, \varepsilon_0, \varepsilon_1, \Theta(\log n)/\mathrm{size}(\alpha))$-experiment for a sufficiently small choice of the constants ε_0 and ε_1, and (ii) $|N'_\alpha(A,i,t)|$ is $\Omega(|N_\alpha(A,i,t)|)$. Round t is* **good** *if steps $2t$ and $2t+1$ are both good.*

Let $M_\alpha(t)$ denote the set of messages of type α that are attempted in step t. For any set of messages X, let $\|X\|$ denote the total number of words in X. The following lemma places an upper bound on $\|M_\alpha(t)\|$. For even steps, the bound follows directly from the protocol definition; for odd steps, it can be proved using Chernoff bounds.

Lemma 6.1 *For any round t and message-type α, if $\sum_{i\in[m]} s_i(t)$ is $O(n)$, then $\|M_\alpha(2t)\|$ and $\|M_\alpha(2t+1)\|$ are $O(n\log n)$.* ■

Lemma 6.2 *If there are nonnegative reals x_i, $0 \le i < m$, such that $s_i(t) \le x_i$ and $\sum_{i\in[m]} x_i = O(n)$, then round t is good whp.*

Proof: We fix indices i and j, $i \in [b]$, $j \in [m]$, and message-type α. For convenience, we drop the subscript α in the rest of the proof.

We first establish part (ii) of Definition 6.2, which concerns odd steps only. Since $\sum_{k\in[m]} s_k(t)$ is $O(n)$, by Lemma 6.1, $\|M(2t+1)\|$ is $O(n\log n)$. Let U be the set of nodes u such that at most $(c_0\log n)/2$ words are attempted from u in step $2t+1$. By an averaging argument, it follows that $|U|$ is $\Omega(n)$, where the hidden constant can be made arbitrarily close to 1 by setting c_0 sufficiently large. By definition, all messages in $N(A_j,i,2t+1)$ are attempted by servers. Given arbitrary $x_0,\ldots,x_{m-1}$ such that $s_k(t) \le x_k$ for all k in $[m]$, the mapping of $B_i(A_j)$ is independent of $M(2t+1) \setminus N(A_j,i,2t+1)$. It follows from bounds on the tail of the hypergeometric distribution [7] that a constant fraction of the messages in $N(A_j,i,2t+1)$ are attempted by nodes in U. Therefore, $|N'(A_j,i,2t+1)|$ is $\Omega(|N(A_j,i,2t+1)|)$ whp.

For part (i) of Definition 6.2, we need to prove that for t' in $\{2t,2t+1\}$, $M'(A_j,i,t')$ is the outcome of an $(M(A_j,i,t'), B_i(A_j), \varepsilon_0, \varepsilon_1, \Theta(\log n)/\mathrm{size}(\alpha))$-experiment. In our proof, the messages in $M(A_j,i,t')$ correspond to balls, and the servers in $B_i(A_j)$ correspond to bins. We will account for parts (i) through (iv) of Definition 6.1 as follows: (1) an analysis of contention at source nodes will account for part (i), (2) the fact that each message is sent to a random server will account for part (ii), (3) an analysis of contention at destination nodes and faults will account for part (iii), and (4) a bound on the contention at destination nodes will account for part (iv).

We first consider point (1). Let X be the set of messages of type α in $M(A_j,i,t')$ that are sent in step t'. We note that α is one of *obj-req, frag-req, server-frag, server-obj*. If α is *obj-req* or *frag-req*, it easily follows from the protocol definition that each message of type α that is attempted is sent. In this case, X equals $M(A_j,i,t')$. If α is in {*server-frag, server-obj*}, then by part (ii) of Definition 6.2 shown above, $|X|$ is at least $c_3|M(A_j,i,t')|$ whp, where c_3 can be made arbitrarily close to 1 by setting c_0 sufficiently large.

Phase 0: *In step $2t$ clients send request messages.*

- Client. Attempt to send a *frag-req* message to each server in $B_0(A)$ and, for $0 \leq i < b$, a *obj-req* message to a random server in $B_i(A)$.

(Remark: Note that each message is actually sent since the bound on the number of words that can be sent by a node is not exceeded.)

Phase 1: *Successfully transmitted Phase 0 messages are received by servers.*

- Server. Let $D(S,t)$ denote the set of clients that are the sources of *obj-req* and *frag-req* Phase 1 messages received by S.

Phase 2: *In step $2t+1$, servers holding a copy or fragment of object A respond to Phase 1 messages. Let $S \in B_i(A)$.*

- Server, $i = 0$. Attempt to send a *client-frag* message to $\min\{\Theta(\log n), |D(S,t)|\}$ clients in $D(S,t)$, and if $D(S,t) = \Omega(\log n)$ then attempt to send a *server-frag* message to $\Theta(\log n)$ random servers in $B_1(A)$.
- Server, $i > 0$. If $|D(S,t)| \geq \pi_0$ then attempt to send a *client-obj* message to $\min\{\pi_1, |D(S,t)|\}$ clients in $D(S,t)$, and if $D(S,t) \geq \pi_2$ then attempt to send a *server-obj* message to π_3 random servers in $B_{i+1}(A)$.

(Remark: If the bound on the number of words a node can send in a step would be exceeded, an arbitrary subset of these messages are actually sent.)

Phase 3: *Successfully transmitted Phase 2 messages are received by clients and servers.*

- Client. If C receives a *client-obj* message or $c_2 \log n$ fragments, then the access attempt is successful. Otherwise, C attempts to access A in round $t+1$.
- Server, $i = 1$. If S receives at least $c_2 \log n$ fragments, then decode A and store it in the cache using LRU policy; otherwise, discard the fragments received.
- Server, $i > 1$. If any *server-obj* message is received, then store A in the cache using LRU policy.

(Remarks: Note that C could receive more than one copy of A, and that S could be sent a *server-obj* message even though S already has a copy. In a practical implementation: (i) C would stop transmission of all but one copy of A, (ii) a check would be added to ensure that a *server-obj* message is sent to S only if S does not already have a copy, and (iii) if fewer than $c_2 \log n$ fragments are received by a client or server, then these fragments would be cached and not discarded since sufficiently many additional fragments are likely to be received in the near future.)

Figure 1: The read-only protocol (object A, client C, server S, round t).

For point (2), it is enough to note that each message is sent to a random server within a block. We now consider point (3). Let U denote the set of up nodes (i.e., neither dead nor down) in round t. Note that $|U|$ is at least $(1-(\phi_0+\phi_1))n$. Consider any message y in $M(A_j,i,t')$. Let V_y be the set of up nodes $u \in U$ such that at most $(c_0 \log n)/2$ words in $M(t') \setminus M(A_j,i,t')$ are destined to u and u has a non-faulty link to the source of message y. Since $\|M(t')\|$ is $O(n \log n)$, by an averaging argument, it follows that $|V_y|$ is $\Omega(n)$, where the hidden constant is arbitrarily close to 1 for c_0 sufficiently large and ϕ_0, ϕ_1, and ϕ_2 sufficiently small. Let W_y be the set of servers in $B_i(A_j)$ that are mapped to nodes in V_y. Since the mapping of servers in $B_i(A_j)$ is independent of $M(t') \setminus M(A_j,i,t')$, it follows from bounds on the tail of the hypergeometric distribution [7] that $|W_y|$ is at least $c_4|B_i(A_j)|$ whp, where c_4 can be set arbitrarily close to 1 for appropriate values of c_0, ϕ_0, ϕ_1, and ϕ_2. For point (4), we use the fact that each server in W_y can receive at least $(c_0 \log n)/2$ words from $M(A_j,i,t')$.

We now establish part (i) of Definition 6.2 by using Definition 6.1 with $M(A_j,i,t')$ substituted for X, $B_i(A_j)$ for U, $M'(A_j,i,t')$ for Y, $(1-c_4)$ for ε_0, and $\Theta(\log n)/\mathrm{size}(\alpha))$ for τ; ε_1 is set to 1 if α is in $\{$*obj-req*, *frag-req*$\}$ and to $1-c_3$ otherwise. For message $y \in M(A_j,i,t')$, $B_i(A_j) \setminus W_y$ plays the role of V_i for ball i. Points (1), (2), (3), and (4) correspond to parts (i), (ii), (iii), and (iv), respectively, of Definition 6.1, and hence the claim holds. ∎

For any server S associated with object A, let $a(S,t)$ be 1 if server S has a copy of A or a fragment of A at the start of round t, and 0 otherwise. For each i in $[b]$, we associate a state variable $s(A,i,t) \in \{\text{complete}, \text{incomplete}\}$ that is defined as follows: if the number of servers S in $B_i(A)$ such that $a(S,t) = 1$ is at least $c_5|B_i(A)|$, then $s(A,i,t)$ is complete; otherwise, $s(A,i,t)$ is incomplete. Let $R(A,i,t)$ denote the set of clients that attempt any *obj-req* or *frag-req* messages to $B_i(A)$ in round t. Let $R'(A,0,t)$ denote the set of clients C that receive at least $c_2 \log n$ distinct *client-frag* messages in round t. For $i > 0$, let $R'(A,i,t)$ denote the set of clients that receive at least one *client-obj* message containing object A in round t. Define predicates P_0 through P_3 as follows:

- $P_0(A,t)$: If $|R(A,0,t)|$ is $\Omega(\log n)$, then for $t+1 \leq t' \leq t+\Theta(\Delta)$, $s(A,1,t)$ is complete.

- $P_1(A, i, t)$: If $|R(A, i, t)|$ is $\Omega(|B_i(A)|)$ and $s(A, i, t)$ is complete, then for $t + 1 \leq t' \leq t + \Theta(\Delta)$, $s(A, i, t')$ is complete.
- $P_2(A, t)$: If $|R(A, 0, t)|$ is $O(\log n)$, then $R'(A, 0, t) = R(A, 0, t)$.
- $P_3(A, i, t)$: If $s(A, i, t)$ is complete and $|R(A, i, t)|$ is $O(|B_i(A)|)$, then: (i) for each $C \in R(A, i, t)$, we have $\Pr[C \in R'(A, i, t)] = \Omega(1)$, and (ii) $|R'(A, i, t)|$ is $\Omega(|R(A, i, t)|)$.

The predicates $P_0(A, t)$ and $P_1(A, i, t)$ (resp., $P_2(A, t)$ and $P_3(A, i, t)$) formalize Invariant 1 (resp., Invariant 2) of Section 1. Lemma 6.3 follows from an analysis of the experiment of Definition 6.1.

Lemma 6.3 *If round t is good, then the following predicates hold for every object A whp: (i) $P_0(A, t)$, (ii) $\forall i > 0 : P_1(A, i, t)$, (iii) $P_2(A, t)$, and (iv) $\forall i > 0 : P_3(A, i, t)$.* ■

6.2 The fixed probability distribution model

We would like to show that, given an arbitrary probability distribution $\mathcal{D}$, our protocol reaches a "steady state" quickly, and that thereafter, in any round, each outstanding request is satisfied with constant probability. The key idea behind this proof is to show that every round is good whp, so that predicates P_0 through P_3 can be used to establish Theorem 1. In fact, if $(s_0(t), \ldots, s_{m-1}(t))$ is fixed, then by Lemma 6.2 each round t is good whp. However, if we consider a sequence of rounds 0 through $t-1$, $(s_0(t'), \ldots, s_{m-1}(t'))$ for any $t' \in [t]$ is a random variable that could depend on our choice of hash functions, and hence Lemma 6.2 is not directly applicable. Although, intuitively, we do not expect this dependence to have a significant impact on the performance of the protocol, it is a significant technical challenge to prove this.

For any round t, we define $d_i(t)$ to be the largest index j such that $s(A_i, j, t)$ is complete. For any i in $[m]$, object A_i is said to be *steady* in round t if $s_i(t)$ is $O(2^{d_i(t)} \log n)$, where the hidden constant is chosen appropriately.

Lemma 6.4 *For any round $t \leq \mathrm{poly}(n)$, we have:*

1. *Round t is good whp.*
2. *Consider object A_i for any i in $[m]$. If A_i is not steady in rounds 0 through t, then $d_i(t+1)$ is $t+1$; otherwise, $s_i(t+1)$ is $O(2^{d_i(t+1)} \log n)$ whp.* ■

The proof of Lemma 6.4 is quite involved; we give a brief overview here. We prove part 1 of Lemma 6.4 by first showing that there exist $O(\log n)$ independent random vectors $(X_{j,0}, \ldots, X_{j,m-1})$ such that: (i) for all j, $\sum_{i \in m} X_{j,i}$ is $O(n)$ whp, and (ii) for any round $t \leq \mathrm{poly}(n)$, there exists some j such that for all $i \in [m]$, $s_i(t) \leq X_{j,i}$ whp. We then invoke Lemma 6.2 to show that round t is good whp for $t \leq \mathrm{poly}(n)$, and hence establish part 1. We prove part 2 by using part 1 and Lemma 6.3.

Proof of Theorem 1: For any i in $[m]$, if A_i is not steady in rounds 0 through $b-2$, then by part 2 of Lemma 6.4, $d_i(b-1)$ is $b-1$. Since $c_1 2^{b-1} \log n > n \geq s_i(b-1)$, for an appropriate choice of constants, A_i is steady in round $b-1$. Thus, for i in $[m]$ and any round $t \leq \mathrm{poly}(n)$, by part 2 of Lemma 6.4, $s_i(t)$ is $O(2^{d_i(t)} \log n)$ whp. For an appropriate choice of constants, predicates P_2 and P_3 imply that $r_i(t+1)$ is at most $\delta s_i(t)$ whp for some constant $\delta < 1$, and the probability that an arbitrary request is satisfied in round t is $\Omega(1)$. Thus round t is steady whp. ■

7 Write Operations

Thus far, we have focused our attention on read-only objects. In this section, we describe our algorithm for handling write operations. We consider two different approaches: the *write-and-update* and the *invalidate-and-write* protocols. We briefly describe both protocols here. We refer the reader to the full version of the paper [12] for details.

At a given time step, any number of clients may attempt to simultaneously initiate a write operation on some object A. The first part of each protocol consists of a simple randomized leader election procedure to select one of these clients to actually write the object A. The selected client then attempts to write the fragments of the new version of object A into the block $B_0(A)$. A time-stamp is sent along with each of these fragments so that future clients reading the fragments can differentiate old fragments from new ones.

The two protocols differ in the second part. In the write-and-update protocol, after the fragments are sent to block $B_0(A)$, updates are propagated to servers in higher-numbered blocks that hold copies of A, by the same method as is used to propagate copies. The write is assumed to "complete" before these updates are propagated. As a result, it is possible that a client reads an old version of an object. We use the following validation scheme to ensure that each client receives a version that is at most $O(\log n)$ steps out of date. A steady stream of validation time-stamps is created by servers in $B_0(A)$ and propagated to higher-numbered blocks that hold copies of A. Each server S that has a copy of A maintains a variable $b(S)$ that denotes the last validation time-stamp received by S. A server S in $B_i(A)$, satisfies a request in round t only if $b(S)$ is at least $t - i$, thus ensuring that the version sent to a client is at most $O(\log n)$ steps old. Since the per-request communication due to the validation time-stamps is asymptotically smaller than that required by the rest of the protocol, the results of Section 4 hold as stated.

In the invalidate-and-write protocol, we maintain, for each object, a fault-tolerant distributed list of all servers and clients holding a copy of the object. When a write operation is performed, before updating the primary copy, the servers in block 0 participate in an invalidation scheme in which each client/server on the list is sent one or more invalidation messages whp. The main advantage of this extension is that clients can make use of locally-cached copies of objects since they are informed once such a copy becomes out of date. The main disadvantage is that it is not possible to guarantee in the worst case that these invalidation messages are all sent quickly (e.g., within $O(\log n)$ steps). The difficulty is that the lists can grow very long over time, and if a large number of write operations are performed over a short period on a set of objects with long associated client/server lists, then it is simply not possible to send all of the invalidation messages quickly. On the positive side, it is possible to prove a good amortized bound on the total number of messages used by the extended protocol for processing a given number of read/write accesses.

8 Concluding Remarks

In order to achieve fault-tolerance and space-efficiency, our protocol uses Rabin's IDA technique to encode each object as a set of fragments such that only a constant fraction of the fragments are needed to reconstruct an object. One shortcoming of IDA is that it does not tolerate errors in the fragments. Suppose, for example, that a client reading an object receives a large number of fragments, each of which is noisy (i.e., contains arbitrarily many errors) with some constant probability $\varepsilon > 0$. Unless the noisy fragments can be easily identified as such, the client cannot efficiently reconstruct the object using IDA. In such a noisy setting, it would be worthwhile to consider variants of our protocol based on the Berlekamp-Welch decoder [4] (see also [15, Appendix A]), which tolerates noise in a constant fraction of the fragments.

We would like to extend our protocols to other interesting models of distributed computation that incorporate asynchrony or locality information. We conjecture that, with suitably modified definitions and appropriate technical assumptions, the performance bounds of the present paper can be extended to apply to models allowing limited forms of asynchrony (e.g., bounded asynchrony). To address the issue of locality, it would be interesting to consider a variant of our protocol in which the number of copies of an object that are created in any region of the network is proportional to its popularity within the region, and where regions are identified on the basis of some hierarchical decomposition of the network.

Acknowledgments

The authors would like to thank Pawan Goyal and David Zuckerman for useful discussions, and Micah Adler, Funda Ergun, and S. Ravikumar for several helpful comments on an earlier draft of this abstract.

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, New York, NY, 1991.

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

[3] Y. Aumann, Z. Kedem, K. V. Palem, and M. O. Rabin. Highly efficient asynchronous execution of large-grained parallel programs. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 271–280, November 1993.

[4] E. Berlekamp and L. Welch. Error correction of algebraic block codes. U.S. Patent Number 4,633,470.

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

[6] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[7] V. Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25:285–287, 1979.

[8] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, pages 85–111, 1990.

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

[10] 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.

[11] P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman. On contention resolution protocols and associated probabilistic phenomena. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 153–162, May 1994.

[12] C. G. Plaxton and R. Rajaraman. Fast fault-tolerant concurrent access to shared objects. Technical Report TR–96–21, Department of Computer Science, University of Texas at Austin, September 1996.

[13] M. O Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *JACM*, 36:335–348, 1989.

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

[15] M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. PhD thesis, Department of Computer Science, University of California at Berkeley, October 1992.

[16] L. Valiant. A combining mechanism for parallel computers. Technical Report TR-24-92, Center for Research in Computing Technology, Harvard University, January 1992.

[17] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, 1990.