# Replicated Data Management in Distributed Systems*

*Mustaque Ahamad*†
*Mostafa H. Ammar*†
*Shun Yan Cheung*‡

†College of Computing,
Georgia Institute of Technology,
Atlanta, GA 30332

‡Department of Mathematics and Computer Science,
Emory University,
Atlanta, GA 30322

**Abstract**

Replication of data in a distributed system is a way to enhance the performance of applications that access the data. A system where data is replicated can provide better fault tolerance capabilities as well as improved response time. However, such improvement is achieved at the expense of having to manage replication by implementing replica control protocols. Such protocols are required to insure that data consistency is maintained in the face of system failures. In this article we describe the issues involved in maintaining the consistency of a replicated database system. We next describe three basic techniques for managing replicated data and discuss the relative merits of each technique. This is followed by a survey of extensions to the basic approaches. A discussion of future directions in research on data replication concludes our presentation.

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358, and by the University Research Committee of Emory University.

# 1 Introduction

Distributed computing systems have become common place in all types of organizations. Computers (or *nodes*) in such systems collectively have large amounts of computational and storage resources. These resources can be exploited to build highly available systems. For example, by replicating a file at nodes that have independent failure modes, a user can be allowed to access the data in the file even when some of the nodes that store copies of the file have failed. Although increased availability is a major benefit of data replication, it also provides the opportunity to share the load generated by user requests between the nodes that have copies. An access to a file requires reading of data from disk, processing and possibly writing the data back to the disk (when it is modified). If data is not replicated, all user requests that access a file must wait at a single node for the data to be read or written. When the data is replicated, load generated by the requests can be shared by nodes having the copies and hence the response time for the requests can be improved.

Data replication increases the probability that there is an operational (i.e., not failed) node that has a copy of the data when a request is made but it may not be sufficient to just locate one such node and read or write the copy stored at that node. Such a copy may be outdated because the node storing it may have failed when the data was last updated. Thus, it is necessary to use algorithms that implement rules for accessing the copies so that correctness is ensured (See section 2). These rules may disallow access to data even when its copies exist at operational nodes when it is not possible to determine if the copies at the operational nodes are up-to-date.

The algorithms that control access to replicated data are called *replica control protocols*. A *replica* refers to a single copy of the data in a system that employs replication. A large number of replica control protocols have been developed. In this paper, we present many of these protocols, explain their workings, and relate them by showing that most of the replica control protocols belong to a small number of protocol families. We also discuss both the performance enhancements made possible by the replica control protocols as well as the costs that are incurred when they are used. The two main performance measures we are concerned with are *data availability* and *response time*. Data availability is defined as the probability that there are enough functioning (or available) resources in the distributed system (nodes storing data and communication network) so that an arriving request can

be satisfied. The response time is defined as the time required for an access request to complete. Thus, response time not only depends on the failure characteristics of the system but also on the load and the I/O, processing and communication capacities in the system.

In section 2, we precisely define the correctness requirements that must be met when data is replicated. Section 3 introduces the concept of replica control. We have categorized the protocols into three basic types and their operation is described in detail in Sections 4, 5 and 6. Section 7 presents a number of methods that are improvements of the basic protocols described in earlier sections. We conclude the paper with a description of possible future directions in Section 8.

## 2 Correctness Criteria

In a replicated data system, although there exist multiple copies of a data item that can be accessed by users, it is desirable to maintain the view that there is *logically* a single copy of the data item. A replica control protocol is a synchronization layer imposed by a distributed system to hide the fact that data is replicated and to present to the users the illusion that there is only a single copy of the data. It provides for a set of rules to regulate reading and writing of the replicas, and to determine the actual value of the data. This allows replication of data to be *transparent* to users. Thus, users are not burdened with the task of implementing procedures that control access to the replicas and can exploit the benefits of high availability and improved response time without being aware of the fact that the data is replicated. Since there are correctness requirements that must be satisfied by a set of data items even when there is a single copy of each item, the same correctness requirements must also be met when the data is replicated. Thus, correctness requirements for a replicated data system are defined using correctness requirements of data when it is not replicated.

Much of the work in replicated data systems has been done in the context of distributed databases. Access requests to data in a database are modeled as *transactions*. A transaction consists of a number of related operations which read and possibly update the state of the database. When a single transaction is executed by the system at a time, its execution transforms the database from one *consistent* state into another consistent state. Thus,

serial execution of a set of transactions (i.e. one at a time) by a database that is initially in a consistent state preserves the correctness of the database and leaves it in a consistent state.

Transactions can be executed concurrently to improve performance and system utilization. When transactions do not access common data items, they do not interfere with each other and the effect of executing them concurrently is equivalent to a serial execution in an arbitrary order. However, when transactions read and write common data items, they must synchronize access to the shared items. Otherwise, the database state may become *inconsistent* and the execution of the transactions may be incorrect.

Consider for example the transactions $T_1$ and $T_2$ where $T_1$ and $T_2$ deposit \$10 and \$20, respectively, to an account with an initial balance of $\$x$. The transactions first read the value of $x$, increase it by the proper amount and write the result back to $x$. Suppose initially $x = \$100$, then the execution orders $T_1T_2$ and $T_2T_1$ will result in $x = \$130$. Both execution orderings produce the same final result. However, in an uncontrolled concurrent execution of $T_1$ and $T_2$, the read and write operations can be interleaved in the following manner: $T_1 : R(x = \$100)$, $T_2 : R(x = \$100)$, $T_1 : W(x = \$110)$ and $T_2 : W(x = \$120)$. The final value of $x$ is \$120 which does not conform with the effect produced by any serial execution of $T_1$ and $T_2$. Data inconsistency may result from concurrent executions of transactions that use common data items.

An accepted notion of correctness for concurrently executing transactions is *serializability*. An interleaved execution of transactions $T_1$, $T_2$, ..., $T_n$ is serializable if the effect of the execution is the same as *some* serial execution of the transactions. Thus, the execution of the transactions $T_1$ and $T_2$ given above is not serializable because the effect of the execution is not the same as any possible serial execution of $T_1$ and $T_2$. Synchronization schemes such as two-phase locking, timestamp ordering and optimistic concurrency control [BHG87], ensure that the effect of concurrent execution of transactions is equivalent to some serial order, i.e., the execution is serializable. Serializability can be ensured by a synchronization protocol if it guarantees that if a data item is modified by the operations of a transaction $T$, it should not be read or written by operations of other transactions until $T$ completes.

Since the goal of a replicated database system is to improve data availability and enhance response time for transactions, such a system must also allow concurrent execution

of transactions but the consistency of the data should not be compromised. In other words, the system must ensure *single-copy serializability* which means that transactions execute as if there is a single copy of the database and the transactions appear to execute in some serial order. The replica control protocols that we discuss in the following sections all satisfy this correctness requirement for replicated data.

# 3 Replica Control Protocols

In order to understand the subtleties involved in designing a replica control protocol, consider the following naive protocol for accessing the replicated data item $x$ in the system shown in Figure 1:

"Read from and write to the local replica"

Transactions $T_C$ and $T_D$ executing concurrently at nodes $C$ and $D$, respectively, each wish to read $x$, add one to the value read and write the result back. If the above protocol is used,

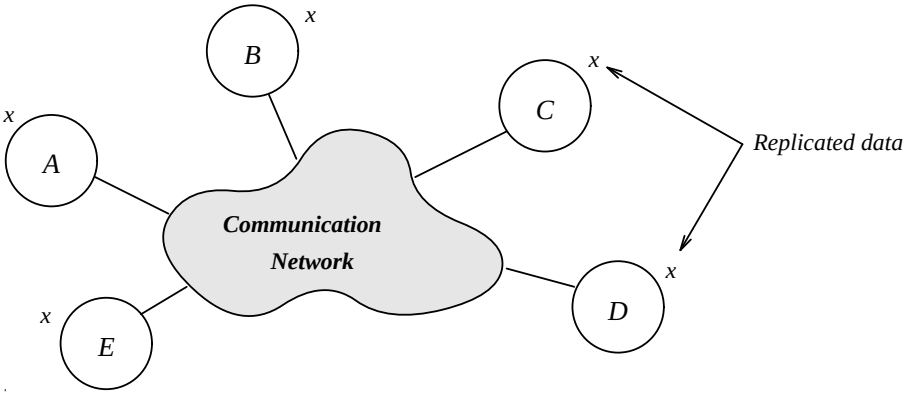


Figure 1: A distributed system with two replicas of $x$.

the final value of $x$ is only one larger than the original value at nodes $C$ and $D$ and it is not changed at other nodes. The result would be the same even if $T_C$ and $T_D$ are executed serially (e.g., at different times). The problem with this protocol is that it does not make the replicas behave as a single copy and fails to maintain data consistency because any future transaction will read an incorrect value.

A simple extension of the above protocol is the following:

"Read from local replica and write to all replicas"

If all replicas are not available for updating (e.g., because of node failure), the transaction is aborted and restarted at a later time. This is the *read one/write all* replica control method. This method will obviously maintain the consistency of the data, but it has a serious drawback. Although the data is highly available for reading, the performance of write operations will degrade (compared to the previous naive protocol) because the likelihood that all replicas are available for updating is relatively low.

In general, all replicas need not be updated by a write operation. Consider for example the following protocol:

"Read and write operations access a majority of replicas"

Since a write operation installs the new value in a majority of replicas and any two majority groups of replicas have at least one copy in common, a read majority group is guaranteed to have a replica that contains the current value. However, all replicas in the read group may not be up-to-date. In order to distinguish the current value of the data, a version number is added for identification. The version number is incremented each time the data is updated so that the current value is always associated with the highest version number. The value returned by a read operation is the one that is associated with the highest version number in the majority read group. Using this protocol a write operation does not need to update all replicas and yet read operations can determine the last value written. Furthermore, read and write operations and two write operations on the same data item will not execute concurrently because they must access a common replica. The node that manages the common replica enforces proper synchronization so that the replica is modified by a single transaction at a time. This protocol that requires operations to access a majority of replicas is a special case of the quorum consensus method which will be discussed in detail in Section 5.

Conceptually, the value of the replicated data item $x$ is a function $f$ of the values $< x_1, s_1 >, < x_2, s_2 >, \ldots, < x_N, s_N >$, where $x_i$ is the value stored at replica $i$ and $s_i$ is some supplementary information (e.g., version number) that may be stored along with $x_i$, for $i = 1, 2, \ldots, N$. The replica control protocol used will determine the function $f$

(see Figure 2). The values of some replicas may not be known because the nodes storing them can fail. Also, some replicas may contain outdated values. In general, all values $< x_1, s_1 >, < x_2, s_2 >, \ldots, < x_N, s_N >$ are not required for $f$ to determine the value of $x$. For example, the function $f$ for the read one/write all protocol is $f(< x_1, s_1 >, < x_2, s_1 > , \ldots, < x_N, s_N >) = x_i$, for any $i = 1, 2, \ldots, N$. Since all replicas are updated by a write operation, all replicas have the current value. In the read majority/write majority protocol, $f(< x_1, s_1 >, < x_2, s_2 >, \ldots, < x_N, s_N >)$ is the value of $x_i$ that is associated with the highest version number found in a majority group of replicas. The function $f$ returns an undefined value if there is an insufficient number of replicas available.

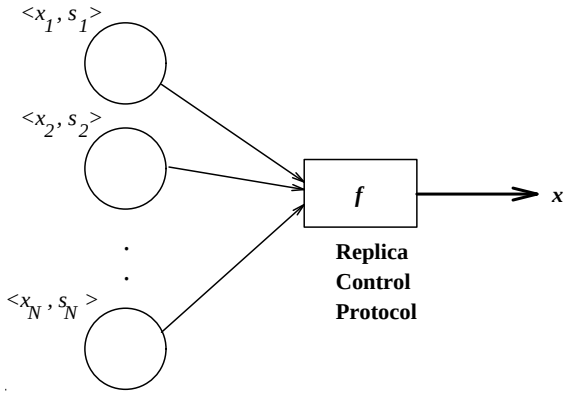


Figure 2: Determining the value of $x$ from replicas $x_1$, $x_2$, ..., $x_N$.

In the subsequent sections we present a number of protocols found in the literature. We first present the basic protocols: primary copy, quorum consensus and available copies.

# 4 Primary Copy Method

The primary copy method was described in [Sto79] and it was used in the Ingres database system for maintaining replicated data. The database is partitioned into data items called *fragments* and data items can be replicated at different nodes. The locations of all replicas are known and in addition, the nodes are arranged in a known order. Each node maintains an *up-list* of nodes that it can communicate with. This list may not accurately reflect the actual state of the system because of message delays. The replica maintained by the node that is lowest in ordering in the up-list is by definition the *primary copy* of the data item.

A transaction usually accesses a subset of all data items. When a data item is written, a write request is sent to the node that holds the primary copy. The changes are subsequently propagated to the other replicas (see Figure 3). A read request can also be sent to the primary copy, but it can be performed more efficiently on a local copy if certain conditions are met.

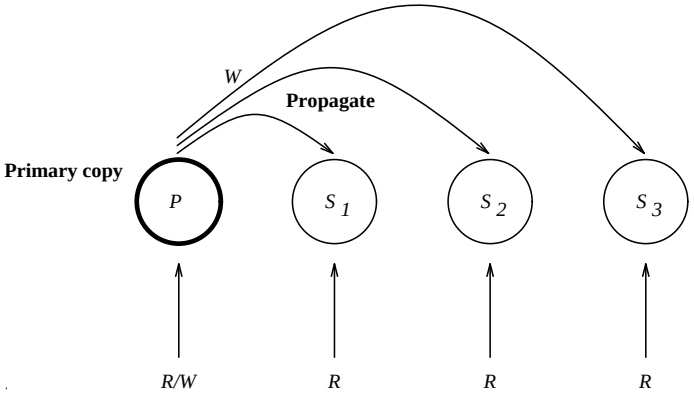


Figure 3: The primary copy method

In the primary copy method, a transaction can read from its local replica only if it accesses a single data item. This is because a transaction that accesses multiple items can find that the replicas stored locally do not conform with the current value, i.e., the state is inconsistent. This will happen when multiple items are updated by a transaction and the changes have not yet been propagated to all replicas. Consider for example an account database of a bank that is partitioned in data items with one hundred accounts per item. Accounts numbers 101 and 202 are stored in data items $x$ and $y$ respectively (see Figure 4). The primary copies of $x$ and $y$ are stored on nodes $A$ and $B$ respectively, and copies of both data items are also stored at node $C$. A transaction executing at node $C$ transfers \$10 from account 101 to account 202, and makes the changes to the primary copy of $x$ and $y$ at nodes $A$ and $B$. The changes are then propagated to the other replicas. Due to difference in propagation delay, the changes made to $x$ and $y$ do not arrive at node $C$ at the same time. Suppose the change for $y$ is incorporated first, then the replicas of $x$ and $y$ at node $C$ will reflect an inconsistent database state (\$10 has been deposited in account 202 without having been taken out of account 101).

When a node fails, it will not respond to request messages sent to it and an operational

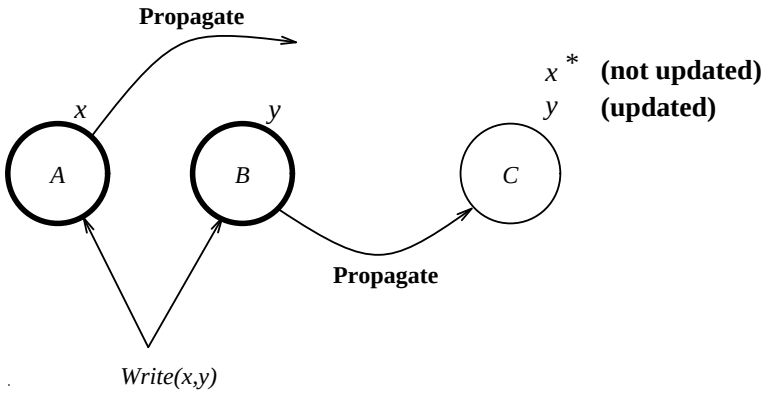


Figure 4: Propagation delay in the primary copy method

node can detect its failure through a timeout mechanism. A node that fails to receive a reply from node $x$ will remove $x$ from its up-list. If node $x$ was maintaining the primary copy, the next node in the up-list becomes the new maintainer.

Link failures can cause network partitioning. The nodes that are in the same partition as node $i$ that maintains the primary copy will continue to recognize it as the one holding the primary copy, but nodes that are separated from $i$ will think that $i$ has failed and choose a different primary copy node. There are now multiple primary copies and each one accepts different update requests. The content of the database will become inconsistent as update operations in different partitions will not be aware of each other's execution. To avoid this situation, the primary copy is allowed to exist only if a majority of all replicas of the item are accessible within the partition. Since there can only be at most one partition with a majority of replicas, this guarantees that two or more primary copies do not exist at the same time. If the network cannot partition, the primary copy scheme can allow read and write operation to access the data if at least one replica is available.

### 4.1 Performance Issues

The primary copy method was designed with the idea that the database was fragmented in such a way that most accesses made to the data originate from users at the node having the primary copy. Thus, local read and write requests can be satisfied by the primary copy without having to involve remote nodes in their processing. Although the changes must be

propagated to the other replicas, the local transaction can finish as soon as the changes are made to the primary copy.

The primary copy becomes a bottleneck in a large system because read requests from transactions at nodes that do not store a replica and all write requests are sent to it. Although read requests can be processed in parallel by nodes with replicas, the processing of write requests is done by a single node.

# 5 Quorum Consensus

In quorum consensus protocols, an operation can be allowed to proceed to completion if it can get permission from a group of nodes. A *minimal* group of nodes that can allow an operation to proceed is called a *quorum group* and the collection of all such groups is called a *quorum set*. An operation (e.g., read or write) can only proceed if it can obtain permission from nodes that have all members of a quorum group. The system requirements determine which groups are in the quorum set. For instance, operations requiring mutual exclusive access can be executed when permission is obtained from a group consisting of a majority of nodes.

The concept of quorum consensus is simple and lends itself to a wide range of applications. In the maintenance of replicated data, quorum sets are used to determine if operations for accessing the replicas can be executed. Generally, the type of operations allowed are read and write, and consequently two quorum sets are defined, $R$ and $W$, for read and write operations respectively. Read operations can be executed concurrently with other read operations but write operations must be performed in a mutually exclusive manner.

Quorum sets are used not only to synchronize conflicting operations, but also to present the single copy view to the users of the replicated data system. Due to this, the following conditions must be satisfied:

- The quorum groups that allow read and write operations to execute must have at least one common node that has a replica and such a node should not allow the replica to be accessed by concurrent operations that are conflicting. This ensures that read operations will return the value installed by the last write operation and any write

operation concurrent with the read operations is properly synchronized.

- The quorum groups that allow write operations must have a common member. This will ensure write operations are not executed concurrently.

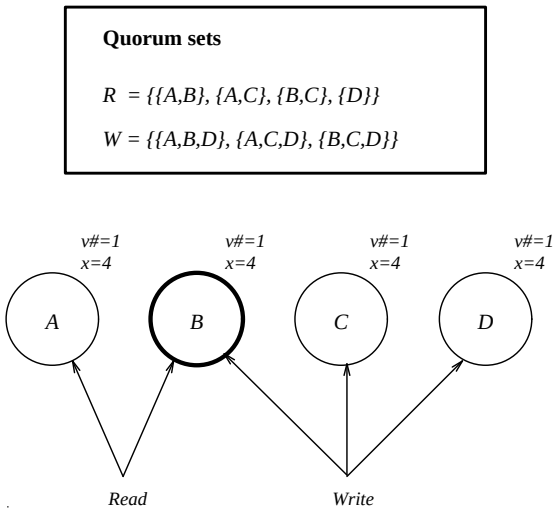


Figure 5: Example of a read and write quorum set pair

Figure 5 shows a pair of read and write quorum sets $R$ and $W$ that are used to regulate reading and writing of the replicated data item $x$. Initially the version number and value are $v\# = 1$ and $x = 4$, respectively, at all replicas. Notice in Figure 5 that any group in $R$ and any group in $W$, and any two groups in $W$ intersect. If a read and a write operation use read and write quorum group $\{A,B\}$ and $\{B,C,D\}$ respectively (see Figure 5), node $B$ will detect the conflict and will allow only one of them to proceed. If the write operation is

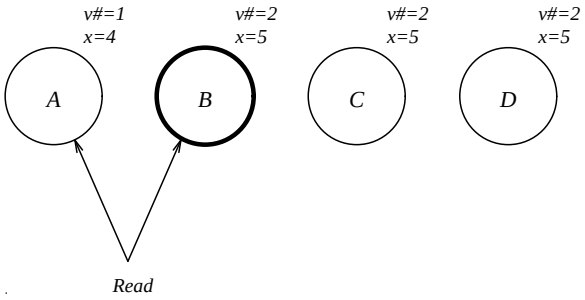


Figure 6: Reading in quorum consensus protocol

allowed to proceed before the read operation and it updates the value $x = 5$ at the replicas

in its quorum group $\{B,C,D\}$, then a subsequent read operation can detect the update using the version number. For instance, the read operation in Figure 6 that uses $\{A,B\}$ as the quorum group will find that the replica at node $B$ has the highest version number and will return the value associated with that version number as the value of $x$.

### 5.1 Voting

In general, a quorum set is specified by listing its quorum groups. However, the number of groups in a quorum set can be exponential. For instance, the number of majority groups formed with $N$ nodes is $\left( \begin{array}{c} N \\ \lceil \frac{N+1}{2} \rceil \end{array} \right)$. The order of this binomial expression is exponential in $N$. An simple way to represent a quorum set is the use of *weighted voting* [Gif79]. In this method, each node $i$ is assigned a positive integral number of votes $v_i$, for $i = 1, 2, \ldots, N$, and a quorum $q$ is defined. A group of nodes $G$ is a quorum group if the nodes in $G$ collectively have at least $q$ votes and removing any node from $G$ results in a group with less than $q$ votes. Quorum sets that can be defined by the use of a vote and quorum assignment are called *vote assignable*. Consider for example the vote assignment to four nodes where nodes $A$, $B$, $C$ and $D$ receive vote 1, 1, 1 and 2, respectively. We denote this vote assignment by the vector $\underline{v} = (1, 1, 1, 2)$. Let the quorum assignment $q$ be 2 and consider the group $\{A,B\}$. The number of votes assigned to nodes in the group is $q$ ($q = 2$) and removal of either $A$ or $B$ will result in a group with less than $q$ votes. Hence, $\{A,B\}$ is a quorum group of $q$ votes. The collection of all quorum groups of $q$ votes is $\{\{A,B\}, \{A,C\}, \{B,C\}, \{D\}\}$ which is the quorum set defined by the vote and quorum assignment $(\underline{v}, q)$. Let $r$ and $w$ be the read and write quorums, respectively. The read and write quorum groups $R$ and $W$ in Figure 5 can be defined using vote assignment $\underline{v}$ and the read and write quorums $r = 2$ and $w = 4$, respectively. In fact, a pair of vote and quorum assignment $(\underline{v}, q)$ uniquely defines a quorum set, but the same quorum set can be defined by many vote and quorum combinations. For instance, the quorum set $\{\{A,B\}, \{A,C\}, \{B,C\}, \{D\}\}$ can also be defined using $\underline{v}' = (2, 2, 2, 3)$ and $q' = 3$.

The synchronization requirements that each read and write quorum group and two different write quorum groups must have non-empty intersections are satisfied if:

- The sum of $r$ and $w$ is greater than $L$, where $L = \sum_{i=1}^{N} v_i$.

- $2w$ is greater than $L$.

The first condition will guarantee that a read quorum group will intersect with any write quorum group because the total votes in two non-intersecting groups are at most $L$ where $r + w > L$. Similarly, the second condition guarantees non-empty intersection of two write groups.

Voting is highly flexible and can be adapted for many types of systems. Consider a system with three replicas of data item $x$ using voting based replica control protocol and all replicas are assigned one vote each. In systems where most operations are read, the quorum assignment used should be ($r = 1$, $w = 3$), i.e., read one/write all. In contrast, systems that require high write availability should use ($r = 2$, $w = 2$), the read majority/write majority setting. Thus read and write quorums can be chosen so that high data availability can be provided for both types of operations.

Weighted voting is not as powerful as the general quorum consensus method. In [GB85], it was demonstrated that there exist quorum sets that cannot be defined using voting. For instance, it is shown in [GB85] that the quorum set $Q = \{\{A,B\}, \{A,C,D\}, \{A,C,E\}, \{A,D,F\}, \{A,E,F\}, \{B,C,F\}, \{B,D,E\}\}$ cannot be defined by any vote assignment.

### 5.2 Multi-Dimensional Voting

The *multi-dimensional* voting (MD-voting) technique, presented in [AAC91], can be used to represent *all* quorum sets. In MD-voting, the vote value $\underline{v_i}$ assigned to a node and the quorum are *vectors* of non-negative integers. The number of dimensions is denoted by $k$ and the votes assigned in the various dimensions are independent of each other. The quorum assignment $\underline{q_k}$ is a $k$-dimensional vector $(q_1, q_2, \ldots, q_k)$ where $q_j$ is the quorum requirement in dimension $j$, for $j = 1, 2, \ldots, k$. The vote vectors are added per dimension and compared to the quorum in the corresponding dimension. In addition, a number $\ell$, $1 \leq \ell \leq k$, is defined which is the number of dimensions of vote assignments for which the quorum must be satisfied. We denote MD-voting with quorum requirement in $\ell$ of $k$ dimensions as MD($\ell, k$)-voting and the term SD-voting (single dimensional voting) refers to the standard weighted voting method described in [Gif79]. In fact, MD(1,1)-voting is the same as SD-voting.

An MD$(\ell, k)$ vote and quorum assignment defines a unique quorum set in a similar manner as standard voting. A group of nodes $G$ is a quorum group in MD$(\ell, k)$-voting if the total votes of the nodes in $G$ collectively satisfy quorum requirements in at least $\ell$ dimensions and removing any node from $G$ results in a group that satisfies quorum requirements in strictly less than $\ell$ dimensions. The same quorum set can be defined by different vote and quorum assignment with possibly different $k$ and $\ell$. The non-vote assignable quorum set $Q$ given previously can can be defined by MD(1,4)-voting using the MD vote and quorum assignments given in Figure 7.

$$\underline{v}_A = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 2 \end{pmatrix} \underline{v}_B = \begin{pmatrix} 3 \\ 1 \\ 0 \\ 0 \end{pmatrix} \underline{v}_C = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 2 \end{pmatrix} \underline{v}_D = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \underline{v}_E = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \underline{v}_F = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \end{pmatrix} \text{ and } \underline{q} = \begin{pmatrix} 5 \\ 3 \\ 5 \\ 5 \end{pmatrix}$$

Figure 7: An MD(1,4)-vote and quorum assignment for $Q = \{\{A,B\}, \{A,C,D\}, \{A,C,E\}, \{A,D,F\}, \{A,E,F\}, \{B,C,F\}, \{B,D,E\}\}$.

### 5.3 Performance Issues

The main drawback of quorum consensus schemes is the relatively high overhead incurred in the execution of the read operations. Reading requires participation of nodes in a read quorum group which usually consists of more than one node. In contrast, reading using the primary copy method requires access only to one replica. This is also the case in the available copies method that will be discussed in the next section. Using quorum consensus, read and write operations can only succeed when a sufficiently large number of replicas are available. In contrast, the primary copies method may be able to operate as long as one replica is available for updating. To achieve the same level of data availability, the system must use a higher degree of replication in quorum consensus methods. A major benefit of quorum consensus based schemes is that arbitrary communication link failures including those that partition the network require no special attention. In case of network partitioning, write operations can be processed by at most one partition. In contrast, the primary copy scheme relies on the majority quorum consensus method for determining if a primary copy can be

established.

The system behavior depends on the pair of read and write quorum sets used. For instance, a system using the read one/write all quorum will have good performance for read operations but write operations will incur high cost and experience low data availability. At the other end of the spectrum, using the read majority/write majority quorum will provide high data availability for both types of operations but also incur a high cost of reading. The optimum read and write quorum setting depends on the mix of transactions and the performance measure in question. It was found in [AA89] that when the read one/write all quorum is used, data availability will decrease after a certain level of replication. This is due to the fact that write operations must update all replicas and the likelihood of the successful completion of a write operation decreases when the number of replicas is increased. This is not the case with the read majority/write majority quorum because data availability always increases for both read and write operations when the degree of replication is increased.

Determining the best read and write quorum set pair or vote and quorum assignment for a given performance measure is difficult due to the complex relation between system behavior and the quorum sets. However, when the quorum sets are fixed, the performance can be easily determined through analytical methods or simulation techniques. The best quorum set pair can be found through a search in the complete set of all quorum sets. In [GB85], an enumeration method is presented that generates a subset of the quorum sets used for synchronizing operations that require mutual exclusion. Each group of the quorum set must intersect with every group in the set to guarantee mutual exclusion and such sets are called *coteries*. In [CAA89], an enumeration algorithm was presented to obtain all read and write quorum sets that are defined by SD-voting.

# 6 Available Copies

## 6.1 The Basic Method

The easiest way to handle node failures is to ignore them. In the basic available copies method, updates are applied to replicas at nodes that are operational and a read operation can use any available replica. It was shown in [BHG87] that this basic method does not

guarantee data consistency. Consider a system that replicates the value of $x$ at node $A$ and $B$, and $y$ at nodes $C$ and $D$. A transaction $T_1$ first reads $x$ and then writes $y$ and a transaction $T_2$ first reads $y$ and then writes $x$. Figure 8 shows a sequence of events that will cause data inconsistency. Transaction $T_1$ and $T_2$ read $x$ at node $A$ and $y$ at node $D$, respectively. After the read operations are completed the nodes $A$ and $D$ fail. Then $T_1$ and $T_2$ proceed to update all available copies of $y$ and $x$, respectively. Since nodes $A$ and $D$ have failed, $T_1$ and $T_2$ will only update the copies at nodes $B$ and $C$. Neither transaction is aware of the read operation of the other and the execution will lead to data inconsistency. The result is not a one-copy serializable execution.

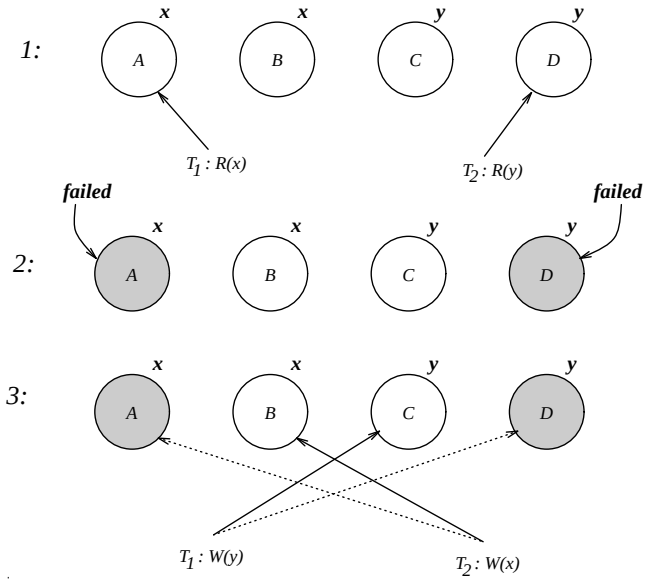


Figure 8: An incorrect execution using a naive read one/write all available copies protocol

This problem is caused by the failures of nodes $A$ and $D$ where transactions $T_1$ and $T_2$ have read the values of $x$ and $y$, respectively. If nodes $A$ and $D$ had not failed, both transactions $T_1$ and $T_2$ would not be able to update $y$ and $x$, respectively (read locks are placed on behalf of the transactions when they read data at a node). Due to node failures, transactions $T_1$ and $T_2$ can no longer synchronize themselves because their read locks are lost.

The correct available copies scheme [BHG87] operates as follows. Read operations can be directed to any node holding the latest value of the data and write operations will

only succeed if at least one replica records the update. A transaction can only terminate successfully when it is able to execute the following validation process which consists of two steps:

1. *missing writes validation*: the transaction makes sure that all replicas that did not receive its updates are still unavailable.
2. *access validation*: the transaction makes sure that all replicas that it read from and wrote to are still available.

### 6.2 The Directory-Oriented Available Copies Method

The simple available copies scheme does not allow dynamic assignment of replicas to nodes and requires that transactions attempt to update replicas at all nodes (even when some nodes have failed). The *directory-oriented* available copies method [BHG87] uses directory information to direct operations to only replicas on nodes that are believed to be operational. This scheme can also be used to dynamically add and remove replicas.

For each data item $x$, there is a directory listing $d(x)$ of the nodes that have replicas. The directory $d(x)$ can itself be replicated and stored at different nodes. The directory for $x$ at node $U$, $d_U(x)$, also contains a list of directory copies for $x$ that node $U$ believes are available. Directories are updated by two special transactions:

- Include($x_A$), for creating a new replica of $x$ at node $A$, and
- Exclude($x_A$), for destroying the replica at $A$.

To process a read operation of $x$, the system first reads a copy of the directory, say $d_U(x)$ and uses the information to find an available copy. A write operation must update all replicas that are listed in the directory entry. Due to node failures, some copies may not be available for updating and the transaction is aborted. The system then runs an Exclude transaction to update the directories and the transaction is restarted.

### 6.3 Performance Issues

An attractive feature of the available copies method is the fact that read operations only need to access one copy of the data. Also, the available copies method provides very high data availability. Both read and write operations can be performed as long as there is one operational node with a replica. However, the available copies scheme is not tolerant to network partition failures. When the network can partition, the available copies method presented above will fail to preserve data consistency. For example, when the system in Figure 8 is separated in the way given in Figure 9, transactions that read and write $x$ and $y$ in the two partitions will not be able to synchronize with each other. To handle network partitions the available copies method must be extended to ensure that write operations can be executed only in one partition.

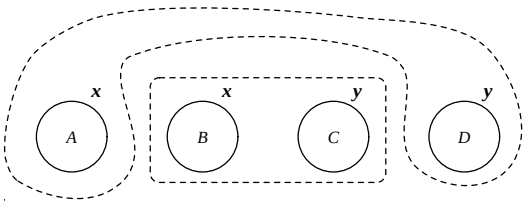


Figure 9: A partitioned network.

## 7 Extensions and Hybrid Schemes

The basic replica control protocols reviewed in the previous sections can be extended with additional provisions to further enhance the data availability provided by them or to improve other performance measures. Similarly, two of the basic schemes can be combined to exploit the advantages of both. In particular, the weighted voting method has proven to be very robust and versatile, and many dynamic voting methods have been derived [Her87, ET86, JM90]. The voting method is also used to augment the available copies scheme to achieve the capability to tolerate network failures. In this section, we review selected protocols.

### 7.1 Dynamic Quorum Adjustment

Dynamic methods can enhance performance by adjusting the rules of a replica control protocol according to the current state of the system. In systems where read operations are predominant, the dynamic quorum adjustment method [Her87] can improve performance by allowing the system to operate with a read quorum set consisting of groups with a small number of nodes. The corresponding write quorum set will have large groups and write operations can complete only when a large number of nodes are operational. Due to failures, a transaction may not be able find a write quorum and the protocol can switch to another, more favorable, pair of read and write quorum sets. The new write quorum set will have smaller groups but the groups in the corresponding read quorum set are larger. This technique is called *quorum inflation*. A complementary technique called *quorum deflation*, is used to reduce the size of read quorum groups when nodes recover. Hence, the size of read and write quorum groups increases and decreases, respectively, when the number of failures increases.

Let ($R_1 = \{\{A\}, \{B\}, \{C\}\}$, $W_1 = \{\{A,B,C\}\}$) and ($R_2 = \{\{A,B\}, \{A,C\}, \{B,C\}\}$, $W_2 = \{\{A,B\}, \{A,C\}, \{B,C\}\}$) be two pairs of read and write quorum sets. Notice that a group in $W_2$ is a subset of a group in $W_1$. Thus, a group of nodes that constitute a write quorum group in $W_1$ is also a write quorum group in $W_2$ and therefore $W_2$ is a more favorable quorum set for writing. Correspondingly, the read quorum set $R_2$ will be less favorable for reading because it will contain larger quorum groups than $R_1$. If one of the nodes fails (making the data unavailable for writing using $W_1$), the quorum inflation technique, which increases the size of read quorum groups while decreasing the size of write groups, is used to switch the system from the quorum set $(R_1, W_1)$ to $(R_2, W_2)$ in case of a failure.

In the dynamic quorum adjustment method, the system can operate at a number of levels and each level has associated read and write quorum sets (in the above example, two levels are shown). Transactions are also assigned a level number and the ones operating at the same level synchronize with each other using the read and write quorum sets defined for the level. Transactions operating at different levels are synchronized by additional read/write rules that ensure lower level transactions are completed before higher level ones. A transaction restarts by choosing a higher level number.

## 7.2 Virtual Partition

The dynamic scheme presented in [ET86] allows a node to make use of its view about the state of the system (information about operational nodes) and adjust the replica control protocol accordingly. The *view* maintained by each node $i$ is a set that contains nodes with which $i$ can communicate. The view need not accurately reflect connectivity of the nodes. Each view is associated with a unique view identifier and two nodes are in the same view if they have the same view identifier. Provisions are made to allow nodes to change their views according to changes in the system.

Each replica of a data item $x$ is assigned one vote and read and write accessibility thresholds $r[x]$ and $w[x]$ are defined. The data item $x$ is read and write accessible by nodes in a view only if $r[x]$ and $w[x]$ votes, respectively, are available at nodes in the same view. The accessibility thresholds $r[x]$ and $w[x]$ satisfy the following relations:

$$\begin{aligned} r[x] + w[x] &> N[x] \\ 2w[x] &> N[x] \end{aligned}$$

where $N[x]$ is the total number of replicas of $x$ in the system. These requirements are similar to those used in the basic quorum consensus protocol to ensure intersection of a read and a write quorum group, and two write quorum groups. However, $r[x]$ and $w[x]$ are *not* the read and write quorums.

When a view $v$ is established, the nodes in the view determine if the data is read and write accessible. In the case that the data is accessible, a read and a write quorum $r_v[x]$ and $w_v[x]$ are chosen which may be different from $r[x]$ and $[w[x]$. The quorums $r_v[x]$ and $w_v[x]$ must satisfy the following constraints:

$$\begin{aligned} r_v[x] + w_v[x] &> N[x] \\ w_v[x] &\geq w[x] \end{aligned}$$

These constraints ensure that a write quorum group in a view $v$ intersects with:

- All read quorum groups in the same view, thus read operations will be able to determine the current value.

- All write quorum groups intersect with each other, thus write operations cannot execute concurrently in two different views.

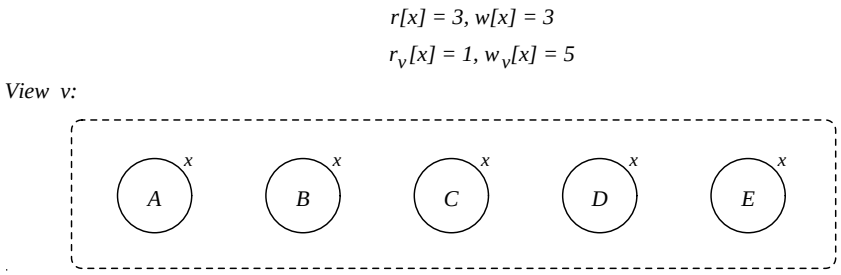


Figure 10: Virtual partition scheme: no failures operation.

For example, consider a five node system using $r[x] = 3$ and $w[x] = 3$. Figure 10 shows the system in the no failures state where all nodes have the same view $v$, and the read and write quorums are $r_v[x] = 1$ and $w_v[x] = 5$ to provide high performance for read operations. Alternately, the system can also use the quorum assignments ($r_v[x] = 2$, $w_v[x] = 4$) or ($r_v[x] = 3$, $w_v[x] = 3$) in view $v$. Suppose nodes $D$ and $E$ are separated from the other nodes as shown in Figure 11, the nodes $A$, $B$ and $C$ will change their views to $v'$. The data is read and write accessible in the $\{A,B,C\}$ partition and in this case, the only read and write quorum assignment allowed are $r_{v'}[x] = 3$ and $w_{v'}[x] = 3$. Thus, in a failure state, the system has fewer choices for quorum settings. The use of quorum assignment in conjunction with accessibility thresholds allows the system to use a favorable pair of read and write quorum sets when it is in a no failure state to improve performance. In failure states, the system can switch to use other, less favorable, quorum assignments and the data remains available.

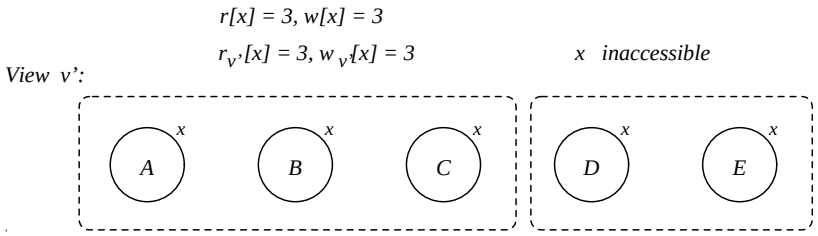


Figure 11: Virtual partition scheme: failure operation.

## 7.3 Dynamic voting

The basic dynamic voting method [JM90] assigns one vote to each replica and maintains in addition to a version number, the update node cardinality $U$ which is number of replicas updated by the last transaction that updated the data item. The version number is used to determine both the current value and the update node cardinality. Reading and writing use the majority quorum with respect to the current update cardinality, and an update is performed on replicas at all operational nodes. The operation of the dynamic voting method is best illustrated by an example. Consider the system in Figure 12 where the data item $x$ is replicated at five nodes. Initially, all nodes contain the same information. The version number $v\#$, update node cardinality $U$ and the value of $x$ are equal to 1, 5 and 4, respectively.

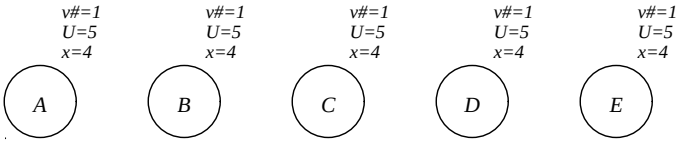


Figure 12: Dynamic voting: no failures operation.

A transaction $T_1$ that wants to access the data item $x$ first determines that the last update operation has written to 5 replicas and that the current version number is 1. If $T_1$ can obtain permission from 3 (a majority of 5) nodes with replicas that contain the current version number 1, it is allowed to proceed. Otherwise, $T_1$ must wait or abort.

Assume nodes $D$ and $E$ fail and a transaction $T_2$ wants to increment $x$ by one. $T_2$ will also determine from information at nodes $A$, $B$ and $C$ that $U = 5$ and $v\# = 1$. Since the group $\{A,B,C\}$ is a majority group with respect to the current update cardinality $U$, $T_2$ can access the data $x = 4$, add one to $x$ and update the replicas at node $A$, $B$ and $C$. The resulting state is given in Figure 13 and notice that the update node cardinality recorded is 3.

Suppose node $A$ also fails and another transaction $T_3$ wants to increment $x$ by one. $T_3$ now determines that $v\# = 2$, $U = 3$ and $x = 5$. Since the current update node cardinality $U = 3$, the group of operational nodes $\{B, C\}$ is a valid majority quorum. Notice that $\{B, C\}$ is not a majority group in the initial state where the update node cardinality is 5.

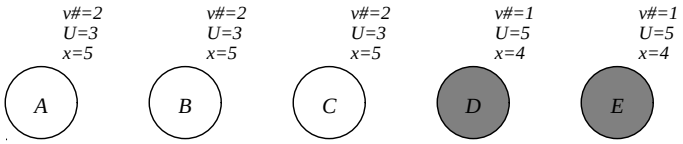


Figure 13: Dynamic voting: failure operation.

Transaction $T_3$ can also be allowed to access $x$ and record the updates at nodes $B$ and $C$. The update node cardinality is now equal to 2 and the resulting state is given in Figure 14.

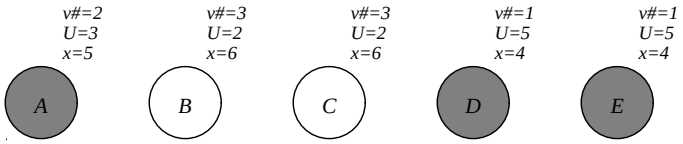


Figure 14: Dynamic voting: failure operation, continued.

The dynamic voting method can thus allow the system to adapt its quorum requirement to changes in the system state. Notice that in Figure 14, a minority of the replicas contain the most recent value. It is thus possible that a majority of the replicas are operational and none of which are holding the current value. For example, if nodes $B$ and $C$ fail, and node $A$, $D$ and $E$ recover simultaneously, we will have the state given in Figure 15 where a majority of the nodes with replicas are operational but none of which has the current value of $x$. The dynamic voting protocol will disallow access to out of date value. A transaction $T_4$ that wants to access $x$ will that $v\# = 2$ and $U = 3$, but only one node (namely $A$) has the most recent value. Nodes $D$ and $E$ which have out of date version numbers cannot be included in the quorum. $T_4$ will not be able to obtain a valid quorum and must wait or abort.

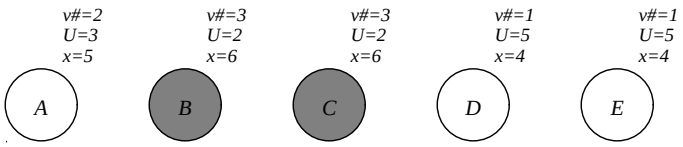


Figure 15: Dynamic voting: failure operation, continued.

The hybrid dynamic voting [JM90] method operates in a similar manner as the basic

dynamic voting scheme. It uses the basic dynamic voting method when the update node cardinality is at least three nodes. When less than three nodes with replicas are operational, the hybrid scheme will use the static majority voting as the replica control method. It was shown in [JM90] that the hybrid scheme provides better data availability than the basic dynamic voting protocol.

### 7.4 Voting with witnesses

The replicas in the basic weighted voting scheme store a version number and the value of the data item. The voting with witnesses method [Pâr86] replaces some of the replicas by *witnesses*, which are copies that only contain the version number but no data. The witnesses are assigned votes and will cast them when they receive voting requests from transactions. They provide the transactions with their version numbers which are used to identify the current value of the data item. Although the witnesses do not maintain data, they can testify about the validity of the value provided by some other replica. Due to the fact that witnesses do not contain the data item itself, the read and write operations on witnesses are implemented as follows:

- When a witness is a member of a read quorum, it provides the requester with its version number. It cannot provide the value of the data item.
- In the case that a witness is a member a write quorum, it records the version number provided in the write request. The value of the data is ignored.

Consider a system in Figure 16 with two full replicas of $x$ and one witness where initially all version numbers are one and $x = 4$. The state of a full replica and a witness can be represented by the tuple (*version number*, *value*) and (*version number*, $-$), respectively. The state of the system in Figure 16 can be represented by the triplet of tuples ((1,4), (1,4), (1, $-$)) where the first, second and third tuples represent the state of the item stored at nodes $A$, $B$ and $C$, respectively. A replica and witness are assigned one vote each and the replica control method used is read majority/write majority. Thus, a majority can be two full replicas, or one full replica and one witness.

If a transaction $T$ updates $x = 5$ using the replica at node $B$ and the witness at node $C$ as the write quorum group, the state of the system becomes ((1,4), (2,5), (2, $-$)). Notice

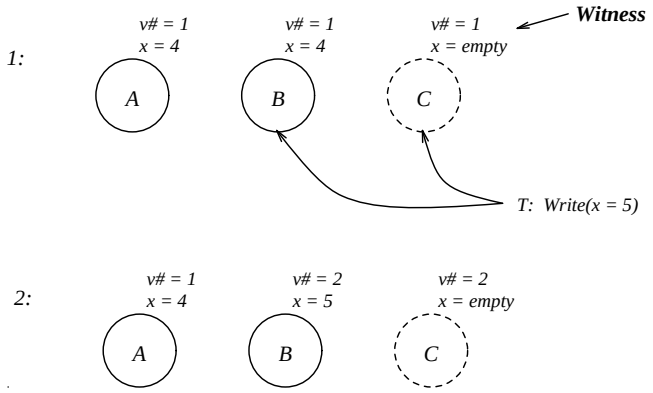


Figure 16: Voting with witnesses using two full replicas and one witness.

that the value is not stored at $C$ but its version number reflects the fact that the replica at node $A$ does not contain the current value. When a read operation uses nodes $A$ and $C$ as quorum group, the version number returned by the witness will testify that the value at node $A$ is out of date. It is thus possible for a read operation to obtain permission from a read quorum group and not find the current value of the data. An analysis in [Pâr86] showed that the data availability in a system with two full replicas and one witness is slightly lower than a system with three full replicas. The voting with witnesses scheme can effectively raise the data availability without having to replicate the data item.

## 7.5 Voting with Ghosts

A system that uses the voting with ghosts method [vRT87] is assumed to consist of a number of network segments and the nodes located in the same segment cannot be separated from one another. A network segment can fail and the nodes in that segment will not be able to communicate. This model can be used to represent the operation of Ethernet segments interconnected through gateways where gateway failures can partition the network but nodes on the same segment can communicate as long as the segment is operational.

The voting with ghosts protocol uses weighted voting as its basic replica control scheme and extends it with the notion of *ghosts* to increase the availability of write operations. A ghost is a process without any storage space and its task is to testify to the fact that a node with a replica has failed. The ghost for a failed node holding a replica is started on

the same segment of the network where the failed node is located. Hence, when a ghost responds to a request, a transaction can safely assume that the network is not partitioned and the node with the replica has failed.

Like a witness in [Pâr86], a ghost is assigned votes and it is equal to the number that was given to the replica at the failed node. Unlike a witness, a ghost does not participate in a read quorum. This difference is caused by the fact that a ghost does not maintain any information on the data item while a witness can know the latest version number. A ghost can be a member of a write quorum group and like a witness, it will ignore the value in the write operation. Furthermore, the version number in the write operation will be ignored as well. The ghost will only return its vote in response to a write request.

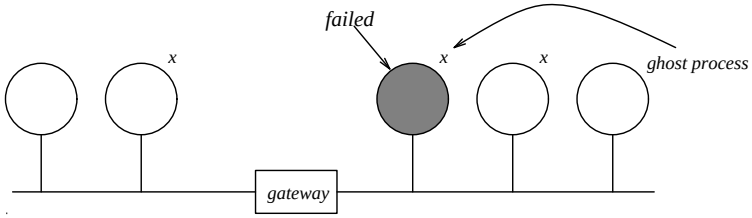


Figure 17: Voting with ghosts.

Consider a system with $N$ replicas of a data item $x$ where all replicas are assigned one vote each. Assume the read quorum is one, then the corresponding write quorum must be $N$ and a failure of any node with a replica will make the system unavailable for writing using the basic voting scheme. In voting with ghosts, the failure of a node with a replica will trigger the creation of a ghost process and it will reply on behalf of the failed node. The ghost will not respond to read requests so that a read operation will only be processed by a node holding a replica of $x$. When a write request is sent to all node with replicas, the ghost will respond on behalf of the failed node and the transaction will be able to obtain a write quorum and complete the write operation. Note that all available replicas are part of a write quorum and will be updated. When a node recovers from failure, it must obtain the latest value of the data before it can participate in a read quorum. Thus, a subsequent read operation will obtain the current value. In fact, the voting with ghosts protocol using the read one/write all quorum assignment, and the basic available copies method operate in a similar manner when the network cannot partition. Both methods will read from one replica and update all available replicas. But the voting with ghosts method is not restricted to

the use of the read one/write all assignment and can use an arbitrary vote and/or quorum assignment.

### 7.6 Regeneration

The regeneration scheme [PNP88] attempts to maintain exactly $N$ available replicas at all times. The scheme was implemented in the Eden distributed system at the University of Washington. When a node storing a replica fails, the system will try to boost the availability of the data by regenerating a new replica on another operational node. Directories that store the location of the replicas are modified accordingly to direct transactions to read and write the newly generated copies. The replicas stored at failed nodes will not be accessible through directories and effectively become garbage when the nodes recover from failures. A garbage collection subsystem is used to reclaim the space occupied by such replicas.

The basic regeneration scheme is read one/write $N$. Read operations will succeed as long as there is at least one available replica. Write operations only succeed if $N$ replicas can be written. When there are fewer than $N$ replicas available for update, the system will regenerate a number of replicas to supplement to a total of $N$ replicas. Write operations will fail if the system cannot regenerate replicas (due to shortage of disk space). The basic regeneration method is similar to the directory-oriented available copies method but uses a policy to include additional replicas when the system detects that fewer than $N$ replicas are available.

An extension to the basic regeneration scheme is to use the available copies scheme when the system has exhausted the disk space for regenerating replicas. As long as the system has space available, write operations will update exactly $N$ replicas. In the event that no more space is available, write operations will only update the available copies. Read operations will access one replica in either cases.

## 8 Future Directions

High speed networking and the falling costs of hardware will make it possible to share resources in distributed systems that span hundreds or thousands of nodes. Although the

reliability of the nodes and network components may be high in these systems, the need for data replication will still exist to ensure that data can be found locally or close to the nodes where requests for it are made so network latency costs can be avoided. Many of the replica control protocols presented in this paper have not been evaluated for such environments and it may be that they do not perform well. The following are some of the issues that must be addressed in the design and evaluation of replica control protocols suitable for large distributed systems.

- The major goal of many of the replica control protocols has been to maximize the availability of data. However, when nodes are highly reliable, a very high level of data availability can be achieved with a small degree of replication [AA89]. Clearly, the small number of nodes that store replicas can become bottlenecks in a large distributed system. Thus, a high degree of replication may be desirable to ensure that a replica can be found close to the node where a request arrives and also the load generated by the requests is shared between a large number of nodes. More work needs to be done to develop replica control protocols that exploit the load sharing benefits of replicated data.

- The communication and processing overhead of replica control protocols is high because they ensure single-copy serializability. Although this type of correctness may be necessary in some applications, there exist many application domains where a weaker correctness condition is acceptable. An example of such a domain may be file systems where users can tolerate occasional inconsistencies to get high data availability at low cost. It is necessary to develop precise characterizations of weaker consistency of data and exploit the weakening in developing efficient replica control protocols.

- There is strong evidence that the network environment has a considerable impact on the effective use of replication. An understanding of the interaction of replication and the networking environment remains to be achieved. For example, a replica control protocol based on voting can use broadcast or multicast communication to reach the nodes that have been assigned votes. Such protocols may not be suitable in networks where broadcast or multicast communication is expensive or not feasible. It may be possible to attain better performance by using a replica control protocol that exploits the mechanisms of the underlying network. It may also be that replication can be

used to overcome the effect of poor quality communication links.

- The performance of a large number of protocols has been studied either analytically or using simulation studies. Unfortunately, there are few actual implementations of these protocols in real systems. It is necessary to build these protocols and experimentally evaluate their performance for realistic application domains. Perhaps the lack of experimental research is the reason for the limited impact that data replication has had in the design and implementation of distributed systems.

Data replication in distributed systems is still an active research area and many of the above issues should be resolved in the coming years. The results of this research and the results that exist today hold the promise for exploiting the potential of distributed systems in building highly available and powerful systems.

## References


[AA89] M. Ahamad and M.H. Ammar. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Transactions on Software Engineering*, 15(4):492–496, 1989.

[AAC91] M. Ahamad, M. H. Ammar, and S. Y. Cheung. Multi-dimensional voting. *ACM Transactions on Computer Systems*, 9(4):399–431, Nov 1991.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[CAA89] S. Y. Cheung, M. Ahamad, and M. H. Ammar. Optimizing vote and quorum assignments for reading and writing replicated data. *IEEE Transactions on Knowlegde and Data Engineering*, 1(3):387–397, September 1989.

[ET86] A. El Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 240–351. ACM, 1986.

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

[Gif79] H. Gifford. Weighted voting for replicated data. In *Proceedings of 7th Symposium on Operating Systems*, pages 150–162. ACM, 1979.

[Her87] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[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, 1990.

[Pâr86] J.-F. Pâris. Voting with witnesses: A consistency scheme for replicated files. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 606–612. IEEE, 1986.

[PNP88] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its Eden implementation. *IEEE Transactions on Software Engineering*, SE-14(7):936–945, July 1988.

[Sto79] M. Stonebreaker. Concurrency control and consistency of multiple copies of data in distributed ingres. *IEEE Transactions on Software Engineering*, SE-5(3):188–194, May 1979.

[vRT87] R. van Renesse and A.S. Tanenbaum. Voting with ghosts. In *Proceedings of 7th Symposium on Distributed Computing Systems*, pages 456–461. IEEE, 1987.

## Additional Reading

The following list cites other important works in the area of data replication.

*Proceedings of the Workshop on Management of Replicated Data.* Editors: L.-F. Cabrera and J.-F. Pâris. IEEE Computer Society Press, 1990.

*Consistency in Partitioned Networks.* S. B. Davidson, H. Garcia-Molina and D. Skeen. In ACM Computing Survey, pages 341–370, 1985.

*Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems.* T. A. Joseph and K. P. Birman. In ACM Transactions on Computer Systems, pages 54–70, 1986.

*A Quorum-Consensus Replication method for Abstract Data Types.* M. Herlihy. In ACM Transactions on Computer Systems, pages 32–53, 1986.

*An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases* P.A. Bernstein and N. Goodman. In ACM Transactions on Database Systems, pages 596–615, 1984.

*Achieving Robustness in Distributed Database Systems.* D.L. Eager and K.C. Sevcik. In ACM Transactions on Database Systems, page 354–381, 1983.

*An Efficient Solution to the Distributed Mutual Exclusion Problem.* D. Agrawal and A. El-Abbadi. In the Proceedings of Principles of Distributed Computing, pages 193–200, 1989,

*Reducing Storage for Quorum Consensus Algorithms.* D. Agrawal and A. El Abbadi. In Proceedings of Very Large Databases Conference, pages 419–430, 1988.

*The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data.* S. Y. Cheung, M. H. Ammar and M. Ahamad. In the Proceedings of 6th International Conference on Data Engineering, pages 438–445, 1990.

*Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects.* A. Kumar. In the Proceedings of 10th International Conference on Distributed Computing Systems, pages 378–385, 1990.

*The Reliability of Voting Mechanisms.* D. Barbara and H. Garcia–Molina. In IEEE Transactions on Computers, pages 1197–1208, 1987.

*Protocols for Dynamic Vote Reassignment.* D. Barbara and H. Garcia–Molina and A. Spauster. In Proceedings of Principles of Distributed Computing, pages 195–205, 1986.

*A Scheme for Maintaining Consistency and Availability of Replicated Files in a Partitioned Distributed System.* J. Tang and N. Natarajan. Proceedings of the 5th International Conference on Data Engineering, pages 530–537, 1989.

*Vote Asignments in Weighted Voting Mechanisms.* Z. Tong and R. Y. Kain. In the Proceedings of the 7th Symposium on Reliable Distributed Systems, pages 138–143, 1988.

*How to Make Your Votes Count.* J.-F. Pâris and F. D. Berman. In the technical report UH-CS-99-16 of the Department of Computer Science, University of Houston, Texas, 1988.

*Voting as the Optimal Static Pessimistic Scheme for Managing Replicated Data.* M. Obradovic and P. Berman. In the Proceedings of 9th International Symposium on Reliable Distributed Systems, pages 126–134, 1990.

*On the Optimality of Voting.* by S. Y. Cheung, M. H. Ammar and M. Ahamad. In the technical report GIT-ICS-90/30 of Georgia Institute of Technology, Atlanta, 1990.

*The Performance of Available Copy Protocols for the Management of Replicated Data.* J.-F. Pâris and D. D. E. Long. In Performance Evaluation, volume 11, pages 9–30, 1990.

*The Reliability of Regeneration-Based Replica Control Protocols.* D. D. E. Long, J. L. Carroll and K. Steward. In the Proceedings of the 9th Symposium on Distributed Computing System, pages 465–473, 1989.