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

D. Agrawal A. El Abbadi
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

In this paper, we present an efficient algorithm for managing replicated data. We impose a logical tree structure on the set of copies of an object. In a failure-free environment the protocol executes read operations by reading one copy of an object while guaranteeing fault-tolerance of write operations. It also exhibits the property of graceful degradation, i.e., communication costs 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.

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 protocols [Gif79, BG87, ES83, DB85, JM87, PL88] are needed to maintain the replicas. 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 network configuration information [ESC85, Her87, ET89]. This information is used to allow operations to adapt to changes in the network configuration.

In this paper, we present a replica control protocol 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. We describe a protocol that operates by reading one copy of an object while guaranteeing fault-tolerance of write operations and still does not require any reconfiguration on account of a failure and subsequent recovery. The protocol provides a comparable degree of data availability as other replica control protocols [Gif79] at substantially lower costs. Furthermore our approach is fault-tolerant, and exhibits the property of graceful degradation [MS85]. In a failure-free environment, the communication costs are minimal and as failures occur the cost of replica control 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 replica control. The tree quorum protocol, which incorporates these ideas, is presented in Section 4. Analysis of the tree quorum protocol and its comparison with other protocols are presented in Sections 5 and 6. We conclude with a discussion of our results.

*This research is supported by the NSF under grant numbers CCE-8809387 and IRI-8809284.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

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* [SS82] and communication links may fail to deliver messages. Combinations of such failures may lead to *partitioning failures* [DGMS85], 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 *transaction* programs. 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*. A commonly accepted correctness criteria in distributed databases is the *serializable* execution of transactions [EGLT76]. The serializable execution is guaranteed by employing a *concurrency control* mechanism, e.g., two-phase locking, timestamp ordering, or an optimistic concurrency control protocol.

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* [BG87] and is enforced by the *replica control* protocol. The correctness criteria for replicated databases is *one-copy serializability* [BG87], which ensures one-copy equivalence and 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 non-empty intersection.

Version numbers or timestamps are used to identify the current copy in a quorum. When timestamps are used intersection of write operations is not necessary [Her86].

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 [Gif79] and dynamic [DB85, JM87, PL88], 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 costly than in the read-one write-all protocol. The second approach [ESC85, ET89, Her87] 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 read-one write-all protocol, i.e., a read operation accesses a single copy, when there are no failures in the system. As failures occur read operations may be required to access more copies. Write operations, on the other hand, tolerate failures, and no reconfiguration protocol is used. 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 presented. We 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 the protocol.

A straight forward 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

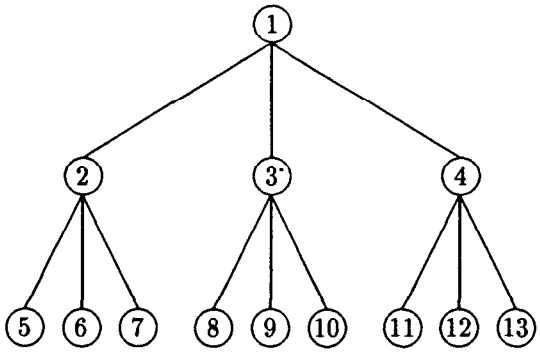

Figure 1: A tree organization of 13 copies of an object

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}. It is clear that any read operation must have at least one copy in common with any write operation, and hence the protocol ensures one-copy equivalence.

The 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 after the failure of several copies at different levels (for example, 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 allow write and read operations to execute but at a higher cost. In particular, read operations can tolerate the failure of all 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 must try and access the root, if the root is inaccessible, the read tries to access a majority of the root's children. For each inaccessible copies 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 copies 2, 3, and 4. All read quorums have a non-empty intersection with any write quorum, e.g., {1,2,3,5,6,8,9}. In the next section, we formally develop the protocol and argue its correctness.

4 The Tree Quorum Protocol

In this section we present the tree quorum protocol for accessing objects in a distributed replicated database. 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

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

votes. To ensure the quorum intersection property, the sum of q_r and q_w must be greater than the total number of copies 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 $2d+1$, i.e., each node has $2d+1$ 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. The logical tree organization may be viewed simply as an ordering of the name directory of copies maintained at each site for location purposes. Extending it to a tree does not impose any extra or special complications.

In Figure 2, we present a protocol for constructing a valid write quorum. We assume that the tree has a well defined root, and that a transaction attempting to construct a write quorum calls the recursive function *WriteQuorum* with the root of the tree, c_0, as parameter. The protocol tries to construct a quorum by selecting the root and a majority of its children. For each selected child, the protocol adds a majority of its children to the quorum. This process continues until the leaves are reached. If successful, this set of copies constitutes a write quorum. If the function is unable to collect the required majority at any level, it returns the empty set to indicate that the write quorum could not be formed. Note that depending on the failures in the system a write operation may construct different write quorums.

In Figure 3, we present a protocol for constructing a valid read quorum. A transaction attempting to construct a read quorum calls the recursive function *ReadQuorum* with the root of the tree, c_0, as parameter. The protocol tries to construct the quorum by selecting the root c_0. If successful, this node constitutes the read quorum. If it fails, it tries to access a majority of the root's children. Again if successful this set constitutes the read quorum, otherwise, for each copy, which is inaccessible, the protocol tries to replace it with a majority of its children. This process is repeated recursively until a set of copies is included in the read quorum, or no such copies are accessible. In this case, the function returns the empty set and the operation is aborted.

Consider a replicated object with thirteen copies. We superimpose a ternary tree on the copies as illustrated in Figure 1, with the sites numbered as shown. In this case $2d+1 = 3$ and $h = 2$. According to the protocol any write quorum 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 write quorum: {1,2,3,5,6,8,9}, {1,2,4,6,7,12,13}, etc. A read quorum, in the best case, is required to access only the root. However, as a result of the failure of the root, a read quorum can be formed from any majority 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 copies 3 and 4 are inaccessible or copies 2 and 4. 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.

We now present a discussion of the upper and lower bounds on the sizes of quorums generated by the tree quorum protocol. In the static voting scheme [Gif79], quorum sizes are fixed although the membership in a quorum may vary. In our approach the sizes of the read quorums vary while the choice of members is relatively less flexible. Consider a tree with n nodes, height h and each node is of degree $2d+1$. In the case of read operations the protocol exhibits the property of graceful degradation [MS85]. In the absence of failures the cost of forming a quorum is minimal, and this cost increases as failures occur in the system. In the best case, a read operation is executed by accessing a *single* copy: the root. However, if the root fails, the cost of forming a quorum increases to $d+1$, a majority of the root's children. As more failures occur, the quorum size increases up to a maximum of $(d+1)^h$. Note that, in general, $(d+1)^h < n/2$. A read operation may tolerate the failure of $n-1$ specific copies, and can tolerate the failure of any $[(d+1)^{h+1}-1]/d-1$ copies. Write operations, on the other hand, are all of the same size: $[(d+1)^{h+1}-1]/d$. Different failures, however, effect write operations in different ways. In the worst case,

```
FUNCTION WriteQuorum(Tree : TREE) : QUORUM;
VAR
    SubTrees, Majority: QUORUM;
BEGIN
    IF Empty(Tree) THEN
        RETURN({});
    ELSE IF Tree ↑ .Root is write accessible THEN
        (* Collect majority of subtrees to be included with the root of the subtree *)
        SubTrees = ⋃_{i∈Majority} WriteQuorum(Tree ↑ .SubTree[i]);
        IF Unable to collect a majority THEN
            RETURN({});
        ELSE
            RETURN(Tree ↑ .Root ∪ SubTrees);
        END; (* IF *)
    ELSE
        RETURN({});
    END; (* IF *)
END WriteQuorum;
```

Figure 2: The Algorithm for constructing a Write Quorum on a Tree of Copies

```
FUNCTION ReadQuorum(Tree : TREE) : QUORUM;
VAR
    MajorityQuorum, Majority: QUORUM;
BEGIN
    IF Empty(Tree) THEN
        RETURN({});
    ELSE IF Tree ↑ .Root is read accessible THEN
        RETURN(Tree ↑ .Root);
    ELSE
        (* Collect majority of subtrees to substitute for the root of the subtree *)
        MajorityQuorum = ⋃_{i∈Majority} ReadQuorum(Tree ↑ .SubTree[i]);
        IF Unable to collect a majority THEN
            RETURN({});
        ELSE
            RETURN(MajorityQuorum);
        END; (* IF *)
    END; (* IF *)
END ReadQuorum.
```

Figure 3: The Algorithm for constructing a Read Quorum on a Tree of Copies

the failure of the root prohibits a write operation from execution, while in the best case, a quorum can still be formed after the failure of $n - \lceil (d+1)^{h+1} - 1 \rceil / d$ specific copies.

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 non-empty intersection. Note that two write quorums will always have a non-empty intersection.

Theorem 1 The tree quorum protocol guarantees the intersection of read and write quorums.

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

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

Induction Hypothesis. Assume that the theorem holds for trees of height h.

Induction Step. Consider a tree of height $h+1$. The read and write quorums constructed for this tree will be of the following form:

1. Read Quorum: {root} OR {majority of read quorums for subtrees of height h}.
2. Write Quorum: {root} $\bigcup$ {majority of write quorums for subtrees of height h}.

Now if a read quorum consists of the root of the tree, it is guaranteed to have a nonempty intersection with any write quorum. If, on the other hand, the read quorum consists of a majority of read quorums for subtrees of height h, it is guaranteed to have at least one subtree in common with any write quorum. Since the subtrees are of height h, the induction hypothesis guarantees that read and write quorums will have a nonempty intersection.

Hence, by induction, the tree quorum protocol guarantees non-empty intersection between read and write quorums. □

One of the restrictions imposed by the suggested implementation for collecting read quorums is that the reads are directed to a specific copy: the root. This has the advantage that if the root is up, read operations accesses a single copy. Read locality may, however, be sacrificed and the root may become a bottleneck. To solve this problem, it might be more desirable to gather a quorum of several relatively-local copies rather than one very remote root copy. This approach could also be used for organizing the tree structure of the copies. For example, consider a network composed of two relatively distant segments: the root could be placed in one of the segments and the second level of the tree in the other segment. In such an organization, transactions executing in a particular segment will use the quorum which is less expensive. If the root is in the transaction's network segment, the root will be accessed. Otherwise, the transaction will access the majority of copies at the second level of the tree. The functions depicted in Figures 2 and 3 should be appropriately modified to enforce this policy.

5 Analysis of the Tree Protocol

In this section, we estimate the cost and the availability of read and write operations in the tree protocol and compare them with the read-one write-all (ROWA) and the voting (VOTE) protocols [Gif79]. It is particularly important to demonstrate that the availability of write operations is not substantially degraded by the tree quorum protocol's requirement that all writes include the root.

The message cost of an operation is directly proportional to the size of quorums required to execute the operation. Thus in the read-one write-all approach, read operations have a cost of one whereas write operations have a cost of n. In the voting protocol, the quorum size corresponding to a majority is $\lceil (n+1)/2 \rceil$. Thus, read and write operations have a cost of $\lceil (n+1)/2 \rceil$.

In the case of the tree quorum protocol (TREE), the size of read quorums vary from 1 to $(d+1)^h$. On the other hand, the cost of write operations is $[(d+1)^{h+1} - 1]/d$. In order to compute the average cost of read operations, we introduce a parameter f that indicates the fraction of read operations that will use the root of the tree to execute. Let RC_k be the average cost of executing read operations in a tree of height k. Thus, the cost RC_{k+1} for a tree of height $k+1$ is:

$$RC_{k+1} = f \cdot 1 + (1-f)(d+1)RC_k$$

where $RC_0 = 1$. Note that the first term in the recurrence relation corresponds to the fraction of read operations, f, that execute using the root of the tree. The second term corresponds to the situation when the read operations collect a read quorum on a majority of subtrees. Thus, $f = 0$ is the upper bound on the cost of read operations which is $(d+1)^h$ for a tree of height

h and degree $2d+1$, and $f=1$ is the lower bound on the cost, which is 1, the same as the read-one write-all protocol.

Figure 4 illustrates the cost of executing read and write operations in various replica control protocols. For the read operations in the tree quorum protocol, we indicate the upper and lower bounds on the cost, and provide a realistic bound for the read costs which corresponds to $f=0.5$ (i.e., 50% of read operations execute using the root). 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 tree quorum 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 our protocol has definite advantages. However, we need to demonstrate that the availability of write operations is not seriously compromised.

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 [AA89]. 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 non-negative constant k, the above probabilities can be represented by the following terms, i.e.,

$$= \binom{2k+1}{k+1} p^{k+1}(1-p)^k + \ldots +$$

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

The availability of read and write operations in the tree quorum protocol can be estimated 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 R_h be the availability of read operations in a tree of height h. Thus, the availability for a tree of height $h+1$ is given as:

R_{h+1} = Probability(Root is up)+
Probability(Root is down)×
[Read Availability of Majority of subtrees]

Assuming that the degree of each node in the tree is $2d+1$, where $d>0$ and taking p as the probability that the root is available and $1-p$ as the probability that the root is unavailable, we get:

$$R_{h+1} = p + (1-p)\times$$

$$\left[\binom{2d+1}{d+1}(R_h)^{d+1}(1-R_h)^d + \ldots + (R_h)^{2d+1}\right]$$

Similarly, let W_h be the availability of write operations in a tree of height h. Then:

W_{h+1} = Probability(Root is up)×
[Write Availability of Majority of subtrees]

i.e.,

$$W_{h+1} = p\times$$

$$\left[\binom{2d+1}{d+1}(W_h)^{d+1}(1-W_h)^d + \ldots + (W_h)^{2d+1}\right]$$

Note that R_0 and W_0 is p. Since the above recurrence relations involve non-linear terms, we illustrate the operation availabilities for specific replica configurations of an object in Figures 5 and 6. In Figure 5, the object is replicated at four sites. Furthermore, we assume that the copies are organized as a tree of degree three with height one. In Figure 6, the object is replicated at thirteen sites. The tree is assumed to be a ternary tree of height two. The availability of read operations in all three protocols is almost identical. As expected, write availability of the tree quorum protocol is inferior to the voting protocol. However, it is substantially superior to the write availability in 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 tree quorum protocol achieves the benefits of the read-one write-all protocol while significantly improving both the cost and the availability of write operations.

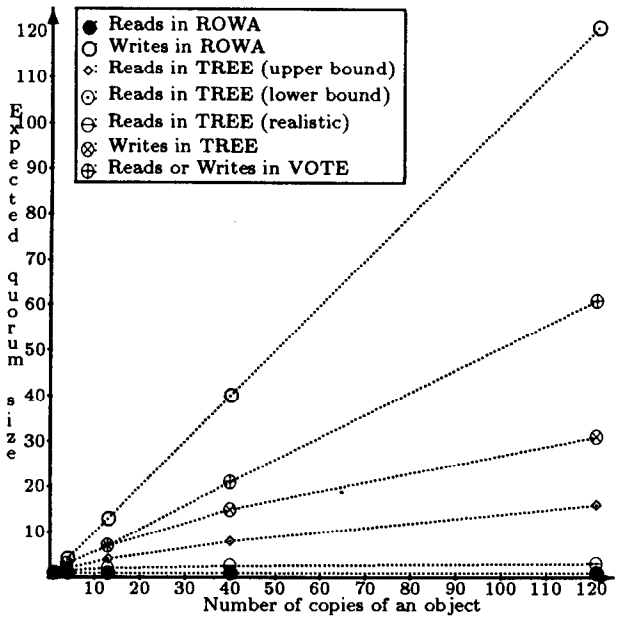

Figure 4: Expected Cost (Note: The tree is of degree 3.)

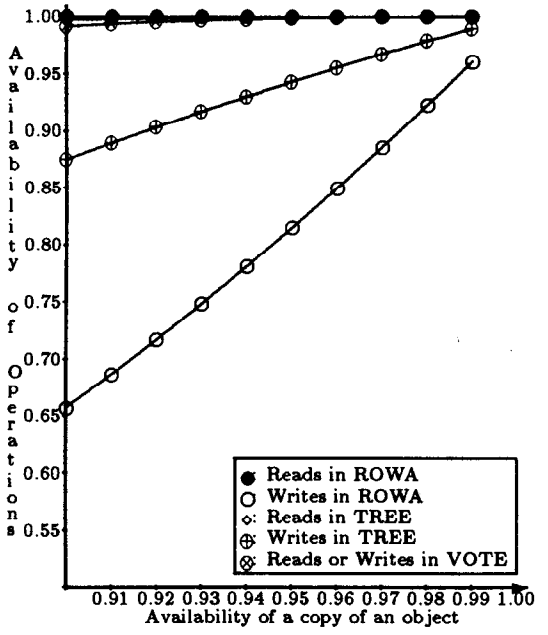

Figure 5: Availability of an object with four copies

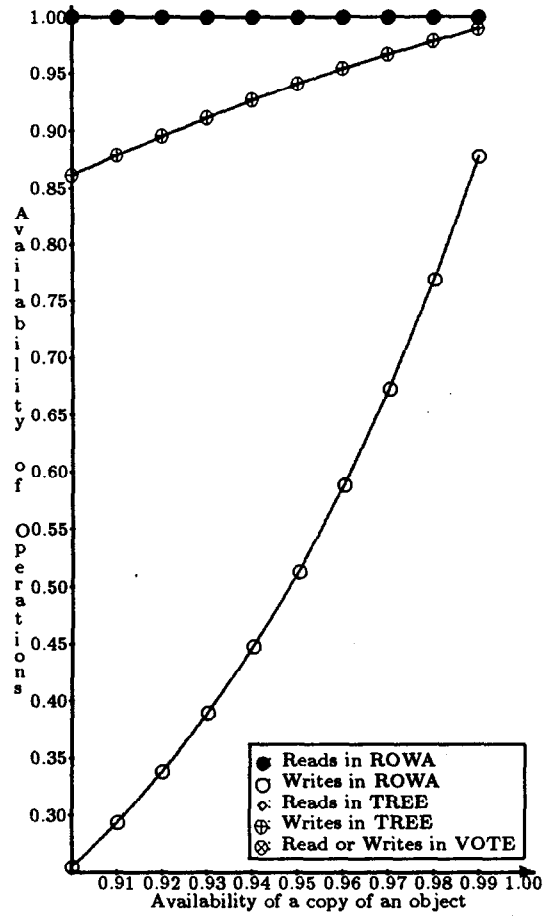

Figure 6: Availability of an object with thirteen copies

6 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. In a failure free system, read operations in both protocols access a single copy while write operations in the tree quorum 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 tree protocol write operations can be executed even after certain failures have occurred and the degree of write availability is significantly higher as indicated by the results in Section 5. 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* [Gif79], a write operation writes w copies, and a read operation accesses r copies where $r + w$ is greater than the total number of copies of the object. In the *dynamic voting protocol* [DB85, JM87, PL88], both read and write operations must access a majority of the copies that were most recently updated. In a failure free system, both the static and the dynamic protocols require read operations to access several copies. For write operations to tolerate the failure of t copies, a read operation using the static approach must always access $t+1$ copies and vice-versa. In the dynamic approach, both read and write operations must access a majority of the copies. The tree protocol, in a failure free system, never requires a read operation to access more than one copy. Furthermore, write operations access $[(d+1)^{h+1}-1]/d$ copies, which in general is less than a majority of copies. When failures occur, in the best case, the static voting protocols can tolerate the failure of a minority of copies. This improved availability requires that both read and write operations access a majority of copies. The *dynamic voting protocol* can tolerate the failure of any number of copies. It, however, requires both operations to access several copies (a majority of the copies most recently written). Using our protocol, read operations can tolerate the failure of any $[(d+1)^{h+1}-1]/d-1$ copies, and $n-1$ specific copies. If the root is accessible, read operations always access a single copy and in the worst case access $(d+1)^h < n/2$ copies. Write operations, in the worst case, cannot be executed after the failure of the root, but can tolerate the failure of $n-[(d+1)^{h+1}-1]/d$ specific copies. The analysis of Section 5, shows that the read availability provided by the tree quorum protocol is comparable to that of the static voting protocol and that the degradation of write availability is not substantial. The cost of both operations is significantly less when the tree quorum protocol is used.

To overcome the problem of expensive read operation in the voting protocols, several algorithms have been proposed that use network configuration information [ESC85, Her87, ET89]. 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 several copies of several different objects. The tree quorum protocol tries to achieve 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 [Mae85] proposed imposing a logical grid on a set of sites to derive efficient $O(\sqrt{n})$ solutions for mutual exclusion. Agrawal and El Abbadi [AE89] proposed imposing a logical tree to solve the mutual exclusion problem using $O(\log n)$ messages. This approach was extended to replica control protocols that use several logical structures imposed on a set of copies [AE90]. Kumar [Kum90] 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 read operations, and unlike other logical structure based approaches, the tree quorum protocol, in a failure-free system does not require read operations to access more than one copy.

7 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: the necessity of read operations to access several copies in order to ensure the fault-tolerance of write operations. Our approach imposes a logical tree structure on the set of copies implementing a logical object. The logical structure will be particularly beneficial if it is organized such that the most reliable site is chosen as the root and the least reliable sites as the leaves. This tree structure is used by transactions to determine how to execute read and write operations both in failure-free and failure-prone environments. In that sense, the structure encodes reconfiguration information.

In any practical distributed database system, most read operations should be executed by an access to a single copy. Our approach ensures such performance in a failure-free environment, and in particular when a specific node, the root, is accessible. This performance is attained without any reconfiguration requirements and without any substantial availability degradation for write operations. 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. Several replicated databases use the read-one write-all protocol for its simplicity, low read costs, and because it does not require any reconfiguration protocols. Our approach is fairly simple, does not require any reconfiguration, and significantly improves the availability of write operations when compared with the read-one write-all approach.

The tree quorum protocol can be easily extended to provide the database designer with the flexibility of determining the degree of availability as well as the cost of different operations. For example, instead of requiring write operations to write copies residing at *all* levels of the tree, they would be required to write copies at all levels except for one. In this case, a write operation does not have to write the root copy, if the site on which it resides is inaccessible, thus increasing the availability of write operations. On the other hand, to ensure correctness, read operations would have to read copies at two levels, thus increasing the cost of read operations. In general a write operation would be required to write copies at w levels, and read operations would read copies at $l-w+1$ levels (where l is the total number of levels in the tree). This approach increases the availability of write operations, while at the same time increasing the cost of read operations. Another extension of our protocol is to use a reconfiguration protocol [ESC85, ET89] for restructuring the logical tree. This approach will be particularly beneficial for large distributed databases.

Finally, the proposed protocol exhibits the property of graceful degradation [MS85], 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 comparable to the simplest and most efficient replica control protocol, the read-one write-all protocol. When failures occur, and unlike the read-one write-all protocol, the tree quorum protocol continues executing both read and write operations with a high probability, although at a higher cost. This approach in designing distributed systems is desirable since it provides fault-tolerance without imposing unnecessary costs on the failure-free mode of operations.

Acknowledgements

We would like to thank the anonymous referees for their constructive comments.

References

[AA89] M. Ahamad and M. H. Ammar. Performance Characterization of Quorum-Consensus Algorithms for Replicated Data. *IEEE Transations on Software Engineering*, 15(4):492–495, April 1989.

[AE89] D. Agrawal and A. El Abbadi. An Efficient Solution to the Distributed Mutual Exclusion Problem. In *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing*, pages 193–200, August 1989.

[AE90] D. Agrawal and A. El Abbadi. Exploiting Logical Structures of Replicated Databases. *Infor-*

mation Processing Letters, 33(5):255–260, January 1990.

[BG87] P. A. Bernstein and N. Goodman. A Proof Technique for Concurrency Control and Recovery Algorithms for Replicated Databases. *Distributed Computing, Springer-Verlag*, 2(1):32–44, January 1987.

[DB85] D. Davcev and W. Burkhard. Consistency and Recovery Control for Replicated Files. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 87–96, December 1985.

[DGMS85] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[EGLT76] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notion of Consistency and Predicate Locks in Database System. *Communications of the ACM*, 19(11):624–633, November 1976.

[ES83] D. Eager and K. Sevcik. Achieving Robustness in Distributed Database Systems. *ACM Transactions on Database Systems*, 8(3):354–381, September 1983.

[ESC85] A. El Abbadi, D. Skeen, and F. Cristian. An efficient fault-tolerant protocol for replicated data management. In *Proceedings of the Fourth ACM Symposium on Principles of Database Systems*, pages 215–228, March 1985.

[ET89] A. El Abbadi and S. Toueg. Maintaining Availability in Partitioned Replicated Databases. *ACM Transaction on Database Systems*, 14(2):264–290, June 1989.

[Gif79] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, pages 150–159, December 1979.

[Her86] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[Her87] M. Herlihy. Dynamic Quorum Adjustments for Partitioned Data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[JM87] S. Jajodia and D. Mutchler. Dynamic Voting. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 227–238, June 1987.

[Kum90] A. Kumar. Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects. In *Proceedings of the Tenth International Conference on Distributed Computing Systems*, May 1990.

[Mae85] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, May 1985.

[MS85] S. R. Mahaney and F. B. Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *Proceedings of the Fourth ACM Symposium on Principles of Distributed Computing*, pages 237–249, August 1985.

[PL88] J. F. Pâris and D. E. Long. Efficient Dynamic Voting Algorithms. In *Proceedings of the Fourth IEEE International Conference on Data Engineering*, pages 268–275, February 1988.

[SS82] R. Schlicting and F. B. Schneider. Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *ACM Transactions on Computer Systems*, 1(3):222–238, August 1982.