An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion

DIVYAKANT AGRAWAL and AMR EL ABBADI
University of California

In this paper, we present an efficient and fault-tolerant algorithm for generating quorums to solve the distributed mutual exclusion problem. The algorithm uses a logical tree organization of the network to generate tree quorums, which are logarithmic in the size of the network in the best case. Our approach is resilient to both site and communication failures, even when such failures lead to network partitioning. Furthermore, the algorithm exhibits a property of graceful degradation, i.e., it requires more messages only as the number of failures increase in the network. We describe how tree quorums can be used for various distributed applications for providing mutually exclusive access to a distributed resource, managing replicated objects, and atomically committing a distributed transaction.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.1 [**Operating Systems**]: Process Management—*mutual exclusion*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Coteries, logical structures, quorums

1. INTRODUCTION

Mutual exclusion is crucial for the design of distributed systems. Many problems involving replicated data, atomic commitment, distributed shared memory, and others require that a resource be allocated to a single process at a time. Solutions to these problems are often vulnerable to site and communication failures. Intersecting quorums can be used to provide fault-tolerant solutions to mutual exclusion problems. However, they usually incur high communication costs. In this paper, we present a new quorum-based algorithm which has low communication cost and can handle both types of failures. In particular, this algorithm results in the first distributed mutual

This research was supported by the NSF under grant numbers CCR-8809387 and IRI-8809284 and by the University of California and IBM Yorktown Heights under grant numbers MICRO 88-179 and 89-137.

Authors' address: Department of Computer Science, University of California, Santa Barbara, CA 93106.

exclusion protocol to tolerate both site failures and network partitioning while in the best case incurring logarithmic costs in the size of the network.

Several algorithms exist to implement mutual exclusion in a distributed environment [3, 10, 25, 7, 18, 12, 23, 6, 26, 19, 17]. The primary site approach [3] requires low communication costs but is highly vulnerable to the failure of the primary site. Lamport uses logical timestamps [10] to implement distributed mutual exclusion. This protocol prevents starvation at the expense of requiring a site requesting mutually exclusive access to the resource to communicate with all other sites. This makes the protocol fairly expensive and not resilient to site and communication failures. Schneider [20] proposed a reliable broadcast based approach for synchronization in distributed systems. The broadcast mechanism is used to implement distributed semaphores, which can be employed to achieve mutual exclusion in distributed systems. Reliable broadcasts, however, are expensive and are not generally available in distributed systems. The majority quorum algorithm [25, 7] is a very simple and elegant scheme to achieve mutual exclusion in a distributed system. In order to attain mutual exclusion, a site must obtain permission from a majority of sites in the network. Since there can be only one majority at any instant, mutual exclusion is achieved easily. The majority quorum algorithm is robust and is resilient to both site and communication failures. This algorithm has been particularly used in replicated databases and to solve the distributed commit problem in systems with site failures and network partitions.

In the above protocols, no assumption is made about the logical or physical organization of the network. The only assumption required is that any two sites in the network can communicate with each other when there are no failures. Maekawa [12] proposes implementing distributed mutual exclusion by imposing a *logical structure* on the network. In this scheme, a set of sites is associated with each site, and this set has a nonempty intersection with all sets corresponding to the other sites. The rule for constructing these sets is based on the structure of *finite projective planes*. A process must obtain permission from all sites in the set associated with its home site before it can achieve mutual exclusion. Since the set intersects with every other set of other sites, mutual exclusion is guaranteed. The interesting aspect of Maekawa's solution is that the size of each of these sets is $\sqrt{n}$, where n is the number of sites in the network. Hence, a process needs to communicate with only $\sqrt{n}$ sites to obtain permission for mutual exclusion. This is in contrast to the majority quorum algorithm which involves communication with $\lceil (n + 1)/2 \rceil$ sites. Thus, imposing a logical structure on the network significantly reduces the overhead of achieving mutual exclusion.

Suzuki and Kasami [23] present a token-based algorithm for distributed mutual exclusion, which requires at most n messages. Helary, Plouzeau, and Raynal [9] propose a token-based algorithm that uses a flooding broadcast technique to locate the token. Raymond [17] proposes another token-based algorithm that uses a spanning tree of the network for locating the token,

and shows that the average number of messages exchanged in this protocol is $O(\log n)$. In the best case no communication is necessary since the token may be available locally while in the worst case the number of messages is proportional to the diameter of the network. A similar token-based scheme for a tree of processes or sites has also been proposed by van de Snepscheut [26]. This protocol guarantees fairness and has similar costs as Raymond's algorithm. Token-based algorithms suffer from poor failure resiliency. In particular, if the site holding the token fails, complex token regeneration protocols must be executed [14]. Furthermore, when this site recovers, it has to undergo a recovery phase during which it is informed that its token has been invalidated. For example, the protocol proposed by Raymond cannot recover from the failure of the site holding the token and all its neighbors. The above algorithms use static trees. Several protocols have been proposed to dynamically restructure the tree [16, 4] resulting in a logarithmic bound on the average number of messages needed to achieve mutual exclusion.

The concept of intersecting quorums captures the essence of mutual exclusion in distributed systems. Garcia-Molina and Barbara [6] have proposed the notion of a *coterie*, which generalizes the notion of quorums. A coterie is a set of sets with the property that any two members of a coterie have a nonempty intersection. Maekawa's solution can be considered as a special instance of a coterie where each member is of equal size, i.e., $\sqrt{n}$ sites. A coterie, on the other hand, permits the individual sets (its members) to be of unequal sizes. In this paper we combine the idea of logical structures and the notion of coteries to develop an efficient and fault-tolerant protocol for mutual exclusion in distributed systems.

We assume that the network is *logically* organized into a tree. We then provide a rule for constructing a coterie and prove that any two members of the coterie have a nonempty intersection. In Maekawa's protocol, only one set is associated with each site. This makes the protocol nontolerant to failures since the failure of any site in the associated set of a site prevents that site from achieving mutual exclusion. Our scheme ensures fault-tolerance by providing several alternative sets to a site requesting mutual exclusion. We show that in the best case our scheme requires permission from only $\lceil \log\ n \rceil$ sites, and in the worst case it requires $\lceil (n + 1)/2 \rceil$ sites. Note that the majority quorum algorithm [25, 7] would always require communication with $\lceil (n + 1)/2 \rceil$ sites. We use this approach to solve the problem of mutually exclusive access to a resource in a distributed system, to manage replicated objects, and to commit distributed transactions. The mutual exclusion protocol described in this paper is the first protocol to tolerate both site failures and network partitioning and still needs only $O(\log n)$ messages in the best case.

In the next section we present the model of the system. In Section 3, we present the tree quorum algorithm for constructing quorums by using a logical organization of a network that is a binary tree. An analysis of both the cost and availability aspects of the tree quorum algorithm is performed in

Section 4. In Section 5, we use the tree quorum algorithm to develop a distributed mutual exclusion protocol and use it for other distributed applications. Section 6 generalizes the tree quorum algorithm to arbitrary trees. We conclude the paper 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 assumption is made about the underlying topology of the network nor is there any assumption about the existence of a multicast facility. A multicast facility, however, can be exploited to improve the performance of the system. We assume that a routing mechanism exists that delivers messages between sites. We also assume that the sites can be logically organized to form a structure such as a binary tree, a tree, a grid, etc.

Sites may either crash or may fail to send or receive messages. Communication links may fail by crashing, or by failing to deliver messages. Combinations of such failures may lead to *partitioning failures* [5], where sites in a *partition* may communicate with each other, but no communication can occur between sites in different partitions.

We now formalize the notion of coteries [6]. A *coterie*, C, is a set of sets where each set g in C is called a *quorum*. The following conditions hold for the quorums in a coterie C:

The Intersection Property. If g and h are quorums in C, then g and h must have a nonempty intersection, i.e., $g \cap h \neq \phi$.

The Minimality Property. There are no two quorums g and h in C such that g is a superset of h.

Coteries can be used to develop protocols that guarantee mutual exclusion in a distributed system. For example, to obtain mutually exclusive access to a resource in the network, a process at site s_i is required to receive permission from some quorum of sites S_i in the network. If all sites in S_i grant permission, the process is allowed to access the resource. Since any pair of quorums have at least one site in common, mutual exclusion can be guaranteed. Note, however, additional mechanisms must be used to eliminate the problems of deadlock and starvation [19]. The minimality property is not necessary for correctness but is useful for efficiency.

3. THE ALGORITHM FOR CONSTRUCTING TREE QUORUMS

The standard approach for implementing coteries is that any set containing a majority of sites forms a quorum (a variation of this approach is to associate with each site a *vote*, and a set of sites form a quorum if the sum of their votes is greater than or equal to a majority of all votes). We now develop an alternative approach, which imposes a logical structure on the sites in the

system. The structure we propose is a binary tree, and we devise an algorithm, which uses this specific tree structure to determine the set of sites that constitute a quorum.

3.1 Quorum Construction in a Binary Tree

Given a set of n sites, we assume that the sites are logically organized to form a binary tree. We will assume the standard tree terminology, i.e., root, child, parent, leaf, etc. The algorithm for constructing quorums can be used with arbitrary trees, however, for simplicity and efficiency we assume that the tree is complete, i.e., if k is the level of the tree then it has $2^{k+1} - 1$ sites and the root is at level k and the leaves are all at level 0. For the purpose of this presentation, any site could be chosen as the root, and any two sites may be chosen as its children, and so on. A *path* in the tree is a sequence of sites $s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_n$, such that s_{i+1} is a child of s_i.

In Figure 1, we present the algorithm for constructing a valid quorum. We assume that the tree has a well defined root, and that a process at a site requesting a quorum calls the recursive function *GetQuorum* with the root of the tree as parameter. The function *GrantsPermission*(*site*) is true if *site* agrees to be a member of the quorum. For example, a site may not agree to be in the quorum on account of failures. The algorithm tries to construct a quorum by selecting any path starting from the root and ending with any of the leaves. If successful, this set of sites constitutes a quorum. If it fails to find a path as a result of the failure (or inaccessibility due to network partitioning) of a site, say s_i, then the algorithm must substitute for that site with two paths, both of which start with the children of site s_i and terminate with leaves. We note that each path must terminate with a leaf, hence if the last site in the path is inaccessible, a quorum cannot be formed and the algorithm terminates with an error condition. If no sites are inaccessible, then the quorum is any set $\{s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_n\}$, where s_1 is the root, s_n is a leaf, and for all $i < n$, s_{i+1} is a child of s_i. If failures occur, then for each failed s_i, the quorum contains (recursively) a path of sites starting from s_j and s_k, where s_j and s_k are the two children of s_i, and ending with leaves. The set constructed by this algorithm is termed as a *tree quorum*.

Consider a distributed system with seven sites. We superimpose a binary tree on the sites as illustrated in Figure 2, with the sites numbered as shown. Following the algorithm, the union of the following tree quorums constitutes a coterie. If no failures have occurred, then any of the following four sets form a quorum: $\{1, 2, 4\}$, $\{1, 2, 5\}$, $\{1, 3, 6\}$, and $\{1, 3, 7\}$. If the root is inaccessible (due to site failures or network partitioning), then the following four sets are quorums: $\{2, 4, 3, 6\}$, $\{2, 5, 3, 6\}$, $\{2, 4, 3, 7\}$, and $\{2, 5, 3, 7\}$. If site 2 or site 3 is down, then $\{1, 4, 5\}$ or $\{1, 6, 7\}$ respectively form quorums. If both sites 1 and 2 are down, then $\{4, 5, 3, 6\}$ and $\{4, 5, 3, 7\}$ are candidates for quorums. Similarly, if sites 1 and 3 are inaccessible, the sets $\{2, 4, 6, 7\}$ and $\{2, 5, 6, 7\}$ are quorums. Finally, if sites 1, 2, and 3 are inaccessible, then the only possible quorum is $\{4, 5, 6, 7\}$.

```
FUNCTION GetQuorum(Tree: TREE): QuorumSet;
  VAR
    left,right: QuorumSet;
  BEGIN
  IF Empty(Tree) THEN
    RETURN({ });
  ELSE IF GrantsPermission(Tree ↑ .Node) THEN
    RETURN({Tree ↑ Node} ∪ GetQuorum(Tree ↑ .LeftChild));
    OR
    RETURN({Tree ↑ .Node} ∪ GetQuorum(Tree ↑ .RightChild));
  ELSE
    left ← GetQuorum(Tree ↑ LeftChild);
    right ← GetQuorum(Tree ↑ RightChild);
    IF (left = φ ∨ right = φ) THEN
      (* Unsuccessful in establishing a quorum *)
      EXIT(error);
    ELSE
      RETURN(left ∪ right);
    END; (* IF *)
  END; (* IF *)
END GetQuorum.
```

Fig. 1 The algorithm for constructing a tree quorum

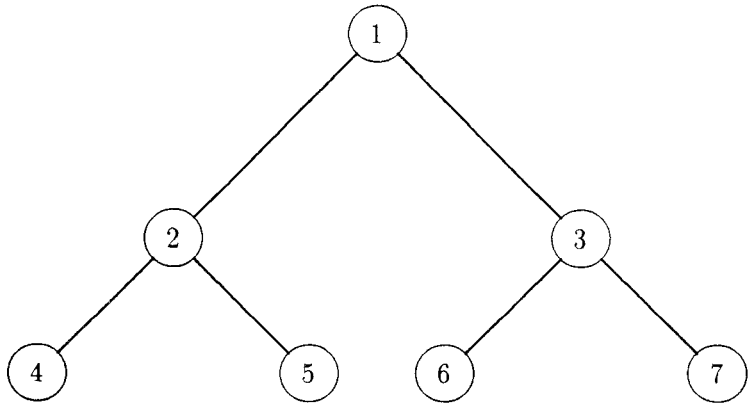

Fig. 2. A sample hierarchy of 7 site network

3.2 Correctness

We now demonstrate that the tree quorums constructed by the proposed algorithm guarantee the intersection and the minimality properties of coteries. The following theorem establishes the correctness of the tree quorum algorithm proposed above.

THEOREM 1. *Tree quorums satisfy the intersection and the minimality properties of coteries.*

PROOF. The proof is by induction on the levels of binary trees.

Basis. Consider a binary tree of level 0, which implies that there is one site s_1 in the network. The set of tree quorums will be $\{\{s_1\}\}$. The intersection and the minimality properties hold trivially for the basis case.

Induction Hypothesis. Assume that the theorem holds for binary trees of level k.

Induction Step. Consider a binary tree of level $k + 1$. Without loss of generality, assume that s_1 is the root of this tree. Now consider the three subtrees of this tree: the subtree consisting of the root, the left subtree, and the right subtree. It can be verified from the tree quorum algorithm in Figure 1 that any tree quorum chosen in this tree will be from one of the following classes:

(1) $\{s_1\} \cup$ {members from the quorum set of the left subtree},
(2) $\{s_1\} \cup$ {members from the quorum set of the right subtree}, or
(3) {members from the quorum set of the left subtree} $\cup$ {members from the quorum set of the right subtree}.

It can be easily verified that the members of class 1 will have a nonempty intersection with classes 1, 2, and 3. Also, members of class 1 are not contained in classes 2 and 3. Similar conditions hold for classes 2 and 3. Thus, the quorum set constructed for a tree of level $k + 1$ is a coterie.

Hence, by induction, tree quorums constructed for a network organized as a binary tree of arbitrary level will satisfy the intersection and the minimality properties. □

3.3 Discussion

In the best case, only $\lceil \log n \rceil$ sites are necessary to form a tree quorum. This case is achieved both when there are no failures, and for certain patterns of failures, e.g., when a site at the level above the leaves fails, a quorum of size $\lceil \log n \rceil$ is still possible. For example, in Figure 2 if site 2 is inaccessible then a quorum of size $\lceil \log n \rceil$ can still be formed with $\{1, 4, 5\}$. This means that in a relatively failure-free environment, the tree quorum algorithm requires fewer messages to form a quorum than any of the previous fault-tolerant protocols [12, 25, 7]. Furthermore, the algorithm can also tolerate the failure of up to $n - \lceil \log n \rceil$ specific sites, and still form a tree quorum. In the above example, quorum $\{1, 4, 5\}$ can be formed if in addition to site 2, sites 3, 6, and 7 have failed.

In the worst case, a majority of sites is necessary for constructing a tree quorum, which is the same as the number of sites required by the quorum algorithm in all cases. This is proved in the following theorem.

THEOREM 2. *The worst-case tree quorum size is* $\lceil (n + 1)/2 \rceil$.

PROOF. The proof is by induction on the levels of binary trees.

Basis. Consider a binary tree of level 0, i.e., a tree consisting of a single site. The size of the quorum is one and therefore the theorem holds.

Induction Hypothesis. Assume that the theorem holds for binary trees of level k, i.e., the worst case quorum size is 2^k since there are $2^{k+1} - 1$ sites in such a tree.

Induction Step. Consider a binary tree of level $k + 1$. This tree consists of a root and left and right subtrees each of level k. The largest quorum in such a tree will occur when the root is down and the largest quorums are chosen from the left and right subtrees. From the induction hypothesis, the worst case size of the quorum is $2^k + 2^k$ or 2^{k+1}, which is equal to $\lceil (n + 1)/2 \rceil$. Note that $n = 2^{k+2} - 1$. □

Note that the algorithm may not be able to form a tree quorum in some cases after the failure of $\lceil \log n \rceil$ sites, e.g., if sites 1, 2, and 4 are inaccessible in Figure 2, the set of sites $\{3, 5, 6, 7\}$, which contains a majority of sites, do not form a tree quorum. Those sites would form a quorum in the majority quorum algorithm. However, in the complementary situation when sites 3, 5, 6, and 7 are inaccessible, our approach can form a quorum, the set $\{1, 2, 4\}$, while the majority quorum algorithm would not succeed.

The tree quorum algorithm exhibits the useful property of *graceful degradation* [13], which is desirable in distributed fault-tolerant systems. In a failure-free environment, the algorithm guarantees low communication costs: only $\lceil \log n \rceil$ sites are necessary to form a quorum. As failures occur, and increase, the cost of forming a quorum may increase, and the probability of forming a quorum decreases. For example in a tree of level k, if a failure is detected at level $i > 0$ while constructing a quorum, the quorum size increases from $k + 1$ to $(k - i) + 2i$. This is because when a node fails instead of one path from the node two paths starting from the node's children must be included in the quorum. Thus, the penalty for failures closer to the root is more severe than the failures in the vicinity of the leaves. Note however, that the tree quorum algorithm always guarantees the formation of a quorum as long as the number of failures is less than $\lceil \log n \rceil$ sites, and may still allow some quorums to be formed even after the failure of $n - \lceil \log n \rceil$ sites.

4. ANALYSIS OF THE TREE QUORUM ALGORITHM

In this section, we analyze two important aspects of the tree quorum algorithm: the cost and availability of forming tree quorums. We first compute the expected number of sites that are needed to obtain a tree quorum and a majority quorum. We then compute the probability of acquiring a tree quorum when there are n sites in the system and compare the availabilities of forming a tree quorum and a majority quorum. Finally, we compare the tree quorum availability with other simple quorum based approaches.

4.1 Message Cost

The number of messages needed to construct a quorum is directly proportional to the size of the quorums. In the majority quorum algorithm, the quorum size corresponding to a majority is

$$\left\lceil \frac{n + 1}{2} \right\rceil.$$

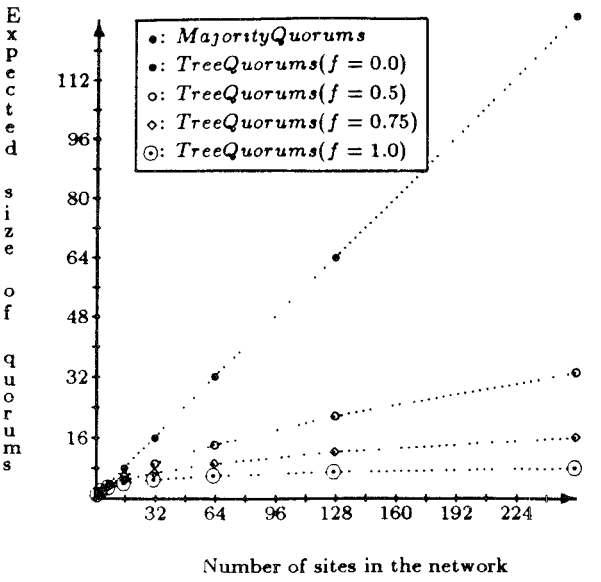

Fig. 3. Expected message cost of the two algorithms.

In the case of the tree quorum algorithm, the size of the tree quorums vary from $\lceil \log n \rceil$ to $\lceil (n+1)/2 \rceil$. The message cost can be computed by using a recurrence relation that calculates the cost of obtaining a quorum in a tree of level $l+1$ in terms of subtrees of level l. Unlike the majority quorum algorithm where all quorums are of equal size, the tree quorums are of varying sizes. We introduce a parameter f that indicates the fraction of quorums that include the root of a tree of level $l+1$. Hence, $1-f$ is the fraction of quorums that do not include the root. Let C_l be the average cost of forming a quorum in a tree of level l. Thus, the cost C_{l+1} for a tree of level $l+1$ is

$$C_{l+1} = f(C_l + 1) + (1-f)(2C_l).$$

The first term arises because the root is included in f quorums while the second term occurs because there are $1-f$ quorums with size $2C_l$. Note that C_0 is equal to one.

Figure 3 illustrates the cost of obtaining a quorum in the two algorithms for varying sizes of the network. In the case of the tree quorum algorithm, we illustrate four curves for different values of f. The extreme values of f being 0 and 1 illustrate the upper and lower bounds of the message cost in this algorithm. We first note that in the worst case, when $f = 0$, the tree quorum algorithm has the same message cost as the majority quorum algorithm. The case with $f = 0$ corresponds to the situation where a larger quorum is always chosen instead of the smaller one. In particular, this will occur when the root of a given subtree is not available. In most distributed systems the probability of a site being available will most likely be greater than 50 percent. In

that case, the curve corresponding to $f = 0.5$ represents a realistic upper bound on the cost of forming a tree quorum. The region limited by the curves $f = 1$ and $f = 0.5$ is significantly cost efficient when compared to the majority quorum algorithm. In particular, in a network of 127 sites, the expected size of the quorum in the majority quorum algorithm is 64, whereas the tree quorum algorithm will have size 7 in the best case and 22 when $f = 0.5$. If we increase the bias towards smaller quorums, for example $f = 0.75$, the message cost becomes approximately logarithmic.

4.2 Availability Analysis

Let p be the probability that a site is available at any time. Furthermore, assume that the number of sites in the system is $n = 2k + 1$ for some nonnegative integer k. We compute the probability of obtaining quorums in the majority quorum algorithm [7, 25] and the tree quorum algorithm. The *availability* of an algorithm is defined as the probability of forming a quorum successfully in that algorithm.

In the case of the majority quorum algorithm, a majority of sites can form a quorum only if a majority or more sites are available. Thus the availability of this algorithm is

Availability(majority) = Probability($k + 1$ sites are up)
+ ⋯
+ Probability($k + i$ sites are up)
+ ⋯
+ Probability($2k + 1$ sites are up).

The above probabilities are the binomial terms, i.e.,

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

Figure 4 illustrates the availability of the majority quorum algorithm for systems with different configurations.

The availability of the tree quorum algorithm can be computed by formulating a recurrence relation. The recurrence relation is in terms of the availabilities of forming quorums in the subtrees of a binary tree. Let A_l be the availability of forming a quorum in a tree of level l. Thus, the availability A_{l+1} of forming a quorum in a tree of level $l + 1$ is given as

Probability(root is up) * Availability(Left subtree)
* Unavailability(Right subtree)
+ Probability(root is up) * Unavailability(Left subtree)
* Availability(Right subtree)
+ Probability(root is up) * Availability(Left subtree)
* Availability(Right subtree)
+ Probability(root is down) * Availability(Left subtree)
* Availability(Right subtree).

Using p as the probability of the root being up, A_l as the availability of a subtree of level l, and $1 - A_l$ as the unavailability of a subtree of level l, we

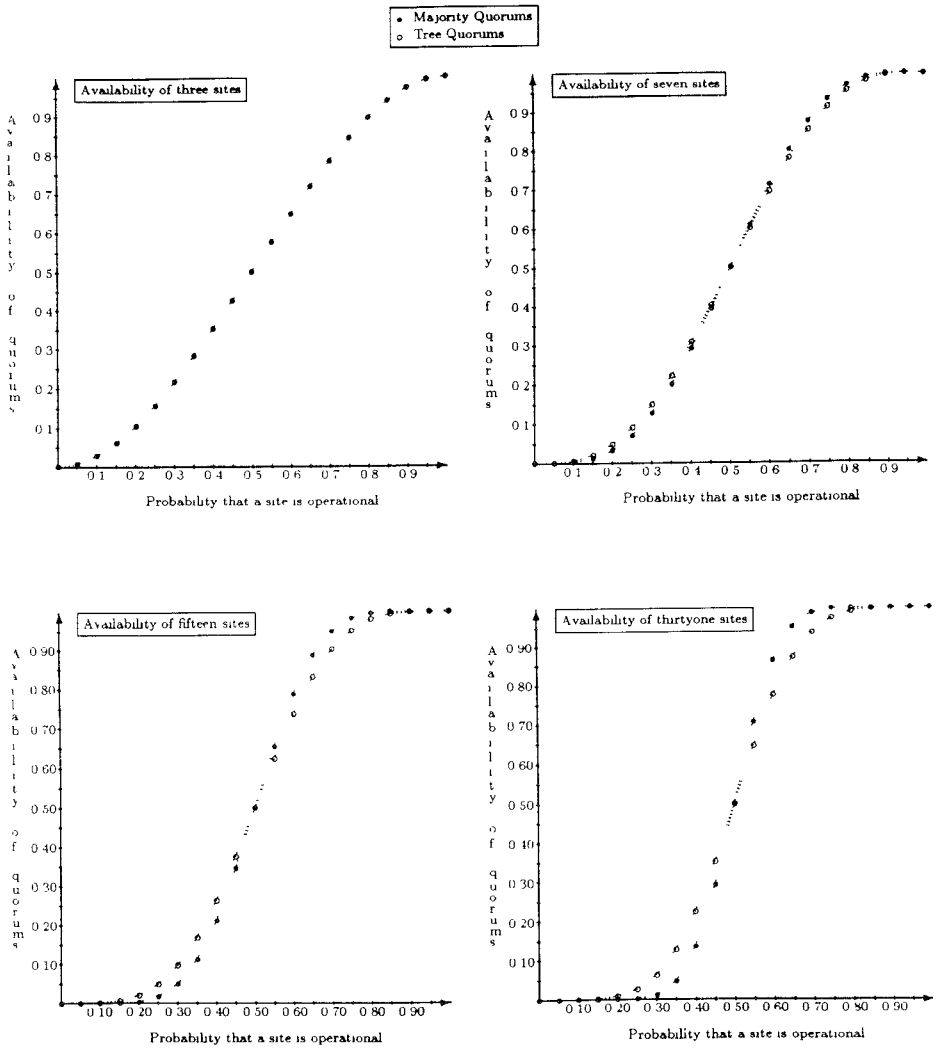

Fig. 4. Availability of majority and tree quorums.

can write the above expression as

$$A_{l+1} = pA_l(1 - A_l) + p(1 - A_l)A_l + pA_l^2 + (1 - p)A_l^2,$$

i.e.,

$$A_{l+1} = 2pA_l + (1 - 2p)A_l^2.$$

Note that the availability in a tree with a single site is $A_0 = p$. The above recurrence involves a nonlinear term and therefore we illustrate the availabilities of logical trees with various configurations in Figure 4. The availability graphs in Figure 4 show that, in general, the algorithms attain

comparable levels of availability. The availability of the tree quorum algorithm becomes inferior to the majority quorum algorithm for the values of p approximately in the range (0.5, 0.75). However, these values correspond to a network where a site is down 50 to 25 percent of the time. For $p > 0.75$, the availabilities attained by the two algorithms become indistinguishable.

Figures 3 and 4 illustrate that the tree quorum algorithm can achieve comparable degree of availability as the majority quorum algorithm but at substantially lower costs. In particular, in systems with site availability greater than 0.9, the tree quorum algorithm provides the same availability as the majority quorum algorithm at approximately logarithmic costs. Thus, the tree quorum algorithm significantly reduces the cost of forming a quorum in a large network without sacrificing availability.

4.3 Comparison with Other Approaches

In the previous subsections, we compared the tree quorum algorithm primarily with the majority quorum algorithm [7, 25]. In this section, we compare the availability of the tree quorum algorithm with some other simple and straightforward approaches for constructing quorums.

We consider an approach in which a set of sites are designated as *distinguished* sites. We can choose $2f(n) + 1$ distinguished sites among n sites where the choice of $f(n)$ is based on several factors such as communication costs and availability. We will consider two specific instances of this approach to construct quorums in the system. The first approach provides constant levels of availability and communication costs, which are independent of the number of sites in the system. In this approach, $2c + 1$ sites are chosen as distinguished sites among n sites such that any set of $c + 1$ distinguished sites form a quorum. For example, if c is three then seven sites are chosen as the distinguished sites and any site that needs to enter critical section must access at least four of the seven distinguished sites. Note that if $c = 0$ the above approach reduces to a primary site approach. Another variation we consider is when $f(n)$ is $\lceil \log n \rceil$ and $2\lceil \log n \rceil + 1$ sites are chosen as distinguished and any $\lceil \log n \rceil + 1$ distinguished sites form a quorum. This approach would guarantee communication costs identical to the best case of the tree quorum algorithm. However, the availability in the two approaches are different.

Figure 5 illustrates the availability of the tree quorum algorithm with respect to the two approaches discussed above. We refer to them as the $2c + 1$ and $2\lceil \log n \rceil + 1$ protocols. Note that c is chosen as three. The four graphs illustrate the availability in each of these algorithms for different site availabilities. In general, the availability of the tree quorum algorithm is higher for a large number of sites. However, the difference in availabilities becomes less significant between the tree quorum algorithm and the $2\lceil \log n \rceil + 1$ protocol when the site availability is high. Thus, the tree quorum algorithm will be useful for large networks and will provide a scalable solution since it has low communication overhead. On the other hand, the

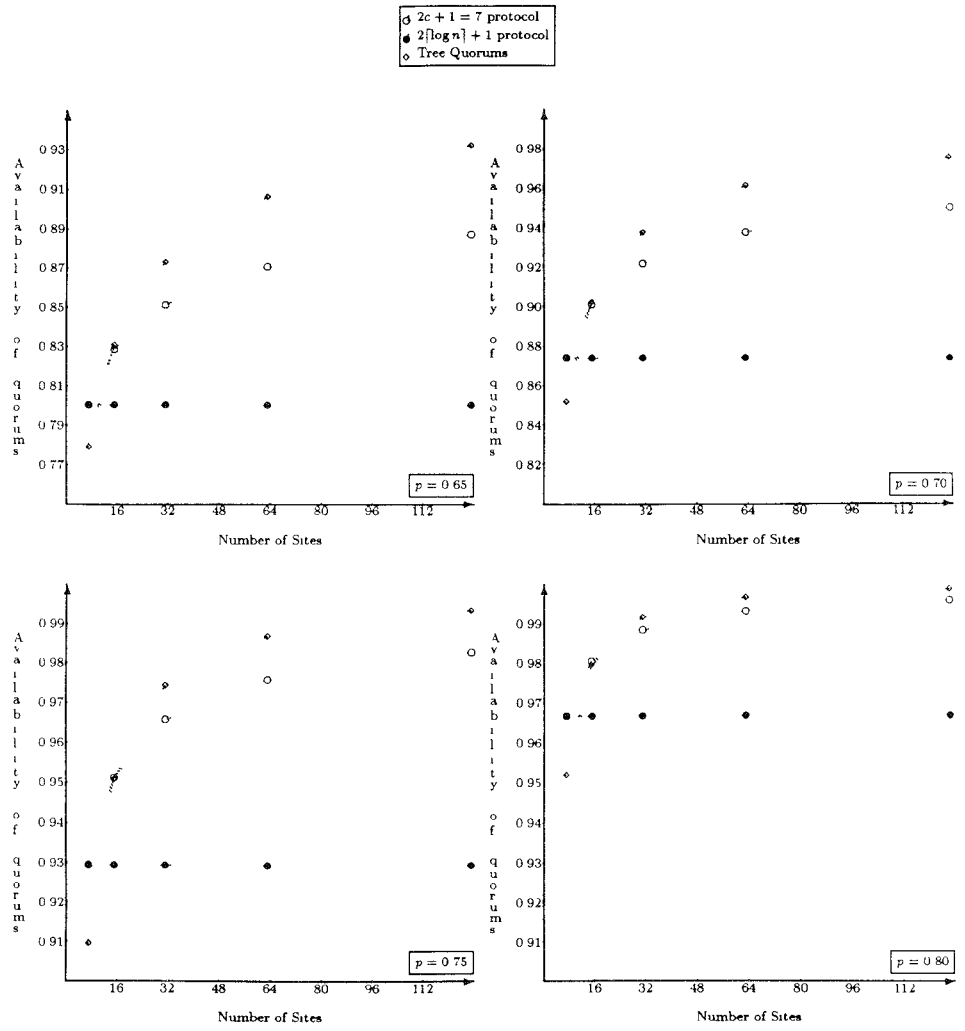

Fig. 5. Comparison with simple algorithms.

high availabilities attained by the simple algorithms demonstrate that they may be good candidates for certain applications. Note, however, that the choice of distinguished sites becomes crucial in attaining high levels of availability especially when network partitions are considered. The tree quorum algorithm requires no such choice.

5. APPLICATIONS

In this section, we start by describing how tree quorums can be used to achieve mutually exclusive access to a resource in a distributed system. We

then use tree quorums for other applications that need distributed mutual exclusion: replicated data management and atomic commitment in distributed databases.

5.1 Mutual Exclusion

In this subsection, we describe how a process can acquire mutually exclusive access to a resource. Our approach is similar to Maekawa [12] and Sanders [19] and is free from deadlocks and starvation. A process is allowed to access the resource only when executing in a critical section. Without loss of generality, we assume that there is only one process per site. As in the standard model for distributed mutual exclusion protocols, we assume first-in first-out message delivery between any pair of processes. To enter its critical section, a process at a site s, must send request messages, *Req*, to a quorum of sites, which is determined using the tree quorum algorithm. Each *Req* message is timestamped with a unique local timestamp (each site has a logical clock [10]). Each site maintains a *request queue*, where *Req* messages are ordered in timestamp order. When a *Req* message is at the head of the queue, the site sends a reply message, *Reply*, to the requesting site. A process requesting to enter its critical section waits until it receives *Reply* messages from a set of sites that form a tree quorum before entering its critical section. Once a process exits from its critical section, it sends relinquish messages, *Relinq*, to all sites in the quorum, so that they may remove the corresponding *Req* message from the head of the queue.

Whenever a new request arrives with a timestamp earlier than the request at the head of the queue, an *Inquire* message is sent to the process whose request was previously at the head of the queue and waits for either a *Yield* or a *Relinq* message. If a site r receives an *Inquire* message, it sends back a *Yield* message to the inquiring site if r has not yet collected enough replies from its quorum. If r had actually acquired all of its necessary replies to access the resource, then it simply ignores the inquire message, and proceeds normally, i.e., by accessing the resource and then sending a *Relinq* message. When a site receives a *Yield* message, it puts the pending request (on behalf of whom the *Inquire* message was sent) at the head of the queue and sends *Reply* message to the requestor. The proof of correctness and freedom from deadlocks and starvation follows from Maekawa [12] and Sanders [19].

5.2 Tree Quorums and Replication

We consider a distributed database, which is a collection of objects stored at different sites in a network. Each object in the database may be replicated and stored at several sites in the network. With each object x we associate a replication tree denoted *Tree*[x], which could be an extension of the name directory of objects. It lists the sites where copies of x are resident and stores the logical tree organization of these copies. We assume that each copy has associated with it a version number which is initialized to zero. In order to perform a read operation on x, the transaction initiating the operation must apply the algorithm of Figure 1 to *Tree*[x], and obtain a read quorum. The

copy read is the one which is associated with the highest version number. Write operations are carried out similarly and all copies in the tree quorum are updated with the new value and a version number greater than any previous version number in the quorum. The requirement that read and write operations overlap at least at one copy of x is guaranteed by the intersection property of the tree quorum algorithm. A further discussion of using logical structures to organize replicated data is presented by Agrawal and El Abbadi [1, 2].

5.3 Tree Quorums and The Commit Protocol

In distributed databases, atomic commitment is necessary to ensure consistent and fault-tolerant termination of distributed transactions. The *two phase commit* [8] and *three phase commit* [21] protocols are used to solve the atomic commitment problem in distributed databases. Blocking occurs when a site participating in the commitment of a transaction cannot terminate (commit or abort) the transaction due to failures. The three phase commit protocol was designed to overcome the problem of blocking, which is inherent in the two phase commit protocol when site failures occur. However, the three phase commit protocol may still block if failures occur that lead to network partitions. A termination protocol, which uses quorums, is executed to consistently terminate a transaction across partitions. During the execution of the commit protocol, sites participating in a transaction may be in one of the following four states: *commit*, *abort*, *committable* and *abortable*. After a partitioning failure, Skeen [22] proposes the execution of a *termination* protocol, which chooses a coordinator to determine the state of all participants in the partition. If there is a site with a committed (aborted) state, or a majority of sites with committable (abortable) states, the transaction is committed (aborted) in that partition. Otherwise, the transaction may still remain blocked. We now apply the tree quorum approach to the problem of committing such transactions. We assume that initially all participants of a transaction are logically ordered in a tree structure (note that this might be already the case for nested transactions [15, 11]). To commit, transactions execute the three phase commit protocol, and if a partitioning failure occurs a termination protocol is executed. The termination protocol requests the state of sites in a tree quorum instead of a majority of sites. As presented by Skeen [22], if a committed (aborted) state, or a tree quorum of committable (abortable) states is collected, the transaction can be committed (aborted), otherwise it is blocked. Thus, the termination protocol may terminate a transaction with as few as $\lceil \log n \rceil$ sites when tree quorums are used.

6. EXTENSIONS TO THE BINARY TREE QUORUM ALGORITHM

We now generalize the tree structure from binary trees to trees where each node has degree d. The algorithm is extended in a straightforward way. Any set of sites, which contains a path from the root to a leaf forms a tree quorum. Furthermore, whenever a node is inaccessible, the set must contain paths starting from all d children to the leaves. This generalization reduces the

size of a tree quorum in the best case to $\lceil \log_d n \rceil$, and hence may tolerate the failure of up to $n - \lceil \log_d n \rceil$ sites. However, in the worst case, as many as $\lceil [(d-1)n + 1]/d \rceil$ sites may be necessary to form a quorum.

If a bound is known on the number of failures in the system then we can derive a logical tree structure that will always guarantee the successful formation of a quorum. More specifically, if the maximum number of possible failures is t, then the tree must be of level t. Any tree of level fewer than t will be unable to provide a quorum if all sites in a path from the root to a leaf have failed. The degree d of such a tree is computed from the total number of sites n in the network and t by using the following expression:

$$\left\lfloor \frac{d^{t+1} - 1}{d - 1} \right\rfloor = n.$$

Such an approach guarantees the formation of a quorum as long as there are less than t failures. However, as was described in the previous section, there are cases where a quorum can be formed in spite of more than t failures. The best case quorum size is $t + 1$ in the above tree while the worst case quorum size occurs when all t failures are near the root of the tree. Let t failures be such that a complete subtree starting from the root is of level $l < t$ is inaccessible, where l is the smallest integer such that $t \leq (d^{l+1} - 1)/(d - 1)$. The size of the quorum in this case is $d^l(t - l + 1)$. The worst case quorum size is illustrated in Figure 6.

An optimization to the algorithm of Figure 1 is that the site initiating the request to form a tree quorum be always included in the final set. This can be achieved by ensuring that whenever there is a choice of a subtree, pick the one that contains the requesting site. If none of the subtrees contain the requesting site then the choice should be made randomly. The random choice helps in distributing the load evenly in the network. Another extension we have made is that the trees are not required to be complete. A site in the tree organization can have arbitrary number of children. Tree quorums can be formed for arbitrary trees as long as a failed site is substituted by all possible paths emanating from all children of the failed site. Figure 7 combines these extensions to form a quorum on a tree with degree greater than two. Note that for each node in the tree there is a field *Degree* that indicates the number of children of that node. Thus, each failed node is substituted by *Degree* number of paths starting from the children of the failed node. The proof of the intersection property for the quorums generated by the algorithm in Figure 7 can be easily established by extending Theorem 1 to arbitrary trees. Note, however, the minimality property may not hold. However, this property is needed only for reducing message costs but is not required for correctness. The minimality property is violated only when a node with only one child fails. This property could be enforced by additionally requiring the algorithm to terminate with an error condition when a node with a single child fails.

The algorithms described above do not require that the trees be complete. However, for the purpose of analysis we made the assumption that the trees

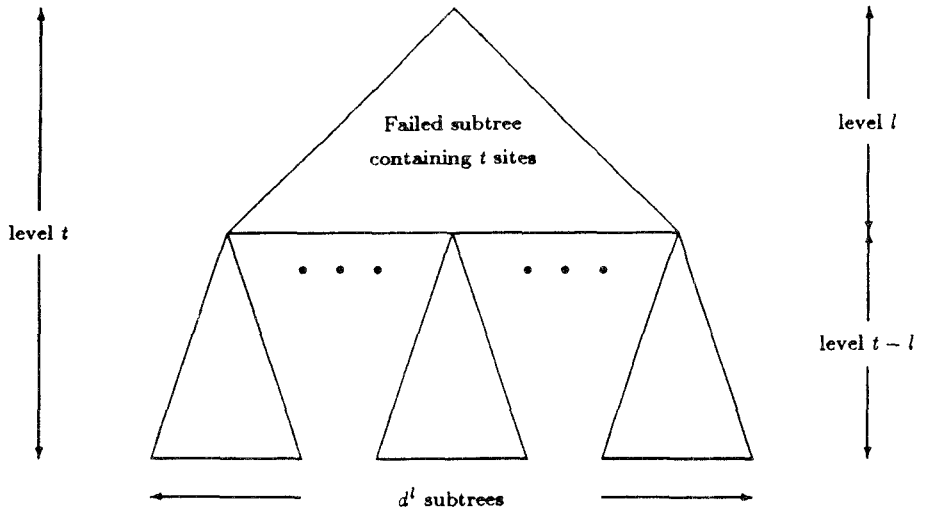

Fig. 6. A worst-case quorum in a tree of level t and degree d; l is computed from $t \leq (d^{l+1} - 1)/(d - 1)$.

```
FUNCTION GetQuorum(Tree : TREE; Requestor : SITE) : Quorum Set;
  VAR
    ChildQuorum : ARRAY[1..MaxDegree] OF QuorumSet;
  BEGIN
    IF Empty(Tree) THEN
      RETURN({ });
    ELSE IF GrantsPermission(Tree ↑ .NODE) THEN
      Pick I such that either:
        • Requestor is in the subtree rooted at Tree ↑ .Child[I], or
        • I is a random number in [1..Tree ↑ .Degree];
      RETURN({Tree ↑ .Node} ∪ GetQuorum(Tree ↑ .Child[I], Requestor));
    ELSE
      ∀ i ∈ [1..Tree ↑ .Degree] : ChildQuorum[i] ← GetQuorum(Tree ↑ .Child[i], Requestor);
      IF (∃ i ∈ [1..Tree ↑ .Degree] : ChildQuorum[i] = φ) THEN
        (* Unsuccessful in establishing a quorum *)
        EXIT(error);
      ELSE
        RETURN(⋃_{i=1}^{tree ↑ Degree} ChildQuorum[i]);
      END; (* IF *)
    END; (* IF *)
END GetQuorum;
```

Fig. 7. The algorithm for constructing a quorum on an arbitrary tree.

are complete, since it simplifies the analysis significantly. If the tree under consideration is incomplete, then the above analysis provides appropriate bounds. A side effect of this observation is that our algorithm can use any *spanning tree* [24] in a network. A spanning tree with a minimum radius (or

a minimum level) is most appropriate for our algorithm and will result in minimum sized quorums.

7. CONCLUSION

In this paper, we have proposed a simple and efficient algorithm for achieving mutual exclusion by introducing tree quorums. The algorithm for constructing tree quorums is novel in its use of a hierarchical structure of the network. This hierarchical structure is a logical organization of the network, and it does not imply that the underlying physical connections in the network actually form a tree. In most networks there is already an informal hierarchy, which is maintained for administrative purposes. For example, a departmental network in a university will most probably be organized in terms of fileservers, instructional workstations, faculty workstations, research workstations, a node that serves as the gateway to the external world, and so on. One of the maxims that is employed to construct such an informal hierarchy is to use the most reliable site as the root and the least reliable sites as the leaves. Thus, reliability generally increases from leaves to the root in the tree. Our approach requires that this informal hierarchy be made explicit, and should be used for distributed applications. Other distributed applications can also benefit from this organization, and thus will distribute the overhead of maintaining a logical control hierarchy in a network.

In a relatively failure-free environment the tree quorum algorithm achieves distributed mutual exclusion by communicating with as few as $\lceil \log n \rceil$ sites. This is not as good as the primary site approach, but is significantly better than previously proposed fault-tolerant algorithms, e.g., the majority quorum algorithm requires $n/2$ and Maekawa's protocol requires $\sqrt{n}$ sites. Note that in the primary site approach there is a danger of the primary site becoming a performance bottleneck. In contrast, this danger can be avoided by making the tree quorum algorithm adaptive. For example, if a root of the hierarchy becomes overloaded, it can ignore requests for mutual exclusion, and yet tree quorums can still be formed by traversing down the network hierarchy. Furthermore, the quorums never exceed the majority of sites in the network.

We now briefly examine the resiliency and fault-tolerance of the tree quorum algorithm. The majority quorum algorithm [7, 25] is the only scheme that is fault-tolerant to any $n/2$ site failures. Maekawa's solution [12] can withstand up to $n - \sqrt{n}$ site failures, but only in specific cases. In particular, $\sqrt{n}$ site failures may make it impossible to achieve mutual exclusion. Another problem with Maekawa's solution is that only one set is associated with each site. Thus, if any site in a set is not available, the site corresponding to that set cannot form a quorum. In our solution a site may be able to obtain mutual exclusion even when there are $n - \lceil \log n \rceil$ site failures. On the other hand, there may be situations when $\lceil \log n \rceil$ site failures make it impossible to construct a quorum. However, unlike Maekawa's solution, the tree quorum algorithm provides several alternative sets to each site for constructing quorums. Hence, the unavailability of one set on account of certain failures will not prohibit a site from achieving mutual exclusion.

Finally, tree quorums have the property of graceful degradation [13]. In a failure-free environment the size of a tree quorum is minimal, i.e., $\lceil \log n \rceil$. On the other hand, as failures occur the size increases to the maximum of $\lceil (n + 1)/2 \rceil$. Thus, our algorithm permits operations to be executed at a low cost, and the cost increases gradually with failures. In particular, our analysis demonstrates that in systems with high site availability, the tree quorum algorithm provides a comparable degree of availability to the majority quorum algorithm at logarithmic costs.

ACKNOWLEDGMENT

We would like to thank Rajiv Gupta for helping with the availability analysis of the tree quorums. We would also like to thank the anonymous referees for their comments and criticisms, which helped in improving the presentation of the paper. The analysis in Section 4.3 resulted from the comments of one of the referees.

REFERENCES

1. AGRAWAL, D., AND EL ABBADI, A. Exploiting logical structures of replicated databases. *Inf. Process. Lett. 33*, 5 (Jan. 1990), 255–260
2. 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* (Aug. 1990), 243–254.
3. ALSBERG, P. A., AND DAY, J. D. A principle for resilient sharing of distributed resources. In *Proceedings of the Second International Conference on Software Engineering* (Oct. 1976), 562–570.
4. BERNABÉU-AUBÁN, J. M., AND AHAMAD, M. Applying a path-compression technique to obtain an efficient distributed mutual exclusion algorithm. In *Proceedings of the Third International Workshop on Distributed Algorithms* (Sept. 1989), 33–44.
5. DAVIDSON, S. B., GARCIA-MOLINA, H., AND SKEEN, D. Consistency in partitioned networks. *ACM Comput. Surv. 17*, 3 (Sept. 1985), 341–370.
6. GARCIA-MOLINA, H., AND BARBARA, D. How to assign votes in a distributed system. *J. ACM 32*, 4 (Oct. 1985), 841–860.
7. GIFFORD, D. K. Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles* (Dec. 1979), 150–159.
8. GRAY, J. N Notes on database systems. In *Operating Systems: An Advanced Course, vol. 60 of Lecture Notes in Computer Science*. R. Bayer, R. M. Graham, and G. Seegmuller, Eds. Springer-Verlag, New York, 1978, 393–481
9. HELARY, J. M., PLOUZEAU, N., AND RAYNAL, M. A distributed algorithm for mutual exclusion in an arbitrary network. *Computer J. 31*, 4 (1988), 289–295.
10. LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 1978), 558–565.
11. LYNCH, N A., AND MERRITT, M . Introduction to the theory of nested transactions. Tech Rep MIT-LCS-TR-367, MIT, Cambridge, Mass., 1986.
12. MAEKAWA, M . A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst. 3*, 2 (May 1985), 145–159.
13. 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* (Aug. 1985), 237–249.
14. MISRA, J. Detecting termination of distributed computations using markers. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing* (Aug. 1983), 290–294.

15. Moss, J E B. *Nested Transactions: An Approach to Reliable Distributed Computing* MIT Press, Cambridge, Mass., 1985.
16. Naimi, M., and Trehal, M. How to detect a failure and regenerate the token in the log n distributed algorithm for mutual exclusion. In *Proceedings of the Second International Workshop on Distributed Algorithms, Lecture Notes in Computer Science 312*, Springer-Verlag, New York, 1987, 155-166
17. Raymond, K A tree-based algorithm for distributed mutual exclusion. *ACM Trans Comput. Syst. 7*, 1 (Feb. 1989), 61-77.
18. Ricart, G., and Agrawala, A K An optimal algorithm for mutual exclusion in computer networks *Commun. ACM 24*, 1 (Jan. 1981), 9-17.
19. Sanders, B. A The information structure of distributed mutual exclusion algorithms *ACM Trans. Comput. Syst. 5* (Aug. 1987), 284-299
20. Schneider, F B Synchronization in distributed programs. *ACM Trans. Program Languages Syst. 4*, 2 (April 1982), 125-148
21. Skeen, D. Non-blocking commit protocols. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (June 1982), 133-147
22. Skeen, D A quorum based commit protocol. In *Proceedings of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks* (Feb. 1982), 69-80
23 Suzuki, I , and Kasami, T. A distributed mutual exclusion algorithm *ACM Trans Comput. Syst. 3*, 4 (Nov 1985), 344-349
24. Tanenbaum, A S. *Computer Networks* 2d ed. Prentice Hall, Englewood Cliffs, N J , 1988.
25 Thomas, R H. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst. 4*, 2 (June 1979), 180-209.
26. van de Snepscheut, J. L Fair mutual exclusion on a graph of processes *Distribut Comput 2* (1987), 113-115.

Received February 1990; revised January 1991; accepted January 1991