The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data

D. AGRAWAL
and
A. EL ABBADI
University of California

In this paper, we present a low-cost fault-tolerant protocol for managing replicated data. We impose a logical tree structure on the set of copies of an object and develop a protocol that uses the information available in the logical structure to reduce the communication requirements for read and write operations. The tree quorum protocol is a generalization of the static voting protocol with two degrees of freedom for choosing quorums. In general, this results in significantly lower communication costs for comparable data availability. The protocol exhibits the property of graceful degradation, i.e., communication costs for executing operations are minimal in a failure-free environment but may increase as failures occur. This approach in designing distributed systems is desirable since it provides fault-tolerance without imposing unnecessary costs on the failure-free mode of operations.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed databases*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems, trees*

General Terms: Algorithms, Economics, Experimentation, Performance, Measurement

1. INTRODUCTION

In a distributed database system, data is replicated to achieve fault-tolerance. One of the most important advantages of replication is that it masks and tolerates failures in the network gracefully. In particular, the system remains operational and available to the users despite failures. However, complex and expensive synchronization protocoles [6, 7, 9, 14, 18, 23] are needed to maintain the replicas. In general, the cost of executing operations in replicated databases increases, since multiple copies of an object may need

This research was supported by the NSF under grants CCR-8809387, IRI-8809284, and IRI-9004998.
Authors' address: Department of Computer Science, University of California, Santa Barbara, CA 93106.

to be accessed for maintaining the mutual consistency of the copies. There has been a considerable research effort to reduce the cost of executing operations while maintaining data availability in replicated databases. A common approach is to use the logical view of the network configuration [10, 11, 17]. This information is used to allow operations to adapt to changes in the network configuration.

In this paper, we present a protocol for managing replicated data that reduces the cost of executing operations without the need for reconfiguration. This is achieved by imposing a logical tree structure on the set of copies of each object. Information available in the logical structure reduces the communication requirements for read and write operations. We refer to the generic protocol that uses the tree structure as the *tree quorum protocol*. The tree quorum protocol is a generalization of the static voting protocol with two degrees of freedom for choosing quorums instead of one. In general, this results in significantly lower communication costs for comparable data availability.

We analyze three specific instances of the tree quorum protocol. First, we develop the *majority tree* protocol which has comparable read and write availability to the quorum protocol [14, 27] but at significantly lower costs. By varying the read and write quorums, we develop the *read root* protocol which executes read operations by reading one copy of an object in a failure-free environment while guaranteeing fault-tolerance of write operations. Finally, the *log write* protocol is described which has very low cost for write operations and yet provides high read and write availability. Furthermore our approach is fault-tolerant, and exhibits the property of graceful degradation [22]. In a failure-free environment, the communication costs are minimal and as failures occur the costs may increase. However, when failures are repaired the protocol reverts to its original mode without undergoing any reconfiguration.

In the next section, we present the model of a distributed replicated database. Section 3 motivates the usefulness of logical structures for managing replicated data. The tree quorum protocol, which incorporates these ideas, is presented in Section 4. Analysis of the majority tree protocol and its comparison with other protocols are presented in Section 5. The read root and log write protocols are also analyzed in the same section. The protocol is extended for incomplete trees in Section 6. In Section 7, we relate the proposed tree quorum protocol to other protocols for managing replicated data. We conclude with a discussion of our results.

2. MODEL

A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network. We assume that sites are *fail-stop* [25] and communication links may fail to deliver messages. Combinations of such failures may lead to *partition-*

ing failures [8], where sites in a *partition* may communicate with each other, but no communication can occur between sites in different partitions. A site may become *inaccessible* due to site or partitioning failures.

A *distributed database* consists of a set of *objects* stored at several sites in a computer network. Users interact with the database by invoking *transactions*. A transaction is a partially ordered sequence of read and write operations that are executed atomically. The execution of a transaction must appear atomic, i.e., a transaction either *commits* or *aborts* [15]. A commonly accepted correctness criteria in databases is the *serializable* execution of transactions [12]. The serializable execution is guaranteed by employing a *concurrency control* mechanism, e.g., locking [12, 26, 3], timestamp ordering [24], or optimistic concurrency control [20] protocols.

In a replicated database, copies of an object may be stored at several sites in the network. Multiple copies of an object must appear as a single logical object to the transactions. This is termed as *one-copy equivalence* [6] and is enforced by a *replica control* protocol. The correctness criteria for replicated databases is *one-copy serializability* [6], which ensures both one-copy equivalence and the serializable execution of transactions.

In order to ensure one-copy equivalence, a replicated object x may be read by reading a *read quorum* of copies, and it may be written by writing a *write quorum* of copies. The following restriction is placed on the choice of quorum assignments:

Quorum intersection property: For any two operations $o[x]$ and $o'[x]$ on an object x, where at least one of them is a write, the quorums must have a nonempty intersection.

Version numbers or timestamps are used to identify the current copy in a quorum. When timestamps are used, intersection of write quorums is not necessary [16]. The notion of intersecting quorums is closely related to *coteries* [13]. In coteries, an added minimality condition is imposed upon quorums while with intersecting quorums a distinction is made between read and write operations.

3. MOTIVATION

The simplest example of a protocol for managing replicated data is one where read operations are allowed to read any copy, and write operations are required to write all copies of the object. The read-one write-all protocol provides read operations with a high degree of availability at a very low cost: a read operation accesses a single copy. On the other hand, this protocol severely restricts the availability of write operations since they cannot be executed after the failure of any copy.

Two main approaches have been used to address the issue of increasing the fault-tolerance of the read-one write-all protocol. The voting approach, both static [14, 27] and dynamic [7, 18, 23], does not require write operations to write all copies, and thus increases their fault-tolerance. The price paid is

that read operations must, in general, read several copies, rendering read operations more expensive than in the read-one write-all protocol. The second approach [10, 11, 17] uses configuration information to provide fault-tolerant operations without requiring read operations to access several copies. In particular, each site maintains some information about its communication capabilities, and may use that to ensure that read operations access a single copy. This improved performance is, however, attained by requiring a special reconfiguration protocol to be executed whenever a change in the network configuration occurs.

In this paper we present a new protocol that achieves the main advantage of the voting approach, i.e., fault-tolerance of write operations without special reconfiguration mechanisms, as well as of the configuration-based approach, i.e., inexpensive fault-tolerant read and write operations. In particular, operations access few copies when there are no failures in the system. As failures occur operations may be required to access more copies. Furthermore, operations tolerate failures without the need for a reconfiguration protocol. This behavior is attained by imposing a logical tree structure on the set of copies of the object. This structure is used by operations to determine the copies that must be read or written. In Figure 1, an example of a ternary tree with thirteen copies of an object is illustrated. Note that this structure is logical, and does not have to correspond to the actual physical structure of the network connecting the sites storing the copies. We will use this tree structure to motivate a special instance of the proposed protocol.

A straightforward replica control protocol that uses the tree structure is one where a write operation is required to write a majority of copies at *all* levels of the tree, e.g., in the tree of Figure 1, any set consisting of the root, and any two copies from $\{2, 3, 4\}$ as well as a majority from $\{5, 6, 7, 8, 9, 10, 11, 12, 13\}$. In this case, a read operation can be executed by accessing a majority of copies at any *single* level of the tree. For example, any of the following sets could form a read quorum: the set consisting of the root, or any set containing two copies from $\{2, 3, 4\}$, or any set containing a majority from $\{5, 6, 7, 8, 9, 10, 11, 12, 13\}$. The protocol described above uses read-one write-all approach to access the levels of the tree and within a level uses the majority protocol. Hence, any read operation must have at least one copy in common with any write operation, and therefore the protocol ensures one-copy equivalence.

This simple protocol has similar performance for read operations as the read-one write-all but has better fault-tolerance for write operations. In particular, when the root is accessible, read operations can always be executed by accessing a single copy. Furthermore, write operations can be executed even after the failure of several copies at different levels, e.g., the failure of copies 4, 7, 8, 11 and 12 does not prohibit write operations. The protocol is therefore similar to the read-one write-all protocol in that when there are no failures read operations access a single copy. As failures occur, this protocol may still execute write operations as well as read operations but at a higher cost. In particular, read operations can tolerate the failure of all

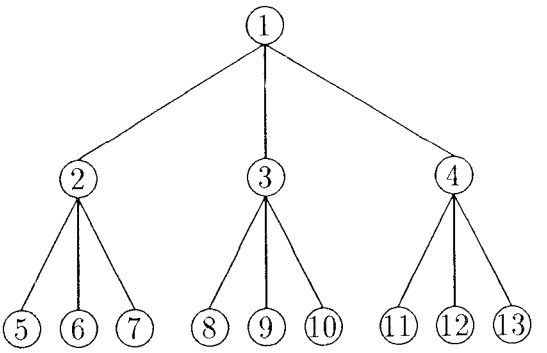

Fig. 1. A tree organization of 13 copies of an object.

except a majority of copies at some level, and write operations can tolerate the failure of a minority[1] of copies at all levels.

Although correct, the performance of this protocol can be improved by further exploiting the tree structure. Instead of requiring a write operation to write a majority of copies at all levels, consider a write operation that writes the root, a majority of its children, and a majority of their children, and so forth. Hence for the tree of Figure 1, a write operation could be executed by writing the following set of copies only: $\{1, 2, 3, 5, 6, 8, 9\}$, which is smaller than the set required by the simple protocol. To ensure the quorum intersection property, a read operation tries to access the root; if the root is inaccessible, the read tries to access a majority of the root's children. For each inaccessible copy in this majority set, the read operation tries to access a majority of its children. For example, consider a network configuration where copies 1 (the root), 2, and 3 are inaccessible. In this case the read may form a quorum by accessing copy 4 and a majority of copy 2's children, e.g., 5 and 7. Alternatively, the quorum may be formed from copy 4 and a majority of copy 3's children, e.g., 9, 10. A read quorum could also have been formed by selecting a majority of children of any two of copies 2, 3, and 4. All read quorums have a nonempty intersection with any write quorum, e.g., $\{1, 2, 3, 5, 6, 8, 9\}$.

While read operations in this protocol are fault-tolerant, a write operation may not execute if the root is inaccessible. In the rest of this paper we develop a generalization of the simple tree protocol described above and the quorum protocol. Instead of requiring each write operation to write copies at *all* levels of the tree (including the root), we require it to write a quorum of levels only. As a result, read operations are required to access a quorum of levels so that there is an intersection of a level between a read and a write operation. Also, instead of requiring each operation to access a majority of children, operations are required to access a quorum of children such that read and

[1] A majority is $\lceil (n + 1)/2 \rceil$ copies and a minority is $\lfloor (n - 1)/2 \rfloor$ copies.

write operations have an intersection of at least one child. These generalizations result in quorums with two degrees of freedom: number of levels and number of children. The sizes of quorums may vary depending on the degree of fault-tolerance as well as the cost associated with the execution of operations. With a simple read-one level write-all levels approach, read operations access a single copy in a failure-free environment, but write operations cannot tolerate the failure of the root. On the other hand, in a protocol where both read and write operations access a majority of levels, read operations must access several levels even when there are no failures in the system. Write operations, however, can be executed even when all the nodes at a minority of the levels have failed. These two examples represent the two extremes in a spectrum of different quorum assignments. In the rest of the paper, we formally develop the protocol and evaluate its performance.

4. THE TREE QUORUM PROTOCOL

In this section, we present the tree quorum protocol for accessing objects in a distributed replicated database. We first present the algorithm for constructing quorums for replicated objects organized in the form of a logical tree. We then demonstrate the correctness of the algorithm, and conclude by showing that tree quorums cannot be implemented by assigning different votes to the copies of an object.

4.1 The Algorithm for Constructing Tree Quorums

The standard approach for implementing quorums associates with each copy a *vote* (often this vote is one). The read quorum for an object x is any set of copies with q_r votes, and a write quorum is any set with q_w votes. To ensure the quorum intersection property, the sum of q_r and q_w must be greater than the total number of votes of x. A simple protocol would require that both read and write quorums contain a majority of copies. Our approach for implementing quorums imposes a logical tree structure on the copies of an object x. Instead of counting votes, a special tree-based protocol is used to construct quorums.

Given a set of n copies of an object x, we logically organize them into a tree of height h, and degree d, i.e., each node has d children, and the maximum height is h. We will assume the standard tree terminology, i.e., root, child, parent, leaf, level, etc. We also assume that the tree is complete, i.e., it has the maximum number of nodes (this restriction will be relaxed in Section 6). The logical tree organization may be viewed simply as an ordering of the copies maintained in the name directory at each site for location purposes. Extending it to a tree does not impose any extra or special complications.

We assume that the tree has a well defined root, c_0, and that a transaction attempting to construct a quorum calls the recursive function *GetQuorum* with the root of the tree c_0 and $\langle l, w \rangle$ as parameters (see Figure 2). The set constructed is called a *tree quorum* of length l and width w and will be denoted by the pair $\langle l, w \rangle$. The protocol tries to construct a quorum by selecting the root and w children of the root, and for each selected child, w of

```
TYPE
    TREE: POINTER TO TreeNode;

    STRUCTURE TreeNode
        Node: COPY;
        Child. ARRAY[1..Degree] OF TREE;
        (* other fields *)
    END; (* TreeNode *)

FUNCTION GetQuorum(Tree: TREE; ⟨Length : INTEGER, Width : INTEGER⟩): QUORUM;
    VAR
        ChildQuorum, TempQuorum : QUORUM;
    BEGIN
    IF Length > Height(Tree) THEN
        (* Unsuccessful in establishing a quorum *)
        EXIT(error);
    ELSE IF Length = 0 THEN
        RETURN({});
    ELSE IF (Tree ↑ .Node) grants permission THEN
        (* Let TempQuorum be a set of children, such that |TempQuorum| = Width *)
        ChildQuorum ← ⋃_{i∈TempQuorum} GetQuorum(Tree ↑ .Child[i], Length − 1, Width);
        RETURN(Tree ↑ .Node ∪ ChildQuorum);
    ELSE
        (* Let TempQuorum be a set of children, such that |TempQuorum| = Width *)
        ChildQuorum ← ⋃_{i∈TempQuorum} GetQuorum(Tree ↑ .Child[i], Length, Width),
        RETURN(ChildQuorum);
    END; (* IF *)
END GetQuorum.
```

Fig. 2. The algorithm for constructing tree quorums with arbitrary length and arbitrary width.

its children, and so on, for depth l. If successful this forms a tree quorum of height l and degree w. However, if some node is inaccessible at depth h' from the root while constructing this tree quorum then the node is replaced recursively by w tree quorums of height $l - h'$ starting from the children of the inaccessible node. The recursion terminates successfully when the length of the quorum to be constructed is zero. The algorithm fails to construct a quorum if the length of the quorum exceeds the height of the remaining subtree.

Consider a replicated object with thirteen copies. We superimpose a ternary tree of height 3 on the copies as illustrated in Figure 1, with the sites numbered as shown. In this case $d = 3$ and $h = 3$. We now construct tree quorums of length 1 and width 2. In the best case, the quorum need only contain the root. However, as a result of the failure of the root, a tree quorum of dimensions $\langle 1, 2 \rangle$ can be formed from any majority (two) of the root's children, i.e., $\{2, 3\}$ or $\{2, 4\}$ or $\{3, 4\}$. If a majority or more of the root's children have failed, then each such copy can be replaced by a majority of its children. Hence, if copies 1, 2, and 3 are inaccessible, then a quorum can be formed from copy 4 and a majority of either copy 2 or copy 3's children, e.g., the sets

$\{4, 5, 6\}$ and $\{4, 8, 10\}$ form quorums. Similar quorums can be formed if either copies 3 and 4 or copies 2 and 4 are inaccessible. Finally, if copies 1, 2, 3, and 4 are inaccessible, then a majority of the children of any two of copies 2, 3 and 4 form a quorum. For example, the following sets are candidate quorums: $\{5, 6, 8, 9\}$, $\{6, 7, 12, 13\}$, $\{8, 10, 11, 13\}$, etc. A quorum with dimensions $\langle 3, 2\rangle$ must include the root. In addition, it must include a majority of copies 2, 3, and 4 and for each such pair it must include a majority of their children. For example the following sets form a $\langle 3, 2\rangle$ quorum: $\{1, 2, 3, 5, 6, 8, 9\}$, $\{1, 2, 4, 6, 7, 12, 13\}$, etc. As a final example consider quorums with dimensions $\langle 2, 2\rangle$. If no failures occur, then any of the following four sets form a valid quorum: $\{1, 2, 3\}$, $\{1, 2, 4\}$, and $\{1, 3, 4\}$. If the root is inaccessible (due to site failures or network partitioning), then any set containing any two copies out of the set $\{2, 3, 4\}$ in addition to any two of their children forms a required quorum. A set containing the root 1, any one of $\{2, 3, 4\}$, and any two children of either of the other two copies also forms a quorum. Finally, if all three children of the root are inaccessible, then a set with the root and any two children of two of the inaccessible copies form a quorum, e.g., $\{1, 5, 7, 8, 9\}$, $\{1, 8, 10, 12, 13\}$, and so on.

4.2 Proof of Correctness

A transaction attempting to execute a read operation calls the recursive function *GetQuorum* with the root of the tree, c_0 and $q_r = \langle l_r, w_r\rangle$, as parameters. The protocol tries to construct such a quorum and if it fails the operation is aborted. A write operation is executed similarly by attempting to construct a tree quorum with dimensions $q_w = \langle l_w, w_w\rangle$. The following constraints guarantee the nonempty intersection of read and write quorums:

$$l_r + l_w > h \tag{1}$$

$$w_r + w_w > d \tag{2}$$

If version numbers [14] are used for replica synchronization, the following constraints ensure the nonempty intersection of any two write quorums:

$$2 \cdot l_w > h \tag{3}$$

$$2 \cdot w_w > d \tag{4}$$

However, if logical timestamps are used instead of version numbers then the second set of constraints is not required [16]. The following theorem establishes the correctness of the tree quorum protocol. We demonstrate that the read and write quorums constructed by the tree protocol will always have a nonempty intersection. If transactions use a correct concurrency control protocol then all executions will be one-copy serializable. A similar proof shows that if write quorums satisfy Eqs. 3 and 4, then any two write quorums have a nonempty intersection.

THEOREM 1. *In a tree of height h and degree d, the tree quorum protocol guarantees that for any read quorum of size $q_r = \langle l_r, w_r\rangle$ and any write quorum $q_w = \langle l_w, w_w\rangle$ such that Eqs. 1 and 2 hold, the read and write quorums must have a nonempty intersection.*

PROOF. The proof is by induction on the height of the trees.

Basis. The theorem holds for a tree of height one, since there is only one copy in the tree.

Induction hypothesis. Assume that the theorem holds for trees of height h and quorums of lengths such that $l_r + l_w > h$. Note that the degree of the tree, d, is fixed.

Induction step. Consider a tree of height $h + 1$ and read and write quorums such that $l_r + l_w > h + 1$ and $w_r + w_w > d$. We now prove that these two quorums must have a nonempty intersection. There are two cases to consider.

Case 1. Read quorum q_r includes the root of the tree, c_0. In this case q_r must include w_r tree quorums of length $l_r - 1$ in any w_r subtrees whose roots are c_0's children. Write quorum q_w may either include the root or not. If it includes the root, then the two quorums have a nonempty intersection. If q_w does not include the root, then it must include w_w tree quorums of length l_w in any w_w subtrees whose roots are c_0's children. Since $w_r + w_w > d$, read and write quorums must have at least one subtree in common. In this subtree the sum of the lengths of the quorums is $(l_r - 1) + l_w$. But $l_r + l_w > h + 1$, and therefore $(l_r - 1) + l_w > h$. Any subtree with root a child of c_0 has a height of h, and therefore from the induction hypothesis, the tree quorums in the subtree have a nonempty intersection. Hence, read and write quorums have a nonempty intersection.

Case 2. Read quorum q_r does not include the root of the tree, c_0. In this case q_r must include w_r tree quorums of length l_r in any w_r subtrees whose roots are c_0's children. Write quorum q_w either does not include the root or it does. If it does not include the root, then it must contain w_w tree quorums of length l_w in any w_w subtrees whose roots are c_0's children. From Eq. 2, there must be a subtree in common between the two quorums, of height h, where q_r and q_w include quorums of lengths l_r and l_w on the subtree in common. Since $l_r + l_w > h$, by the induction hypothesis it follows that q_r and q_w have a nonempty intersection. If q_w includes the root, then it must include w_w tree quorums of length $l_w - 1$ in any w_w subtrees whose roots are c_0's children. Again, there must be a subtree in common between the two quorums, of height h, where q_r and q_w includes quorums of lengths l_r and $l_w - 1$ on this subtree. Since $l_r + l_w > h + 1$, $l_r + (l_w - 1) > h$. From the induction hypothesis, the tree quorums in the subtree have a nonempty intersection, and hence q_r and q_w have a nonempty intersection.

Hence, by induction, the theorem follows. □

4.3 Tree Quorums versus Vote Assignments

We now argue that the tree quorum protocol cannot be implemented by assigning nonnegative integer votes to the copies in the system. That is, the tree quorum protocol does not have an equivalent vote assignment in the voting protocol [14]. Consider an object with thirteen copies organized as a

ternary tree of height three as shown in Figure 1. By using the tree quorum protocol in which the read quorums are $q_r = \langle 1, 2 \rangle$ and write quorums are $q_w = \langle 3, 2 \rangle$, the following sets are valid read and write quorums:

(1) Read Quorum: {1}.

(2) Write Quorums: {1, 2, 3, 5, 6, 8, 9}, {1, 2, 3, 6, 7, 8, 9},

Let v_i be the nonnegative integer vote associated with copy i, and let N be the total number of votes. Then from the above sets, we can write the following equations:

$$(v_1) + (v_1 + v_2 + v_3 + v_5 + v_6 + v_8 + v_9) > N, \tag{5}$$

$$(v_1) + (v_1 + v_2 + v_3 + v_6 + v_7 + v_8 + v_9) > N. \tag{6}$$

Equations 5 and 6 must hold since the sum of the votes in the read and write quorums must exceed the total number of votes in the system. Note that, as mandated by the protocol, the following sets are not valid write quorums:

(1) Invalid Write Quorums: {1, 2, 3, 5, 6, 7, 8}, {1, 2, 3, 6, 8, 9, 10}.

Since the above sets are not valid write quorums, the total votes of the copies in each of the above sets when added to the votes in the read quorum must be smaller than N, i.e.,

$$(v_1) + (v_1 + v_2 + v_3 + v_5 + v_6 + v_7 + v_8) < N, \tag{7}$$

$$(v_1) + (v_1 + v_2 + v_3 + v_6 + v_8 + v_9 + v_{10}) < N. \tag{8}$$

By solving Eqs. 5 and 7, we get $v_7 < v_9$, and from the other two equations, we get $v_7 > v_9$. Clearly, there is no integer vote assignment that can satisfy both these relations simultaneously. Hence, there exists no equivalent vote assignment for the tree quorum protocol. A similar technique was used to show that coteries may not always be implemented using nonnegative integer vote assignments [13].

An operation can also be executed by broadcasting the request to all copies. After receiving the replies from all sites, the operation succeeds if the set of sites that replied is a superset of any quorum. Another alternative is to choose a specific quorum and send the request only to those sites. The selection criterion for the specific quorum would depend on factors such as: communication cost, load balancing, etc. In particular, load balancing may be needed to avoid excessive queueing delays at the root or copies near the root.

The tree organization of replicated objects may be constructed based on several possible criteria. One way to organize the tree would be to capture the logical relationship among the copies of an object. For example, in a banking application, customers generally operate from a specific physical branch most of the times. However, occasionally they may access their accounts from other branches of the bank. In such applications, we can organize the tree such that the main account becomes the root and other copies of the account are in the lower levels of the tree. Another criterion for organizing trees is to capture the physical reliability of different computer sites storing the copies of the

object. In this case, more reliable sites should be at or near the root and less reliable sites should be at the leaves.

5. ANALYSIS OF THE TREE PROTOCOL

In this section, we compute the cost and the availability of read and write operations of various instances of the tree quorum protocol and compare them with the read-one write-all (ROWA) and the voting (VOTE) protocols [14, 27].

5.1 Expected Cost Analysis

The message cost of an operation is directly proportional to the size of the quorum required to execute that operation. Thus in the read-one write-all approach, read operations have a cost of one while write operations have a cost of n. In the voting protocol, the quorum size corresponding to a majority provides maximum availability [5]. We consider read operations to have a cost of $(n + 1)/2$ and write operations to have a cost of $n/2 + 1$[2]. Note that this ensures that when there is an even number of copies, reads are cheaper to execute than writes.

In the tree quorum protocol, if read quorums have dimensions $\langle l, w \rangle$, then from Eqs. 1 and 2 write quorums must have dimensions $\langle h - l + 1, d - w + 1 \rangle$, where h is the height and d is the degree of the tree. Thus, the size of a read quorum may be as small as $(w^l - 1)/(w - 1)$ if all copies in the quorum are from the upper levels of the tree. In the worst case, the size may increase to $w^{l-1} \cdot (w^l - 1)/(w - 1)$ when all copies in the quorum are from the lower levels of the tree. Similarly, the size of write quorums vary from $[(d - w + 1)^{h-l+1} - 1]/(d - w)$ to $(d - w + 1)^{h-l} \cdot [(d - w + 1)^{h-l+1} - 1]/(d - w)$. In order to compute the average cost of an operation, we introduce a parameter f that indicates the fraction of operations that use the root of the tree to execute. Let $MC_h[l, w]$ be the average cost of executing operations with a tree quorum of length l and width w in a tree of height h. Thus, the cost $MC_{h+1}[l, w]$ for a tree of height $h + 1$ is:

$$MC_{h+1}[l, w] = f(1 + w \cdot MC_h[l - 1, w]) + (1 - f) \cdot w \cdot MC_h[l, w] \quad (9)$$

where $MC_l[l, w]$ is $[w^l - 1]/(w - 1)$ and $MC_h[0, w] = 0$. Note that the first term in the recurrence relation corresponds to the fraction of operations, f, that execute using the root of the tree and collect quorums of length $l - 1$ on w subtrees. The second term corresponds to the situation when the operations do not access the root and collect quorums of length l on w subtrees. Thus, $f = 0$ is the upper bound on the cost of operations and $f = 1$ is the lower bound on the cost.

Figure 3 illustrates the cost of executing operations in various replica control protocols. We consider an instance of the tree quorum protocol where read quorums are $\langle (h + 1)/2, (d + 1)/2 \rangle$ and write quorums are $\langle h/2 +$

[2] We are assuming integer division.

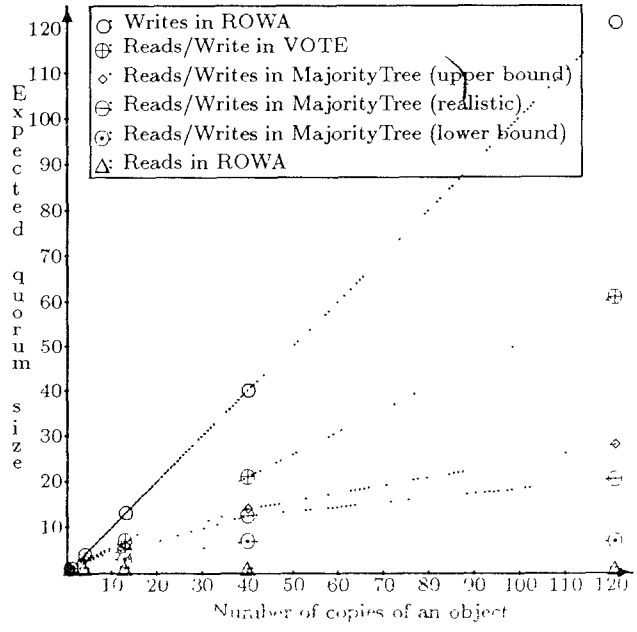

Fig. 3. Expected cost; ternary tree· $q_r = \langle (h+1)/2, 2 \rangle$, $q_u = \langle h/2 + 1, 2 \rangle$.

$1, d/2 + 1\rangle$. This instance is referred to as the *majority tree* protocol since the lengths correspond to a majority of levels in a tree of height h and the widths correspond to a majority of children at a level in a tree with degree d. For the majority tree protocol and the majority voting protocol, only the cost of executing write operations is illustrated for clarity. In fact, the cost of read operations differ (and is smaller) when the height of the tree is an even number (for majority tree) or when the number of copies is an even number (for majority voting); otherwise the read and write costs are identical. For operations in the majority tree protocol, we indicate the upper and lower bounds on the costs, and provide a realistic estimate for the costs when $f = 0.5$ (i.e., 50% of operations execute using the root of any subtree). When compared to majority voting, both read and write operations are significantly less costly in the majority tree protocol. Write operations are also significantly less costly than in the read-one write-all protocol; however, the increase in fault-tolerance and decrease in communication for writes is at the expense of larger read quorums. Thus, in terms of communication costs the majority tree protocol has definite advantages.

5.2 Availability Analysis

In this section, we demonstrate that the availability of read and write operations in the majority tree protocol is comparable to majority voting. Let p be the probability that a copy of an object is available for read or write operations. Furthermore, assume that there are n copies of the object in the

system. Since read operations on this object can be executed by accessing any copy of the object in the read-one write-all protocol, the availability of read operations is $[1 - (1 - p)^n]$. Since all copies are needed to execute write operations in this protocol, the availability of write operations is p^n.

In the case of the voting protocol, the majority quorum assignment is optimal for both read and write operations [5]. Thus, the availability of read and write operations is:

Probability(*majority* copies are available)
+ Probability(*majority* + 1 copies are available)
...
+ Probability(*majority* + i copies are available)
...
+ Probability(all copies are available).

If we let n be equal to $2k + 1$ for some nonnegative constant k, the above probabilities can be represented by the following terms, i.e.,

$$= \binom{2k+1}{k+1} p^{k+1}(1-p)^k + \cdots + \binom{2k+1}{k+i} p^{k+i}(1-p)^{k-i+1} + \cdots + p^{2k+1}.$$

The availability of read and write operations in the tree quorum protocol can be computed by formulating recurrence relations for both read and write availabilities. The recurrence relation is in terms of the availabilities of these operations in the subtrees of a tree of copies of an object. Let $A_h[l, w]$ be the availability of operations that require a tree quorum of length l and width w in a tree of height h and degree d. Thus, the availability in a tree of height $h + 1$ is given as:

$$\begin{aligned} A_{h+1}[l, w] = {} & \text{Probability(Root is up)} \\ & \times [\text{Availability of } w \text{ subtrees with } A_h[l-1, w]] \\ & + \text{Probability(Root is down)} \\ & \times [\text{Availability of } w \text{ subtrees with } A_h[l, w]]. \end{aligned}$$

By taking p as the probability that the root is available and $1 - p$ as the probability that the root is unavailable, we get:

$$\begin{aligned} A_{h+1}[l, w] = p \times \Bigg[& \binom{d}{w} (A_h[l-1, w])^w (1 - A_h[l-1, w])^{d-w} + \cdots + \\ & \binom{d}{w+i} (A_h[l-1, w])^{w+i} (1 - A_h[l-1, w])^{d-w-i} \\ & + \cdots + (A_h[l-1, w])^d \Bigg] \end{aligned}$$

$$+(1-p)\times\left[\binom{d}{w}(A_h[l,w])^w(1-A_h[l,w])^{d-w}+\cdots+\binom{d}{w+i}(A_h[l,w])^{w+i}(1-A_h[l,w])^{d-w-i}+\cdots+(A_h[l,w])^d\right].$$

Note that $A_h[l,w]=0$ for $l>h$, $A_h[0,w]=1$, and $A_1[1,w]=p$. Since the above recurrence relations involve nonlinear terms, we illustrate the operation availabilities for specific replica configurations of an object in Figures 4 and 5. For the majority tree protocol, read quorums have dimensions $\langle (h+1)/2,(d+1)/2\rangle$ and write quorums have dimensions $\langle h/2+1, d/2+1\rangle$. In Figure 4, the object is replicated at four sites and is organized as a ternary tree of height two. In Figure 5, the object is replicated at thirteen sites and the tree is assumed to be a ternary tree of height three. The availability of read operations in all three protocols is almost identical. In the case of four copies, the write availability of the majority tree protocol is inferior to the majority voting protocol. However, it is substantially superior to the write availability in the read-one write-all protocol. In the case of thirteen copies, the availability in the majority tree protocol is almost the same as the majority voting protocol for both read and write operations. Recall that, for the chosen configurations of read and write quorums, the cost of operations in the majority tree protocol is substantially lower than majority voting. Thus when there is a large number of copies, the majority tree protocol provides high availability of operations at substantially lower costs.

5.3 Special Instances of the Tree Quorum Protocol

The quorum sizes in the tree quorum protocol have two degrees of freedom. First, the tree quorums can be varied by changing the lengths of read and write quorums. Second, the tree quorums can be varied by changing their widths. For example in the previous subsection, the read quorums have length $(h+1)/2$ and width $(d+1)/2$ while the write quorums have length $h/2+1$ and width $d/2+1$, which corresponds to a majority in both degrees of freedom. We analyze the following special instances of the tree quorum protocol:

Read root. Read quorums have dimensions $\langle 1,(d+1)/2\rangle$ and write quorums are $\langle h, d/2+1\rangle$.

Log write. Read quorums have dimensions $\langle 1, d\rangle$ and write quorums are $\langle h, 1\rangle$.

In the read root protocol [5], a read operation is executed by accessing a *single* copy in the best case. However, if the root fails, the cost of forming a quorum increases to $(d+1)/2$, a majority of the root's children. As more failures occur, the quorum size increases up to a maximum of $[(d+1)/2]^h$. Note that, in general, $[(d+1)/2]^h < n/2$. Write operations, on the other hand, are all of the same size: $[[(d+1)/2]^{h+1}-1]/[(d-1)/2]$.

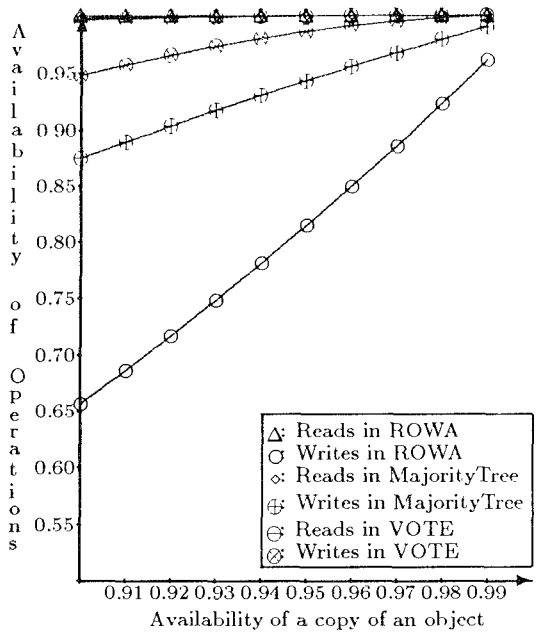

Fig. 4. Availability of an object with four copies; ternary tree: $q_r = \langle 1, 2 \rangle$, $q_w = \langle 2, 2 \rangle$.

Figure 6 illustrates the cost of executing read and write operations in the read root protocol. For the read operations in this protocol, we indicate the upper and lower bounds on the cost, and provide a realistic cost estimate, which corresponds to $f = 0.5$ (i.e., 50% of read operations execute using the root of any subtree). In fact, for a ternary tree if $f = 0.5$, the above recurrence relation yields the read cost to be $(h + 1)/2$ which is logarithmic in the number of copies. The cost of executing read operations is comparable in both the read root and the read-one write-all protocols. Write operations, on the other hand, are significantly less costly. When compared with the voting protocol, both read and write operations are less costly. Thus, in terms of cost the read root protocol has definite advantages.

In Figure 7, the availability of read and write operations in the read root protocol is illustrated for a ternary tree with thirteen copies. The read availability in this protocol is comparable to both the read-one write-all and the majority voting protocols. Write availability, however, is inferior to majority voting but is substantially superior to the read-one write-all protocol. In particular, for the case of thirteen copies, write availability increases from 25% to more than 85%, when $p = 0.9$. In conclusion, the read root protocol achieves the benefits of the read-one write-all protocol while significantly improving both the cost and the availability of write operations. Write availability is better in the case of majority tree protocol (see Figure 5), however, this is at the cost of more expensive read operations (see Figure 3).

Now, we describe the log write protocol, where write costs are decreased significantly by requiring that write operations access a set of copies that

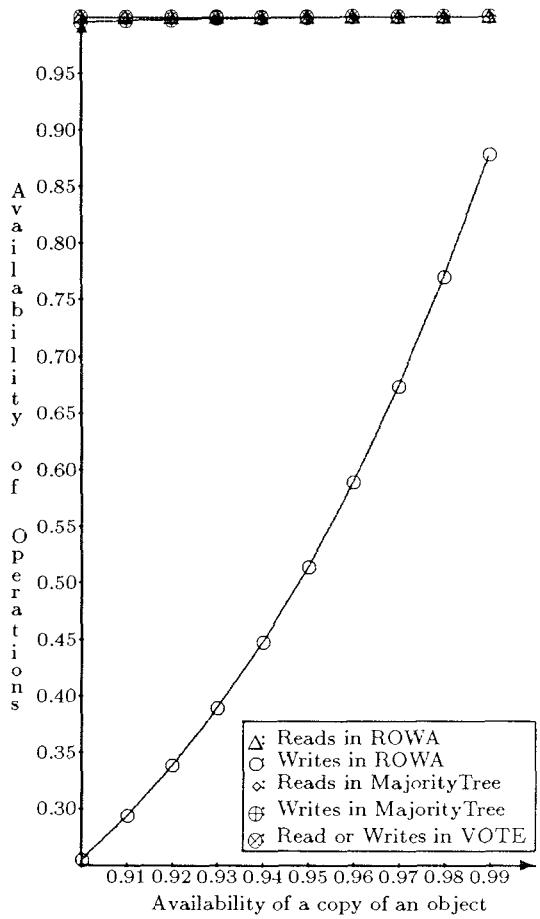

Fig. 5. Availability of an object with thirteen copies; ternary tree: $q_r = \langle 2, 2 \rangle$; $q_u = \langle 2, 2 \rangle$.

form a path from the root to one of the leaves in the tree, i.e., the width of write quorums is set to one. Read operations, on the other hand, form a front, i.e., the root of the tree or all of its children, and recursively for each inaccessible child all of its children must be included. Thus, read quorums have a length of one and width of d. As shown in Figure 8, the costs of both read and write operations are on the average less than the majority tree protocol. The cost of read operations in the best case is one (the root) while in the worst case it could be all the leaves of the tree. Unlike the read root protocol, the log write protocol is biased towards write operations in terms of cost, since all write operations have logarithmic cost (Figure 8). However, in terms of availability, the protocol is still biased towards read operations since the read availability is almost identical to the read-one write-all and the majority voting protocol (Figure 9).

In conclusion, we note that, in general, the proposed protocols exhibit the property of graceful degradation [22]. Operations with varying cost have a

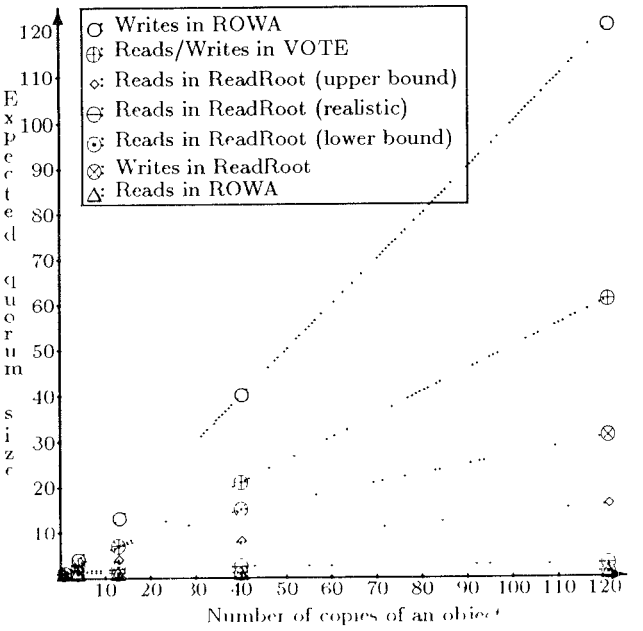

Fig. 6. Expected cost; ternary tree: $q_r = \langle 1, 2 \rangle$, $q_w = \langle h, 2 \rangle$.

property that in the absence of failures the cost of forming a quorum is minimal, and this cost increases as failures occur in the system. For example, both the read root and log write protocols require the reading of a single copy when there are no failures, but as failures occur the cost increases and in the worst case may be proportional to the number of leaves in the tree. The property of graceful degradation is particularly important in distributed database systems since it provides fault-tolerance without imposing unnecessary costs when there are no failures.

6. INCOMPLETE TREES

The tree quorum protocol assumes that the underlying tree structure is complete. This assumption is undesirable since it restricts the number of copies of an object. For example, for ternary trees the number of copies can only be 1, 4, 13, etc. In this section, we first describe an extension of the tree quorum construction algorithm that can be applied to incomplete trees. We then analyze the tree quorum protocol for incomplete trees. We conclude with a discussion of how to structure incomplete trees.

6.1 The Tree Quorum Protocol for Incomplete Trees

A tree of height h and degree d is called an *incomplete tree* if the height of the root is h and each node in the tree has at most d children. Note that in a complete tree all interior nodes must have exactly d children. Figure 10 illustrates an incomplete tree with height 3 and degree 3.

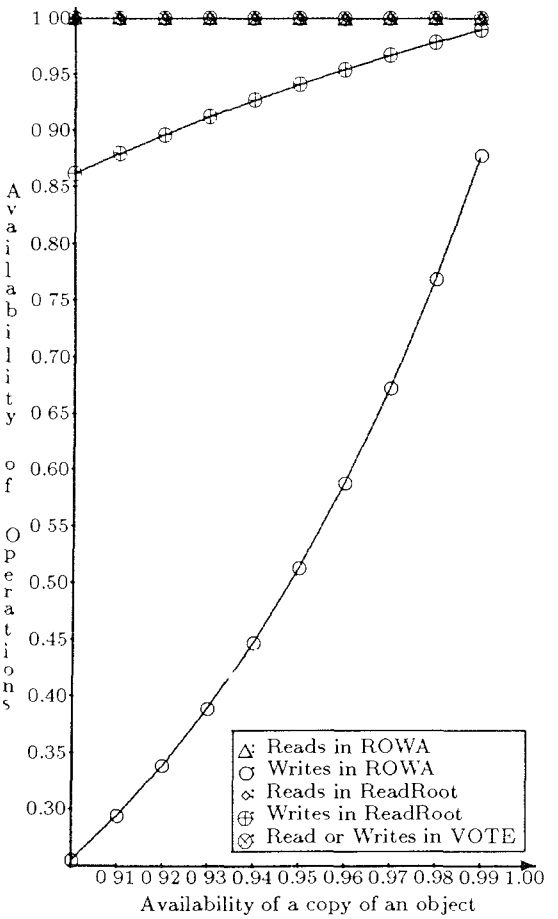

Fig. 7. Availability of an object with thirteen copies; ternary tree $q_r = \langle 1, 2 \rangle$, $q_w = \langle 3, 2 \rangle$

In Figure 11, we describe a protocol for constructing quorums on incomplete trees. In this protocol, an incomplete tree is viewed as a complete tree with some subtrees missing. As before, the protocol tries to construct a quorum by selecting the root and w children of the root, and for each selected child, w of its children and so on. If successful this forms a quorum of height l and degree w. However, if some node is inaccessible at depth h' while constructing this tree then the node is replaced recursively by w subtrees of height $l - h'$. Let n_i be the actual number of children of node c_i in the tree. Thus, each node has $d - n_i$ missing children. When node c_i is included in a tree quorum $d - n_i$ subtrees with the missing children of c_i as roots are implicitly included in that quorum. Hence, instead of accessing w subtrees (recursively) while constructing a tree quorum, only $w - (d - n_i)$ subtrees need to be included for each node c_i. If $w - (d - n_i)$ is zero or negative then no more subtrees are needed. The recursion terminates successfully when the

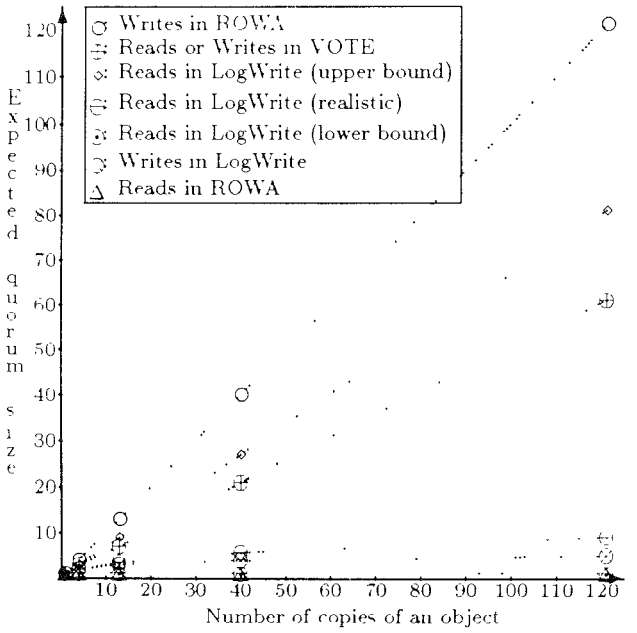

Fig. 8. Expected cost; ternary tree: $q_r = \langle 1, 3 \rangle$, $q_w = \langle h, 1 \rangle$

length of the quorum to be constructed is zero and it terminates unsuccessfully when the length of the quorum exceeds the height of the subtree.

When the algorithm uses the tree of Figure 10, it would generate the following quorums when $q_r = q_w = \langle 2, 2 \rangle$. If the root is accessible then only copies 2 or 3 need to be included, i.e., $\{1, 2\}$ and $\{1, 3\}$ are valid quorums. If the root is inaccessible, then copies 2 and 3 must be included in the quorum as well as one of 4 and 5 and two of 6, 7, and 8, e.g., $\{2, 3, 4, 6, 7\}$. The following theorem establishes the correctness of the algorithm.

THEOREM 2. *In a tree of height h and degree d, the incomplete tree quorum protocol guarantees that for any read quorum* $q_r = \langle l_r, w_r \rangle$ *and any write quorum* $q_w = \langle l_w, w_w \rangle$ *such that Eqs. 1 and 2 hold, the read and write quorums must have a nonempty intersection.*

PROOF. The proof is by induction on the height of the trees.

Basis. The theorem holds for a tree of height one, since there is only one copy in the tree.

Induction hypothesis. Assume that the theorem holds for trees of height h and quorums $\langle l_r, w_r \rangle$ and $\langle l_w, w_w \rangle$ such that $l_r + l_w > h$. Note that the degree of the tree, d, is fixed.

Induction step. Consider a tree of height $h + 1$ and read and write quorums such that $l_r + l_w > h + 1$ and $w_r + w_w > d$. We now prove that these two quorums must have a nonempty intersection. Two cases arise depending on whether the read quorum includes the root of the tree or not.

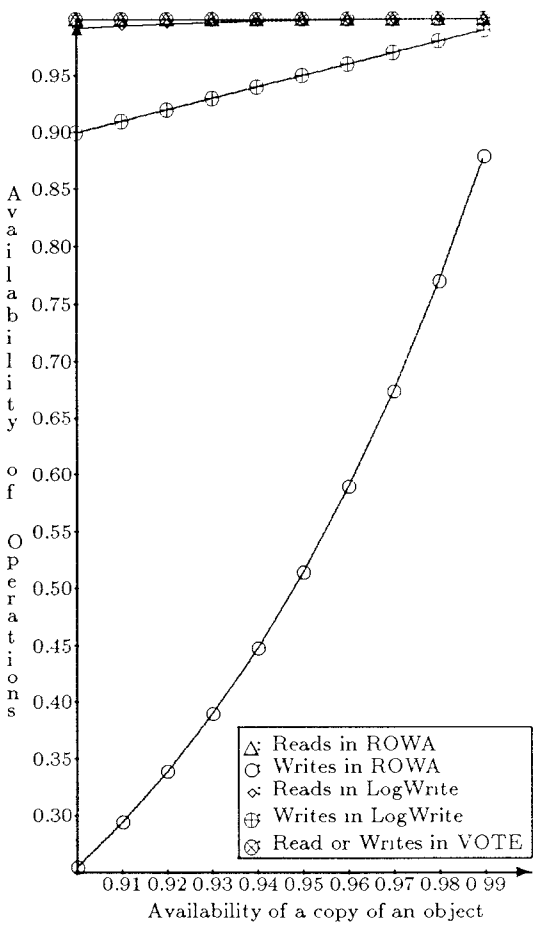

Fig. 9. Availability of an object with thirteen copies; ternary tree: $q_r = \langle 1, 3 \rangle$, $q_w = \langle 3, 1 \rangle$.

Case 1. Read quorum q_r includes the root of the tree, c_0. Let n_0 be the number of actual children of c_0. Clearly, $n_0 \leq d$. Note that $d - n_0$ are the missing children of c_0 since we are considering trees that are incomplete. If $w_r < (d - n_0)$ then the read operation terminates successfully if the height of root c_0 is greater than or equal to l_r. Otherwise, q_r must include $w_r - (d - n_0)$ tree quorums of length $l_r - 1$ in any subtrees whose roots are c_0's actual children. Write quorum q_w may either include the root too or not. If it includes the root, then the two quorums have a nonempty intersection. If q_w does not include the root, then it must include w_w tree quorums of length l_w in any w_w subtrees whose roots are c_0's physical children. There are two cases to consider.

First, when q_r includes $w_r - (d - n_0)$ subtrees. Thus, the number of the subtrees included in the read and write quorums is: $w_r - (d - n_0) + w_w$, i.e., $w_r + w_w - (d - n_0)$. Since $w_r + w_w > d$, the number of subtrees is greater

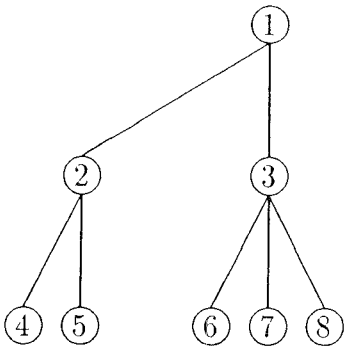

Fig. 10. An incomplete ternary tree of height 3 with 8 copies.

```
TYPE
    TREE POINTER TO TreeNode,

    STRUCTURE TreeNode
        Node COPY,
        Child ARRAY[1 Degree] OF TREE,
        Degree INTEGER,
        ChildCount INTEGER, (* Number of actual children *)
        (* other fields *)
    END, (* TreeNode *)

FUNCTION GetQuorum(Tree TREE, (Length INTEGER, Width INTEGER)) QUORUM,
VAR
    ChildQuorum, TempQuorum QUORUM,
    SubtreesNeeded INTEGER,
BEGIN
    IF Length > Height(Tree) THEN
        (* Unsuccessful in establishing a quorum *)
        EXIT(error),
    ELSE IF Length = 0 THEN
        RETURN({}),
    ELSE IF (Tree | Node) grants permission THEN
        SubtreesNeeded ← Width − (Tree | Degree − Tree | ChildCount),
        IF SubtreesNeeded > 0 THEN
            (* Let TempQuorum be a set of children, such that |TempQuorum| = SubtreesNeeded *)
            ChildQuorum ← ⋃_{i∈TempQuorum} GetQuorum(Tree | Child[i], Length − 1, Width),
        END, (* IF *)
        RETURN(Tree | Node ∪ ChildQuorum),
    ELSE
        (* Let TempQuorum be a set of children such that |TempQuorum| = Width *)
        ChildQuorum ← ⋃_{i∈TempQuorum} GetQuorum(Tree | Child[i], Length, Width),
        RETURN(ChildQuorum),
    END, (* IF *)
END GetQuorum
```

Fig. 11. The algorithm for constructing tree quorums for incomplete trees.

than n_0. Hence, the read and write quorums must have at least one physical subtree in common. In this subtree the sum of the length of the quorums is $l_r - 1 + l_w$. But $l_r + l_w > h + 1$, and therefore $l_r - 1 + l_w > h$. Any subtree whose root is a child of c_0 has height h, and therefore from the induction hypothesis, the tree quorums in the subtree have a nonempty intersection. Hence, read and write quorums have a nonempty intersection.

Second, when the read quorum does not include any subtrees. In this case, $w_r < (d - n_0)$, but w_r is greater than $d - w_w$. Combining the two inequalities we get $w_w > n_0$, indicating that the write operation cannot succeed. Hence, the theorem vacuously holds.

Case 2. Can be argued similarly.

Hence, by induction, the theorem follows. □

6.2 Analysis of Incomplete Trees

In this section, we conduct a cost and availability analysis similar to that in Section 5. However, the complexity increases due to the irregular structure of the subtrees involved in the recurrence equations. In particular for complete trees, all subtrees of a given node have the same structure which is not the case with incomplete trees. We consider an incomplete tree with a maximum degree d for each node. Furthermore, let n be the number of actual subtrees that are children of the root. Note that $n \leq d$. Let $MC_h^i[l, w]$ be the average cost of executing operations with a tree quorum of length l and width w in the ith subtree of the root with height $h + 1$. Thus, the cost $MC_{h+1}^{root}[l, w]$ for an incomplete tree of height $h + 1$ is:

$$MC_{h+1}^{root}[l, w] = \begin{cases} 1 & w < d - n \\ f * (1 + \text{average cost of forming } \langle l - 1, w \rangle \\ \quad \text{quorums in } w - (d - n) \text{ subtrees}) \\ +(1 - f) * \text{average cost of forming } \langle l, w \rangle \\ \quad \text{quorums in } w \text{ subtrees} & w \geq d - n. \end{cases}$$

The above equation can be expressed as follows:

$$MC_{h+1}^{root}[l, w] = \begin{cases} 1 & w < d - n \\ f * \left(1 + 1/\binom{n}{w - (d - n)} \sum_{|S| = w - (d-n)} \sum_{i \in S} MC_h^i[l - 1, w]\right) \\ +(1 - f) * 1/\binom{n}{w} \sum_{|S| = w} \sum_{i \in S} MC_h^i[l, w] & w \geq d - n. \end{cases} \quad (10)$$

In the above equation, a set S for a given size enumerates one possible selection of subtrees that can be chosen from the total number of subtrees. The average cost is computed by dividing the sum over all such S by the total number of selections. As in Section 5.1, $MC_l^i[0, w] = 0$ whereas $MC_l^i[l, w]$ needs to be computed individually and depends upon the structure of the ith tree. For example, if the tree is a root with a single child then $MC_2^i[2, 2]$ is one, assuming the maximum degree is three. Furthermore, if the number of children is two then the cost will be two, whereas when it is three it will be three. Note that Eq. 9 in Section 5.1 is a special case of the above equation in which all subtrees have equal costs and the tree is complete. In particular, the above summation reduces to the sum of the costs of forming quorums on the required number of children. Equation 10 computes the average cost for

the subtrees. However, it can be easily modified to compute the maximum and minimum costs attained by the protocol.

The availability analysis for incomplete trees is generalized in the same manner as described above. Let $A_h^i[l, w]$ be the availability of operations that require $\langle l, w \rangle$ sized tree quorums in the ith subtree of the root with height $h + 1$. Then, the availability $A_{h+1}^{root}[l, w]$ for an incomplete tree of height $h + 1$ is:

$$A_{h+1}^{root}[l, w]$$

$$= \begin{cases} \text{Probability(Root is up)} & w < d - n \\ \text{Probability(Root is up)} * \text{Availability of } \langle l - 1, w \rangle \\ \quad \text{quorums in } w - (d - n) \text{ subtrees} \\ + \text{Probability(Root is down)} * \text{Availability of } \langle l, w \rangle \\ \quad \text{quorum in } w \text{ subtrees} & w \geq d - n. \end{cases}$$

Let p be the probability that the root is available and $1 - p$ be the probability that the root is down, the above equation can be expressed as follows:

$$A_{h+1}^{root}[l, w] = \begin{cases} p & w < d - n \\ p * \left(\Sigma_{|S| \geq w-(d-n)} \Pi_{i \in S} A_h^i[l - 1, w]\right. \\ \left. * \Pi_{j \notin S}(1 - A_h^j[l - 1, w])\right) \\ + (1 - p) * \left(\Sigma_{|S| \geq w} \Pi_{i \in S} A_h^i[l, w]\right. \\ \left. * \Pi_{j \notin S}(1 - A_h^j[l, w])\right) & w \geq d - n. \end{cases} \tag{11}$$

Note that $A_h^i[l, w] = 0$ for $l > h$, $A_h^i[0, w] = 1$, and $A_1^i[1, w] = p$. The availability of quorums in the tree depends on the number of children the root has. We now use the above equations to compute the average cost and availability of the incomplete tree illustrated in Figure 10. The analysis is performed using the majority tree protocol, i.e., the tree quorums have size $\langle 2, 2 \rangle$. By using the recurrence relation in Equation 10, the average cost is:

$$MC_3^1[2, 2] = 5 - 2f - f^2$$

where f is the fraction of quorums that include the root of the tree/subtree. Hence, when $f = 0$, the cost is 5 nodes which is the same as the cost always needed by the majority voting protocol and when $f = 1$, the cost is as small as 2 nodes. Thus, the realistic estimate for the cost, when $f = 0.5$, is about 3.75.

Next, we use Eq. 11 to compute the availability of majority tree quorums for the same tree. By expanding the recurrence relation we obtain the availability of tree quorums as a function of p, where p is the probability that a site is available:

$$A_3^1[2, 2] = 2p^7 - 7p^6 + 11p^5 - 10p^4 + 3p^3 + 2p^2.$$

Figure 12 illustrates the availability of quorums in both our protocol and in the majority voting protocol. In this example, since there is an even number

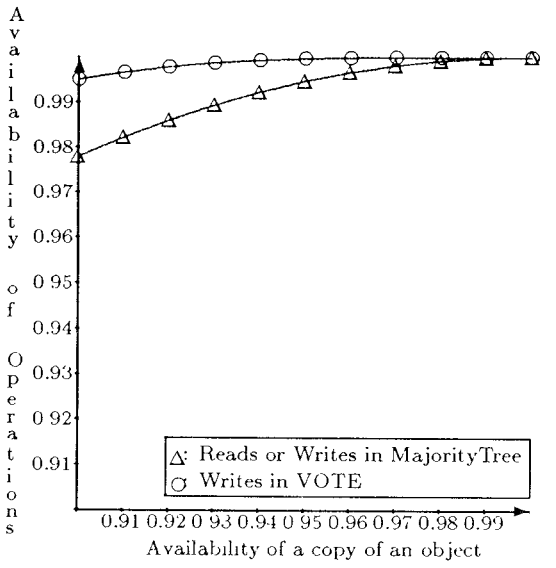

Fig. 12. Availability of an incomplete tree with 8 copies

of copies, we are only comparing the write availability in the majority voting protocol. As was the case for complete trees, the availabilities attained by both protocols are comparable for $p \geq 0.9$, while the costs are significantly less in the tree quorum protocol even with incomplete trees. It is interesting to note that when $p < 0.69$, the availability attained by the tree protocol is always better than the majority voting protocol.

6.3 Structuring Incomplete Trees

In this section, we explore the issue of structuring a logical tree on a set of copies. Since we are admitting the possibility of incomplete trees, the number of options that are available to the database designer increase immensely. These options include determining the number of copies that the database should provide, the overall availability of the object, and the cost of executing operations. As described in the previous section, the operation availability and cost are a function of the structure of the logical tree. Given a number of copies and a desired maximum degree of the tree, there may be several alternative tree structures that provide varying degrees of availability and cost. An appropriate tree can be selected by computing the availability and cost for all alternative structures.

We illustrate this approach by considering different ternary trees with the overall goal of maximizing write availability. We consider an object with eight copies. Some of the possible logical organizations of these copies are illustrated in Figure 13. For this example, we restrict ourselves to trees with both height and maximum degree three. By using Eq. 11, the write availability (when using majority tree protocol) for the four trees in the figure can be

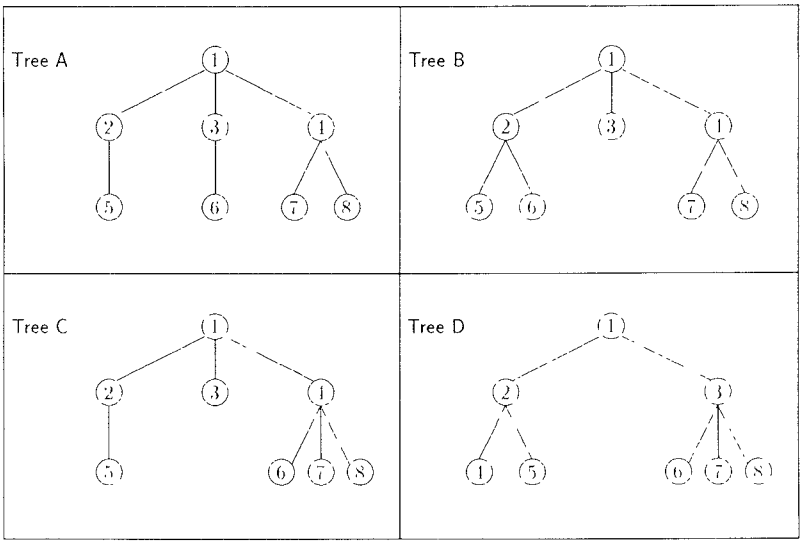

Fig. 13. Various configuration of trees with 8 copies.

expressed as follows:

$$\textit{WriteAvailability}(\text{Tree A}) = 4p^5 - 10p^4 + 6p^3 + p^2,$$
$$\textit{WriteAvailability}(\text{Tree B}) = -6p^7 + 21p^6 - 21p^5 + 7p^3,$$
$$\textit{WriteAvailability}(\text{Tree C}) = -8p^7 + 28p^6 - 32p^5 + 10p^4 + 2p^3 + p^2,$$
$$\textit{WriteAvailability}(\text{Tree D}) = 2p^7 - 7p^6 + 11p^5 - 10p^4 + 3p^3 + 2p^2.$$

Figure 14 illustrates the write availability for the above trees in the meaningful range of p, i.e., $p > 0.5$, we find that tree B has the best overall write availability. Thus, if write availability is the criterion for choosing the tree structure then structure B is most appropriate. Furthermore when availability is considered, if a node has fewer children than the majority (for the majority tree protocol), the children nodes do not contribute towards the availability. In tree A, for example, the presence of nodes 5 and 6 does not effect the overall availability of operations. Hence, the above heuristic can be used to eliminate logical structures such as trees A and C. A similar comparison can be carried out for computing the average, minimum, and maximum costs of executing operations on the trees illustrated in Figure 13. The final decision of choosing a structure would depend upon the factors p, the probability that a site is available, and f, the fraction of operations that access the root of the tree/subtree, and the desired availability and costs.

7. RELATED WORK

In this section we describe different replica control protocols and compare them to the proposed tree quorum protocol. The simplest replica control protocol is the read-one write-all protocol, where a read operation is executed by reading any copy and a write writes all copies of the object. This protocol is

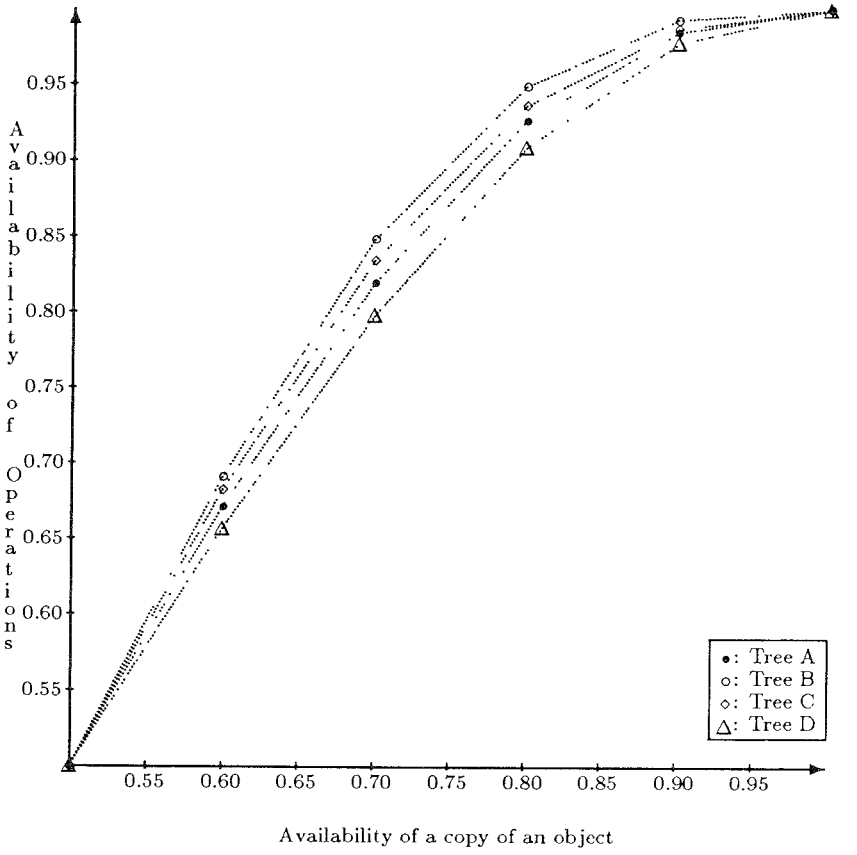

Fig. 14. Availability of incomplete trees A, B, C, and D.

a special instance of the tree quorum protocol when $q_r = \langle 1, 1 \rangle$ and $q_w = \langle h, d \rangle$. Of special interest is a comparison with the read root protocol. In a failure free system, read operations in both protocols access a single copy while write operations in the read root protocol do not write more than half the copies written by the read-one write-all protocol. When failures occur write operations can never be executed using the read-one write-all protocol. In the read root protocol write operations can be executed even after certain failures have occurred and the degree of write availability is significantly higher. Read operations may be required to access several copies, while attaining a comparable degree of availability in the presence of failures.

In order to increase the fault-tolerance of write operations, voting protocols were proposed, where write operations are not required to write all copies. In the static voting protocol [14, 27], a write operation writes Q_w copies, and a read operation accesses Q_r copies where $Q_r + Q_w$ is greater than the total number of copies of the object. The static voting protocol is a special instance of the tree protocol when all nodes of the tree have degree one. Thus, the tree

quorum protocol is a generalization of the static voting protocol with two degrees of freedom for choosing quorums. In general this results in lower costs for comparable data availability. In particular, we demonstrated that when the number of copies becomes large the majority voting and majority tree protocols achieve the same levels of availability. However, the majority tree protocol attains this level at a much lower cost. In the dynamic voting protocol [7, 18, 23], both read and write operations must only access a majority of the copies that were most recently updated. The read root and the log write protocols, in a failure free system, never require a read operation to access more than one copy. Furthermore write operations, in general, access less than a majority of copies. If the root is accessible, read operations always access a single copy and in the worst case access copies proportional to the number of leaves in the tree. Write operations, in the worst case, cannot be executed after the failure of the root, but can tolerate a large number of failures.

To overcome the problem of expensive read operations in the voting protocols, several algorithms have been proposed that use network configuration information [10, 11, 17]. This information is used to allow operations to adapt to changes in the network configuration. As a result read operations can always be executed by accessing a single copy. To ensure correctness, a special protocol must be executed whenever a new network configuration occurs. This protocol can be relatively costly since it involves communicating with multiple copies of several objects in the database. The tree quorum protocol achieves the advantages of reconfiguration protocols, i.e., low cost operation execution, while maintaining availability. However, it avoids the cost of reconfiguration by encoding the reconfiguration information in the logical tree structure. If failure patterns are more likely to occur in the levels close to the leaves, the tree quorum protocol is expected to perform better than reconfiguration-based protocols. On the other hand, if failure patterns are adverse to the tree structure, a reconfiguration-based approach will have better performance.

Finally, the notion of imposing logical structures on a network of sites has been proposed before to solve different problems. Maekawa [21] proposed imposing a logical grid on a set of sites to derive efficient $O(\sqrt{n})$ solutions for mutual exclusion. Agrawal and El Abbadi [1] proposed imposing a logical tree to develop a fault-tolerant distribution mutual exclusion algorithm which requires $\log n$ messages in a failure-free environment. This approach was extended to replica control protocols that use several logical structures imposed on a set of copies [2]. Kumar [19] constructs a logical tree on a set of copies, where the copies actually correspond to the leaves of the tree. This results in a protocol where read and write quorums are of size $2^{\log_3 n}$. Our protocol draws on many of these ideas, and extends them to develop an efficient and fault-tolerant replica control protocol. The distinguishing feature of our approach is that we directly address the issue of low cost operations and the quorums sizes dynamically adapt to the changes in the network configuration.

8. CONCLUSIONS

In this paper we have proposed a fault-tolerant protocol for managing replicated data. The design of the protocol directly addresses one of the main problems of replicated data: low cost read and write operations while providing high data availability. Our approach imposes a logical tree structure on the set of copies implementing a logical object. This relaxes the traditional restriction that a majority of copies must be accessed to attain maximum availability. The majority tree protocol achieves comparable level of data availability as majority voting at substantially lower costs. One important aspect of the tree quorum protocol is that it uses the tree structure to dynamically adapt to failures without requiring special reconfiguration protocols. In that sense, the structure encodes reconfiguration information. Avoidance of reconfiguration is especially attractive in systems where failures may be frequent, and where the size of the database is large and geographically dispersed. The logical tree structure also provides the database designer with more degrees of freedom in determining the size of the quorums. In particular, each quorum is determined by both its length and width, and changing these variables results in different instances of the proposed replica control protocol. Of special interest is the read root protocol, which requires read operations to read a single copy in a failure-free system, and the log write protocol, which has logarithmic cost for write operations while read operations access a single copy in the absence of failures. Finally, the proposed protocol exhibits the property of graceful degradation [22], which is especially attractive in distributed systems that may suffer from failures. In a failure-free system, the costs imposed by the tree quorum protocol are minimal, and as failures occur the cost may increase. This approach in designing distributed systems is desirable since it provides fault-tolerance without imposing unnecessary costs on the failure-free mode of operations.

REFERENCES

1. AGRAWAL, D., AND EL ABBADI, A. An efficient solution to the distributed mutual exclusion problem. In *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing* (Edmonton, Alberta, Canada, Aug. 1989), pp. 193–200.
2. AGRAWAL, D., AND EL ABBADI, A. Exploiting logical structures of replicated databases. *Inf. Process. Lett. 33*, 5 (Jan. 1990), 255–260.
3. AGRAWAL, D., AND EL ABBADI, A. Locks with constrained sharing. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems* (Nashville, Tenn. Apr. 1990), pp. 85–93.
4. AGRAWAL, D., AND EL ABBADI, A. The tree quorum protocol: An efficient approach for managing replicated data. In *Proceedings of Sixteenth International Conference on Very Large Data Bases* (Brisbane, Australia, Aug. 1990), pp. 243–254.
5. AHAMAD, M., AND AMMAR, M. H. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Trans. Softw. Eng. 15*, 4 (Apr. 1989), 492–495.
6. BERNSTEIN, P. A., AND GOODMAN, N. A proof technique for concurrency control and recovery algorithms for replicated databases. In *Distributed Comput. 2*, 1 (Jan. 1987), 32–44.
7. DAVCEV, D., AND BURKHARD, W. Consistency and recovery control for replicated files. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (Orcas Island, Wash., Dec. 1985), pp. 87–96.

8. DAVIDSON, S. B., GARCIA-MOLINA, H., AND SKEEN, D. Consistency in partitioned networks. *ACM Comput. Surv. 17*, 3 (Sept. 1985), 341–370.
9. EAGER, D., AND SEVCIK, K. Achieving robustness in distributed database systems. *ACM Trans. Database Syst. 8*, 3 (Sept. 1983), 354–381.
10. EL ABBADI, A., SKEEN, D., AND CRISTIAN, F. An efficient fault-tolerant protocol for replicated data management. In *Proceedings of the Fourth ACM Symposium on Principles of Database Systems* (Portland, Ore., Mar. 1985), pp. 215–228.
11. EL ABBADI, A., AND TOUEG, S. Maintaining availability in partitioned replicated databases. *ACM Trans. Database Syst. 14* (2) (June 1989), 264–290.
12. ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. The notion of consistency and predicate locks in database system. *Commun. ACM 19*, 11 (Nov. 1976), 624–633.
13. GARCIA-MOLINA, H., AND BARBARA, D. How to assign votes in a distributed system. *J. ACM 32*, 4 (Oct. 1985), 841–860.
14. GIFFORD, D. K. Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles* (Pacific Grove, Calif., Dec. 1979), pp. 150–159.
15. GRAY, J. N. Notes on database systems. In *Operating Systems: An Advanced Course*, R. Bayer, R. M. Graham, and G. Seegmuller, Ed. vol. 60, *Lecture Notes in Computer Science*, Springer-Verlag, 1978, pp. 393–481.
16. HERLIHY, M. A quorum-consensus replication method for abstract data types. *ACM Trans. Comput. Syst. 4*, 1 (Feb. 1986), 32–53.
17. HERLIHY, M. Dynamic quorum adjustments for partitioned data. *ACM Trans. Database Syst. 12*, 2 (June 1987), 170–194.
18. JAJODIA, S., AND MUTCHLER, D. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Database Syst. 15*, 2 (June 1990), 230–280.
19. KUMAR, A. Performance analysis of a hierarchical quorum consensus algorithm for replicated objects. In *Proceedings of the Tenth International Conference on Distributed Computing Systems* (Paris, May 1990), pp. 320–327.
20. KUNG, H. T., AND ROBINSON, J. T. On optimistic methods for concurrency control. *ACM Trans. Database Syst. 6*, 2 (June 1981), 213–226.
21. MAEKAWA, M. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst. 3*, 2 (May 1985), 145–159.
22. MAHANEY, S. R., AND SCHNEIDER, F. B. Inexact agreement: Accuracy, precision, and graceful degradation. In *Proceedings of the Fourth ACM Symposium on Principles of Distributed Computing* (Minaki, Ontario, Canada, Aug. 1985), pp. 237–249.
23. PÂRIS, J. F., AND LONG, D. E. Efficient dynamic voting algorithms. In *Proceedings of the Fourth IEEE International Conference on Data Engineering* (Los Angeles, Feb. 1988), pp. 268–275.
24. REED, D. P. Naming and synchronization in a decentralized computer system. Tech. Rep. MIT-LCS-TR-205, Cambridge, MA, Sept. 1978.
25. SCHLICHTING, R., AND SCHNEIDER, F. B. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst. 1*, 3 (Aug. 1982), 222–238.
26. SILBERSCHATZ, A., AND KEDEM, Z. Consistency in hierarchical databases systems. *J. ACM 27*, 1 (Jan. 1980), 72–80.
27. THOMAS, R. H. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst., 4*, 2 (June 1979), 180–209.

Received October 1990; revised February 1991; accepted April 1991.