Resilient Logical Structures for Efficient Management of Replicated Data*

D. Agrawal A. El Abbadi
Department of Computer Science
University of California
Santa Barbara, CA 93106
{agrawal,amr}@cs.ucsb.edu

Abstract

Replicated data management protocols have been proposed that exploit a logically structured set of copies. These protocols have the advantage that they provide limited fault-tolerance at low communication cost. The proposed protocols can be viewed as analogues of the read-one write-all protocol in the context of logical structures. In this paper, these protocols are generalized in two ways for a grid and a tree structure. First, the quorum based approach is applied to develop protocols that use structured read and write quorums, thus attaining a high degree of data availability for both read and write operations. Next, the reconfiguration or views approach is developed for both grid and tree structures resulting in protocols that attain high degrees of availability at significantly low communication cost for read operations. In this sense, the proposed protocols have the advantages of the read-one write-all protocol for low cost read operations as well as the majority quorum protocol for high data availability.

*This research is supported by the NSF under grant numbers IRI-9004998 and IRI-9117904.

Proceedings of the 18th VLDB Conference
Vancouver, British Columbia, Canada 1992

1 Introduction

Data replication is used in distributed systems to increase data availability and to achieve fault-tolerance. By storing multiple copies of data at several sites in the system, there is an increased likelihood of data remaining available and accessible to users despite site and communication failures. However, complex and expensive synchronization mechanisms [Gif79, Tho79, Sto79, ES83, DB85, ESC85, BG87, PL88, JM90] are needed to maintain the consistency and integrity of data. One of the simplest protocols for managing replicated data is one where read operations on an object 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. This protocol results in the imbalance of availability of read and write operations: read operations have a higher availability whereas write operations have a lower availability when compared to the case when data is not replicated.

In order to increase the availability of both read and write operations, the quorum protocol [Gif79, Tho79] was proposed. The quorum protocol generalizes the read-one write-all protocol by imposing an intersection requirement between read and write operations. Write operations can be made fault-tolerant since they do not need to access all copies of an object. However, this is at the cost of requiring read operations to access more than one copy of the object. Dynamic quorum protocols [DB85, PL88, JM90] have also been proposed to further increase availability in replicated databases. However, these approaches do not address the issue of

low-cost read operations.

The view-based protocol [ESC85, ET89] uses configuration information to achieve high availability of both read and write operations, as well as low cost of read operations. The configuration information allows users to execute transactions by employing the read-one write-all protocol and hence results in low cost read operations. When changes occur in the network due to failures and recovery, a reconfiguration protocol is executed to update the configuration information. The reconfiguration protocol uses the notion of quorums and hence read and write operations have high availability. This approach is especially useful if failures are infrequent and persistent, and hence represents an alternative design philosophy to the quorum approach. In particular, with the quorum approach each operation is penalized for fault-tolerance purposes, but changes in the network configuration remain transparent to the transactions. On the other hand, the reconfiguration approach executes a special protocol when the network configuration changes (not necessarily immediately), but as a result user transactions are not penalized and can always execute using the desired communication cost.

Recently, several researchers have proposed imposing a logical structure on the set of copies in the database, and using structural information to create intersecting quorums. Protocols that use a logical structure, e.g., the grid protocol [Mae85, CAA90] and the tree protocol [AE91, AE90], execute operations with low communication costs while providing fault-tolerance for both read and write operations. However, as with the original read-one write-all approach, the improved performance for read operations results in a degraded write availability when failures occur. In particular, structure-based protocols are vulnerable to the failure of specific sites. In this sense, these protocols are analogous to the read-one write-all protocol, i.e., low cost read operations but write operations are vulnerable to failures. In this paper, we first describe the extension of these two protocols to improve the fault-tolerance of write operations by using the notions of structured read and write quorums with respect to a logical structure. As is the case in the standard quorum protocols [Gif79, Tho79], the increased fault-tolerance for write operations is at the increased cost of executing read operations. In order to let users continue using the analogues of the read-one write-all protocol in the context of a logical structure, we develop reconfiguration protocols for dynamically adapting to failures and recovery. This results in the following dichotomy. Users accesses are through the simple analogues of the read-one write-all protocol with respect to a logical structure and therefore have low communication cost for read operations. On the other hand, the reconfiguration protocol uses the notion of quorums in the context of a logical structure to ensure high data availability.

The paper is organized as follows. In the next section, we present the model of the system. The quorum and reconfiguration protocols for grid-based logical structures are presented in Section 3 and for tree-based logical structures are presented in Section 4. Both sections include performance analyses that demonstrate the superiority of the proposed approach. The paper concludes with a discussion of our results.

2 Model

A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network. We assume that sites are *fail-stop* [SS82] and communication links may fail to deliver messages. Combinations of such failures may lead to *partitioning failures* [DGS85], where sites in a *partition* may communicate with each other, but no communication can occur between sites in different partitions. A site may become *inaccessible* due to site or partitioning failures.

A *distributed* database consists of a set of *objects* stored at several sites in a computer network. Users interact with the database by invoking *transactions*, which are partially ordered sets of atomic read and write operations. The execution of a transaction must be atomic, i.e., a transaction either *commits* or *aborts* [Gra78]. A commonly accepted correctness criteria in databases is the *serializable* execution of transactions [EGLT76]. The serializable execution is guaranteed by employing a *concurrency control* mechanism.

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 a *replica control* protocol. The correctness criteria for replicated databases is *one-copy serializability* [BG87], which ensures both one-copy equivalence and the serializable

execution of transactions. In order to ensure one-copy equivalence, a replicated object x may be read by reading a *read quorum* of copies, and it may be written by writing a *write quorum* of copies. The following restriction is placed on the choice of quorum assignments [Gif79, Tho79]:

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

Version numbers or timestamps are used to identify the current copy in a quorum. When timestamps are used, intersection of write quorums is not necessary [Her86]. The notion of intersecting quorums is closely related to *coteries* [GB85]. In coteries, an added minimality condition is imposed upon quorums while with intersecting quorums a distinction is made between read and write operations.

The view-based protocol uses configuration information to achieve fault-tolerance of both read and write operations, while reduing the cost of executing read operations. We can summarize the rules for reconfiguration from [ESC85, ET89] as follows:

- Configuration Rule: Each site s has associated with it a set $View[s]$, which is the set of sites s assumes it can communicate with. Each view has associated with it a special identifier, which uniquely determines a configuration. Furthermore, these identifiers are totally ordered.

- Operation Execution Rule: Read and write operations are executed by accessing copies that reside on the sites with the same view. Read and write operations executed in the same view must have a nonempty intersection.

Reconfiguration is initiated through a special transaction called a *view-reconfiguration* transaction. The reconfiguration transaction updates copies and configurations in an atomic step. The transaction succeeds in including an object in a new view only when it can determine the current value of that object. In order to ensure that the new view has the current value, the reconfiguration transaction must access a *reconfiguration quorum* which satisfies the following property:

- Reconfiguration Quorum Property: A reconfiguration quorum must have a nonempty intersection with any write quorum and two reconfiguration quorums must have a nonempty intersection[1].

The above rules in addition to a concurrency control protocol ensure one-copy serializability [ET89].

3 The Grid Structure

In this section we propose using a grid structure to define quorums for both read and write operations. We start by describing previously proposed protocols that use a grid structure, and generalize them using the quorum approach to decrease their vulnerability to site and communication failures. We then apply the reconfiguration paradigm to grid-based quorum protocols, and thus reduce the cost of executing read operations, while maintaining a high degree of data availability.

3.1 The Grid Quorum Protocol

Maekawa [Mae85] proposed using the notion of finite projective planes to obtain a distributed mutual exclusion algorithm where all quorums are of equal size. If the number of sites in the system is n then mutual exclusion can be achieved by communicating with $\sqrt{n}$ sites. Maekawa further suggested logically organizing the sites in the form of a two-dimensional grid to ensure quorums of size $O(\sqrt{n})$. Chang, Ahamad, and Ammar [CAA90] extended Maekawa's grid protocol for replicated data to support read and write operations. Unlike Maekawa's protocol where an operation excludes every other operation, in this protocol read operations do not exclude other read operations but ensure exclusion of write operations. Furthermore a write operation excludes both read and write operations. In this protocol, n copies of a data object are logically organized in the form of a $\sqrt{n} \times \sqrt{n}$ grid[2] as shown in Figure 1. Read operations on this object are executed by acquiring a read quorum that consists of a copy from each column in the grid. Write operations, on the other hand, are executed by acquiring a write quorum that consists of all copies in one column and a copy from each of the remaining columns. Since read and write operations access $O(\sqrt{n})$ copies, we will refer to this protocol as the $\sqrt{read/write}$ protocol.

In Figure 1, copies $\{1,7,13,19,25\}$ are sufficient to execute a read operation whereas copies $\{1,6,11,16,21,7,13,4,20\}$ will be required to execute

[1] The latter part of the condition is relaxed in [ET89].

[2] For simplicity, we assume that n is a perfect square.

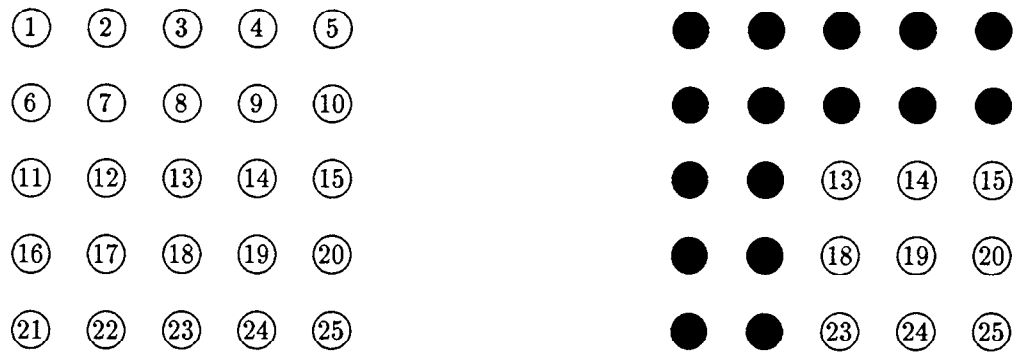

Figure 1: A grid organization and an example of a failure pattern: black circles indicate unavailable copies

a write operation. It can be easily shown that a write quorum intersects with both read and write quorums in this protocol. This protocol provides low-cost execution of operations ($\sqrt{n}$ compared to $n/2+1$ in the majority protocol [Tho79, Gif79]) while providing a comparable degree of availability [CAA90]. However, there are several drawbacks associated with this protocol. In particular, if copies in an entire column become unavailable due to failures or network partitioning, read and write operations cannot be executed. Note that this is the case even though a majority of copies are available in the system. Similarly, if copies in an entire row are unavailable, no write operations can be executed. These shortcomings of the protocol can be overcome either by using the notion of quorums or by employing dynamic reconfiguration in the context of grids.

We start by generalizing the notion of quorums in the context of the grid structure. Given an object organized as a grid of dimensions $\sqrt{n} \times \sqrt{n}$, we define a *read grid quorum* and a *write grid quorum*. In general, a *grid quorum* has length l and width w, and is represented as a pair $\langle l, w \rangle$. A read operation is executed by accessing a read grid quorum $\langle l, w \rangle$, which is formed from l copies in each of w different columns. A write operation is executed by writing a write grid quorum, $\langle l, w \rangle$, which is formed from (a) l copies in each of w columns as well as (b) any $\sqrt{n} - l + 1$ copies in each of $\sqrt{n} - w + 1$ columns. (The second component of the write quorum is to ensure the intersection between two write quorums, if timestamps are used instead of version numbers, this component is not needed.) In order to ensure the quorum intersection property between read and write quorums, if the read grid quorum is of size $\langle l, w \rangle$ then the write grid quorum must be $\langle \sqrt{n} - l + 1, \sqrt{n} - w + 1 \rangle$. Note that if both length and width of a write quorum is greater than $\sqrt{n}/2+1$, then the second component, i.e., condition (b), is vacuously satisfied by the copies already included due to condition (a). It is easy to show that quorums that satisfy these conditions have non-empty intersections.

In the $\sqrt{read/write}$ protocol proposed by Chang, Ahamad and Ammar [CAA90], read quorums have length one and width $\sqrt{n}$, while write quorums have length $\sqrt{n}$ and width one. This protocol is especially interesting since both read and write operations have size $O(\sqrt{n})$. The protocol is, however, vulnerable to the failure of an entire column or row in the grid. If an entire column is inaccessible, both read and write operations cannot be executed, while if an entire row is inaccessible, write operations are unavailable. We now define an alternative quorum assignment in which the two dimensions of the grid quorums are used as follows. The length dimension is used for ensuring efficient execution of read operations and the width dimension for ensuring fault-tolerance of write operations. In particular, we use the assignment that corresponds to the read-one write-all in the length dimension and the majority quorum in the width dimension. That is, read quorums have dimensions $\langle 1, \sqrt{n}/2+1 \rangle$ and write quorums have dimensions $\langle \sqrt{n}, \sqrt{n}/2 + 1 \rangle$. We will refer to this protocol as the $\sqrt{read}$ grid quorum protocol. This assignment ensures that read operations have a low cost, $\sqrt{n}/2 + 1$, and have a high degree of availability, since they are not vulnerable to the failure of a minority of the columns. Write operations, on the other hand, are more available than the read-one write-all protocol since only $n/2 + \sqrt{n}$ copies are needed to execute write operations. However, few specific failures may render write operations unavailable.

To further increase the fault-tolerance of write operations, we define the *majority grid quorum* assignment where both read and write quorums have dimensions $\langle\sqrt{n}/2+1, \sqrt{n}/2+1\rangle$, i.e., both operations are executed by accessing a majority of copies in a majority of the columns. This protocol tolerates the failures of three quarter of the sites in the grid. However, this increase in fault-tolerance is at the increase in the cost of read operations: instead of accessing $\sqrt{n}/2$ copies, reads now have to access $n/4$ copies.

Consider an object with 25 copies organized as a grid of 5×5 as shown in Figure 1. A read grid quorum of dimensions $\langle 2,3\rangle$, may be formed from the following set {1,11,7,17,19,24}. Note that in this case, no copies in the third and fifth columns are included in the quorum. Hence unlike the $\sqrt{read/write}$ protocol, the unavailability of two entire columns is tolerated by a read operation. In this case, a write grid quorum must have length 4 and width 3, and hence can be formed from the following set: {1,6,11,21,3,8,13,23,5,10,15,25}. Again notice that this set ensures that a write quorum can tolerate the unavailability of an entire row (in this case the fourth row), or two columns (in this case the second and the fourth columns).

3.2 Reconfiguring Grids

Although the grid quorum protocol achieves high data availability, increase in fault-tolerance is obtained at the increased cost of read operations. For example, to tolerate the failure of an entire row, a quorum grid protocol must have write quorums with lengths $\sqrt{n}-1$: read quorums with dimensions $\langle 2, \sqrt{n}/2+1\rangle$, and write quorums have dimensions $\langle\sqrt{n}-1, \sqrt{n}/2+1\rangle$. In this case a read operation must access two copies instead of one in a majority of columns, thus doubling the number of copies accessed by a read operation, when compared with the $\sqrt{read}$ assignment. Instead the reconfiguration approach can be used to increase fault-tolerance while maintaining low communication costs. After reconfiguration the $\sqrt{read}$ assignment rules are used for constructing read and write grid quorums. Note that these rules ensure that any write quorum must have a non-empty intersection with every other read or write quorum. For reconfiguration we use the majority grid quorum assignment, which provides the highest degree of fault-tolerance (in general the majority assignment provides the best data availability [AA89] when copies have equal weights). The reconfiguration rule must ensure that a reconfiguration quorum intersects with every write operation. We therefore impose a requirement on the write rule that any write quorum must at least write a majority grid quorum.

We now summarize the rules. Assume a grid of dimensions $\sqrt{n} \times \sqrt{n}$, and that there are l available columns and w available rows in the current view (a row or a column is *available* if it contains at least one copy in the current view):

- Reconfiguration Rule: A reconfiguration quorum is formed from any majority grid quorum of dimensions $\langle\sqrt{n}/2+1, \sqrt{n}/2+1\rangle$. Thus in any view both l and w must be at least as large as $\sqrt{n}/2+1$.
- Write Quorum Rule: A write quorum is formed from all available copies in $\sqrt{n}/2+1$ columns in the current view. Each available column included in the write quorum must have at least $\sqrt{n}/2+1$ copies in the current view.
- Read Quorum Rule: A read quorum is formed from one copy in any $l-\sqrt{n}/2$ of the l available columns in the current view.

A reconfiguration quorum must access a majority of columns and in each column it must access a majority of copies. This guarantees that there can be at most one view at any time where an object is accessible for both reading and writing. Since a write quorum accesses a majority of columns of the base structure, it will intersect with any reconfiguration quorum on at least one column. Furthermore, in that column, the write quorum accesses at least a majority of copies and hence must intersect with any reconfiguration quorum on at least one copy. The intersection between read and write quorums in a view follows from the above quorum rules.

Consider the scenario in which all copies except a quarter of the grid are unavailable as shown in Figure 1. A reconfiguration transaction can still form a new view by accessing a reconfiguration quorum consisting of the available copies {13,14,15,18,19,20,23,24,25}. Write operations in this view will write all copies whereas read operations access any single copy. This example is one of the many possible failure scenarios in which as many as 3/4th of the total number of copies become unavailable and yet read and write operations can be executed. No other variants of the static quorum protocol (where all copies are treated equal) can tolerate as many failures.

3.3 Grid Reconfiguration Availability

In this section, we compute the availability of the proposed reconfiguration protocol for grids and compare it with the availability of read and write operations in the majority quorum protocol [Gif79, Tho79]. In this analysis we confine ourselves to site failures only. In the case of the quorum protocol, read and write quorums can be constructed as long as a majority of copies of an object are available. Let p be the probability that a copy of an object is available for constructing a quorum. Then, the data availability in the majority quorum protocol 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 integer 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 reconfiguration quorums on a grid of size $\sqrt{n} \times \sqrt{n}$ is computed as follows. Let $\sqrt{n}$ be equal to $2k+1$ for some non-negative integer k. The grid reconfiguration protocol is successful if it can construct a reconfiguration quorum on the grid of size $\langle \sqrt{n}/2+1, \sqrt{n}/2+1 \rangle$, or equivalently $\langle k+1, k+1 \rangle$. That is, the reconfiguration quorum must include a majority of copies in a majority of the columns. We compute the availability of reconfiguration quorums for grids in two stages. First, we compute the availability of a majority of copies in a single column of the grid. Let A_{mc} be the availability of a majority of copies in a single column, which can be computed as:

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

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

The next step is to compute the availability of a majority of columns that have a majority or more available copies of an object. This can be once again computed as follows, i.e.,

$$\binom{2k+1}{k+1} A_{\text{mc}}^{k+1}(1-A_{\text{mc}})^k + \ldots + A_{\text{mc}}^{2k+1}$$

Unfortunately, the above equations for the data availabilities in the two protocols do not have a closed form. In order to compare the availabilities, we illustrate the availabilities in the two protocols for specific replica configurations of an object. In addition, we depict the availabilities of read and write operations in the read-one write-all protocol[3] and in the $\sqrt{read/write}$ protocol[4]. In particular, Figure 2(a) illustrates the availabilities in the above protocols when there are nine copies of an object that are organized as a grid with three columns and three rows. Similarly, Figure 2(b) illustrates the availability with twenty-five copies that are organized as a grid with five columns and five rows.

Figure 2 clearly shows the imbalance between read and write availability when the read-one write-all protocol is used. This imbalance is partially remedied by the $\sqrt{read/write}$ protocol due to the use of a logical structure. However, as the figure shows, there still remains a significant imbalance between read and write availabilities. For example with 25 copies and when an individual copy has availability 75%, read availability is 100% whereas write availability is approximately 74%. The quorum protocol and the proposed reconfiguration based protocol in this case have read and write availability of approximately 99%. This analysis illustrates that by using the reconfiguration approach data availability in the grid protocol can be made comparable to the quorum protocol while maintaining the low cost of read operations. In particular, read operations in the grid protocol access fewer than $\sqrt{n}/2$ copies compared to $n/2$ in the quorum protocol. Note that with 9 copies, the availability of both operations in the quorum and reconfiguration based protocol becomes almost 100% when the availability of an individual copy is greater than 85%. Similarly, with 25 copies, this availability level is attained beyond copy availability of 75%. In this sense, we are able to achieve the

[3] In the read-one write-all protocol, the read availability is $1-p^n$ and the write availability is p^n, where p and n are as defined above.

[4] In the $\sqrt{read/write}$ protocol, the availability of reads is $(1-(1-p)^{\sqrt{n}})^{\sqrt{n}}$ and writes is $(1-(1-p)^{\sqrt{n}})^{\sqrt{n}} - (1-p^{\sqrt{n}} - (1-p)^{\sqrt{n}})^{\sqrt{n}}$ [CAA90].

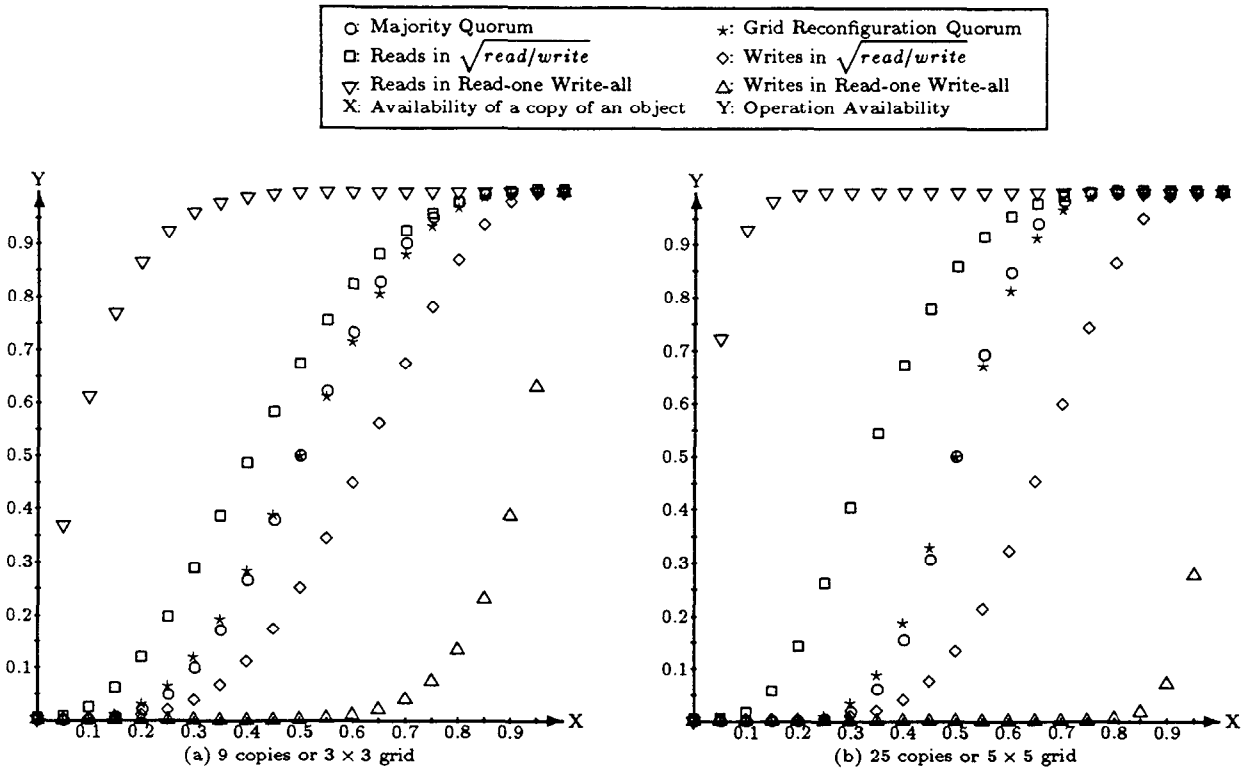

Figure 2: Comparison of the availabilities in various protocols

high degree of read availability of the read-one write-all protocol for both read and write operations using the reconfiguration based approach. This is achieved without the high cost of the read operations required by the quorum protocol.

4 The Tree Structure

In this section, we consider a logical structure based on trees. We first summarize the analogue of the read-one write-all protocol when copies of an object are logically organized as a tree. We next present a generalization of this protocol which can be used to construct read tree quorums and write tree quorums. Finally, we propose a reconfiguration protocol for trees that permits the users to use the analogue of the read-one write-all protocol in the context of trees for executing read and write operations but uses the notion of tree quorums for dynamic reconfiguration in the presence of persistent failures.

4.1 The Tree Quorum Protocol

In [AE91], Agrawal and El Abbadi proposed using a logical tree structure over a network of sites to achieve fault-tolerant distributed mutual exclusion. This protocol has a property of graceful degradation in the sense that when there are no failures mutual exclusion can be achieved by communicating with $\log n$ sites. As failures occur, the size of the quorum may increase to the maximum of $n/2 + 1$ sites. This protocol was extended to manage replicated data [AE90] by distinguishing between read and write operations. Write operations in the tree protocol must access a write quorum which is formed from the root, a majority of its children, and a majority of their children, and so forth until the leaves of the tree are reached. To ensure the quorum intersection property, a read operation tries to access the root; if the root is unavailable, the read tries to access a majority of the root's children. For each unavailable copy in this majority set, the read operation tries to access a majority of its children. It can be easily shown that read and write operations have a nonempty intersection. We will refer to this protocol as the *ReadRoot* protocol.

In Figure 3, an example of a ternary tree with thirteen copies of an object is illustrated. Note that this structure is logical, and does not have to correspond to the actual physical structure of the network connecting the sites storing the copies. A write operation could

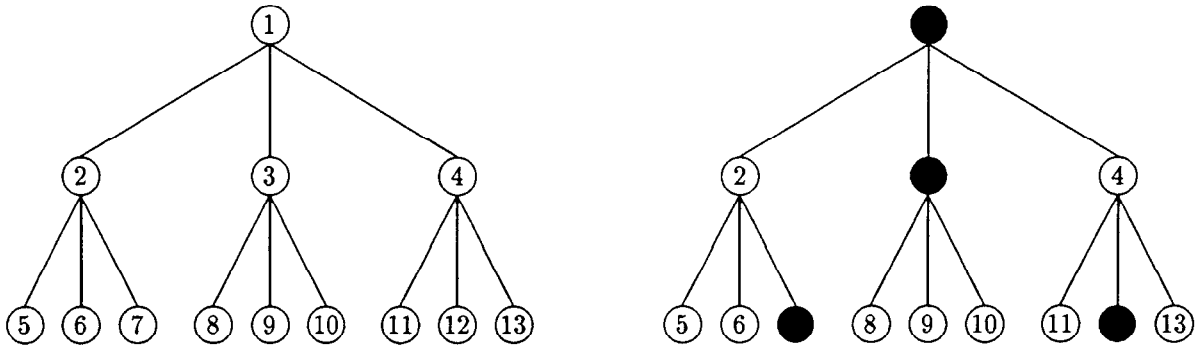

Figure 3: A tree organization with an example of a failure pattern: black circles indicate unavailable copies

be executed by writing the following set of copies only: {1,2,3,5,6,8,9}. A read operation can be executed by accessing the root in the best case. If failures occur, the set of copies accessed may be different. For example, consider a network configuration where copies 1 (the root), 2, and 3 are unavailable. 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. The desirable aspects of this protocol is that the cost of executing read operations is comparable to the read-one write-all protocol while the availability of write operations is significantly better [AE90]. Furthermore, when compared with the majority protocol, the cost of executing both read and write operations is substantially less without significantly reducing write availability. However, as is the case with the grid protocol, this protocol has some drawbacks. If more than a majority of the copies in any level of the tree become unavailable, write operations cannot be executed. For example, if the root of the tree is down, no write operations can be executed. This problem can either be solved by generalizing the tree protocol along the lines of the quorum protocol or by using dynamic reconfiguration.

We first summarize the extension of quorums in the context of the tree structure [AE92]. Given a set of n copies of an object x, we logically organize them into a tree of height h, and degree d, i.e., each node has d children, and the maximum height is h. We will assume the standard tree terminology, i.e., root, child, parent, leaf, level, etc. We also assume that the tree is complete, i.e., it has the maximum number of nodes. As is the case with the grid quorums, the tree quorums also have two dimensions associated with them, i.e., the length l which corresponds to the height of the tree and the width w which corresponds to the degree of the nodes in the tree. A tree quorum of size $\langle l, w \rangle$ is constructed as follows:

- if $l = 0$ then an empty set is a valid tree quorum;
- if $l \neq 0$ and the tree is empty then the tree quorum cannot be constructed;
- if the root is available then the root and tree quorums of size $\langle l-1, w \rangle$ recursively constructed from any w subtrees out of d subtrees of the root form a required tree quorum;
- if the root is unavailable then tree quorums of size $\langle l, w \rangle$ constructed from any w subtrees out of d subtrees of the root form a required tree quorum.

In the *ReadRoot* protocol described above, the write quorums have length h and width $d/2+1$. Similarly, the read quorum is of size $\langle 1, d/2+1 \rangle$. In order to ensure the quorum intersection property between read and write tree quorums, if the read tree quorum is of size $\langle l, w \rangle$ then the write tree quorum must be $\langle h-l+1, d-w+1 \rangle$. Similarly, the intersection between any two write tree quorums is guaranteed if the sum of the lengths of the two quorums exceeds the height h of the tree and the sum of the widths exceeds the degree d of the tree. It can be easily shown that when the read and write tree quorums have lengths and widths that satisfy the above constraints, they have a nonempty intersection.

Consider a replicated object with thirteen copies which are organized as a ternary tree of height 3 as illustrated in Figure 3. We now construct tree quorums of length 2 and width 2. In the best case, the

quorum need only contain the root and two of its children, i.e., $\{1,2,3\}$, $\{1,2,4\}$, or $\{1,3,4\}$. However, as a result of the failure of the root, a tree quorum of size $\langle 2,2\rangle$ can be formed by merging two tree quorums of size $\langle 2,2\rangle$ on any majority (two) of the root's subtrees, e.g., {2, 5, 6, 3, 8, 9} or {2, 5, 6, 4, 11, 12}. A set containing the root 1, any one of $\{2,3,4\}$, and any two children of either of the other two copies also forms a quorum. Finally, if all three children of the root are inaccessible, then a set with the root and any two children of two of the inaccessible copies form a quorum, e.g., $\{1,5,7,8,9\}$, $\{1,8,10,12,13\}$ etc. If the root and a majority or more of the root's children have failed, then a tree quorum of the required dimensions cannot be constructed.

4.2 Reconfiguring Trees

The tree quorum protocol overcomes the vulnerability of the *ReadRoot* protocol for executing write operations when a majority of copies at any level in the tree is inaccessible by increasing the length of the read tree quorums and by reducing the length of the write tree quorums. However, this approach results in read operations becoming more expensive since they cannot be executed by accessing a single copy. Instead of using the tree quorum protocol for executing user level operations, we can use the notion of dynamic reconfiguration which would allow users to continue using the basic tree protocol. We use the notion of majority tree quorums with dimensions $\langle h/2+1, d/2+1\rangle$ for reconfiguration.

Given a tree of height h with degree d for each interior node, the following rules are observed for executing operations and for reconfiguration:

- Reconfiguration Rule: A reconfiguration quorum is formed from any majority tree quorum of dimensions $\langle h/2+1, d/2+1\rangle$.
- Write Quorum Rule: Write operations are executed by requiring that tree quorums of majority or $d/2+1$ width and length involving all levels in the reconfigured tree be obtained. To ensure the intersection with reconfiguration quorums, the length of write quorum must be at least $h/2+1$.
- Read Quorum Rule: Let d_c denote the number of available children of node c in the reconfigured tree. Read operations are executed by acquiring a tree quorum in which the width at node c (when c is not included) is $d_c - d/2+1$ and length 1. To ensure correctness $d_c - d/2+1$ must be positive.

Since the sum of the lengths of read and write tree quorums exceeds the height of the reconfigured tree and the sum of their widths exceeds the degree of each node in the reconfigured tree, read and write operations are guaranteed to have a nonempty intersection. Furthermore, since write quorums are of length at least $h/2+1$, write operations are guaranteed to intersect with any reconfiguration tree quorum. Since the sum of the lengths of two reconfiguration tree quorum exceeds the height of the tree and the sum of the widths exceeds the degree, only one configuration can be formed at any time.

We now discuss the availability and performance aspects of the reconfiguration protocol with the help of an example. Consider the ternary tree of height three shown in Figure 3. In the *ReadRoot* protocol, read quorums have a length one and width two and write quorums have a length three and width two. If the copy at the root fails write operations cannot be executed unless the tree is reconfigured. If a reconfiguration transaction is executed after the failure pattern shown in Figure 3, a new view can be constructed since a reconfiguration tree quorum of length two and width two can still be constructed without the root. Read and write operations can be executed once again on this newly reconfigured tree. For example, a read quorum may comprise of any one available node from level two or no nodes from level two and any one node from {5,6,11,13}. The write tree quorum on the other hand would correspond to copies in the set {2,5,6,4,11,13}, since its length must be at least two. If we use the tree quorum protocol then in order for write operations to be fault-tolerant to the failure of the root, write quorums must have length at most $h-1$ or 2, e.g., read and write tree quorums with size $\langle 2,2\rangle$. Read and write tree quorums for the failure configuration depicted in Figure 3 must be {2,5,6,4,11,13}. Thus, read operations become significantly more restrictive and expensive in the absence of the reconfiguration protocol.

4.3 Tree Reconfiguration Availability

The availability of read and write operations in the tree quorum protocol can be computed by formulating recurrence relations for both read and write availabilities. The recurrence relation is in terms of the availabilities of these operations in the subtrees of a tree of

copies of an object. Let $A_h[l,w]$ be the availability of operations that require a tree quorum of length l and width w in a tree of height h and degree d. Thus, the availability in a tree of height $h+1$, $A_{h+1}[l,w]$, is given as:

Prob(Root up)$\times$[w subtrees available with $A_h[l-1,w]$]

+

Prob(Root down)$\times$[w subtrees available with $A_h[l,w]$]

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

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

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

Note that $A_h[l,w]=0$ for $l>h$, $A_h[0,w]=1$, and $A_1[1,w]=p$. Since the above recurrence relation involves nonlinear terms, we illustrate the operation availabilities for a specific replica configuration of an object with thirteen copies in Figures 4. The reconfiguration quorum has the dimension $\langle h/2+1, d/2+1\rangle$, i.e., $\langle 2,2\rangle$.

Figure 4 illustrates the availabilities of read and write operations in various protocols. As in Figure 2, we include read and write availabilities in the read-one write-all protocol, the majority quorum protocol, and the *ReadRoot* protocol without reconfiguration [AE90]. We again observe that there is an imbalance in availability between read and write operations that use the *ReadRoot* protocol without reconfiguration. For example with 13 copies and copy availability of 75%, read availability is almost 99% whereas write availability is approximately 50%. With reconfiguration, read and write availability in this case become approximately 92%. For copy availability greater than 85% the data availability becomes approximately 100%. Read and write operations, on the other hand, have costs comparable to the *ReadRoot* protocol. This analysis illustrates that by using the reconfiguration approach, data availability in the tree protocol can be made comparable to the majority quorum protocol while maintaining the low cost of read operations.

5 Discussion

In this paper we proposed a uniform approach for increasing the fault-tolerance of logically structured replicated objects. Compared with the unstructured approach, logical structures increase fault-tolerance while ensuring low read operation execution cost. However, the traditional imbalance between the availabilities of reads and writes in the read-one write-all protocol is "inherited" in the analogues that use logical structures. We therefore proposed exploiting the notions of quorums in the context of grid and tree structures. This was attained by extending the quorum approach to a two dimensional pair instead of the traditional one dimensional quorum assignment. These two dimensions are used to manipulate the availabilities and costs of read and write operations. The assignments that result in the lowest operation cost, however, have the least degrees of data availability (this is similar to the read-one write-all protocol). For maximal availability, our probabilistic analysis has shown that a majority structured quorum assignment provides the highest degree of data availability for both read and write operations (this corresponds to the highest availability attained by the majority assignment in unstructured data [AA89]). However, majority assignments require the highest read and write execution costs. We therefore adapted the views approach [ET89] used in unstructured replicated databases to both the grid and the tree logical structures. This provides us with comparable data availability to the majority assignment, while significantly reducing the cost of executing read operations. Reconfiguration has the added cost of the view reconfiguration transaction, which is executed whenever reconfiguration is required. Logical structures, however, provide a limited degree of fault-tolerance for write operations. Hence, for some temporary, short-term failures, no reconfiguration needs to be performed, and most read and write operations will be allowed to execute. Thus, reconfiguration will only need to be performed when failures persist and result in degraded availability for both read and write

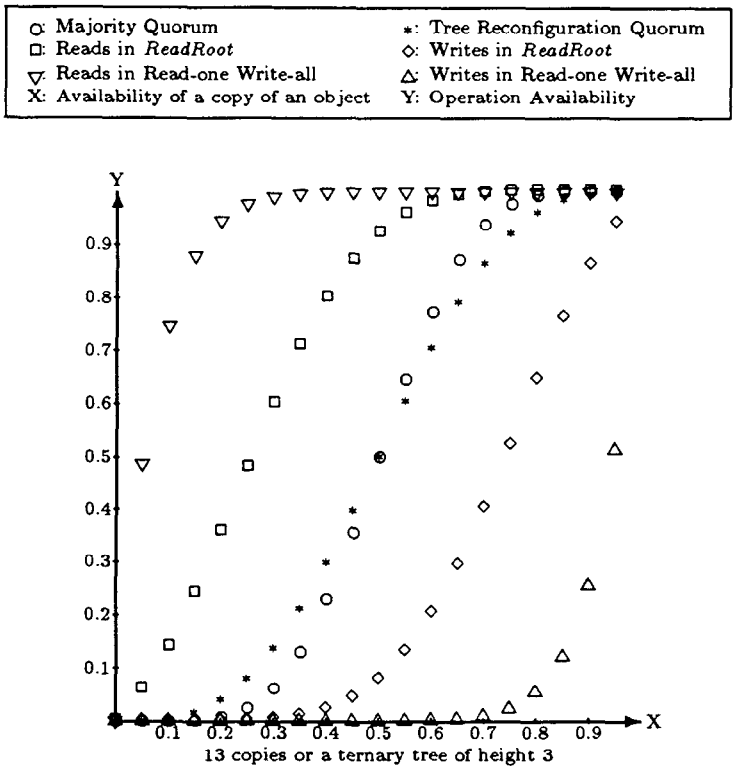

Figure 4: Comparison of the availabilities in various protocols

operations. Finally, although the results in this paper are in terms of two specific logical structures, this approach can be extended to any logical structure for which the notion of quorums and majority quorums can be defined.

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.

[AE90] D. Agrawal and A. El Abbadi. The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In *Proceedings of Sixteenth International Conference on Very Large Data Bases*, pages 243–254, August 1990.

[AE91] D. Agrawal and A. El Abbadi. An Efficient and Fault-tolerant Solution for Distributed Mutual Exclusion. *ACM Transactions on Computer Systems*, pages 1–20, February 1991.

[AE92] D. Agrawal and A. El Abbadi. The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. *ACM Transaction on Database Systems*, 1992. To appear.

[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.

[CAA90] S. Y Cheung, M. H. Ammar, and M. Ahamad. The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 438–445, January 1990.

[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.

[DGS85] 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.

[GB85] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the Association of the Computing Machinery*, 32(4):841–860, October 1985.

[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.

[Gra78] J. N. Gray. Notes on database systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer-Verlag, 1978.

[Her86] M. Herlihy. A Quorum-Consensus Replication Method for Abstract Data Types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[JM90] S. Jajodia and D. Mutchler. Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. *ACM Transactions on Database Systems*, 15(2):230–280, June 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.

[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. Schlichting 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.

[Sto79] M. Stonebraker. Concurrency Control and Consistency in Multiple Copies of Data in Distributed INGRES. *IEEE Transactions on Software Engineering*, 3(3):188–194, May 1979.

[Tho79] R. H. Thomas. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transaction on Database Systems*, 4(2):180–209, June 1979.