Protocols for Maintaining Consistency of Replicated Data

Ricardo Anido *N. C. Mendonça*

Relatório Técnico DCC–95-05

Junho de 1995

Protocols for Maintaining Consistency of Replicated Data

Ricardo Anido* N. C. Mendonça†

Abstract

Data replication is a useful technique for improving the performance and availability in distributed database systems. Most of the protocols proposed in the literature to control the access to the replicated data use some form of voting for maintaining the data consistent, and follow a *pessimistic* approach, in the sense that they prevent possibly unsafe changes to the replicated data. In this paper several of those protocols are described and compared. Some *optimistic* protocols, and some protocols not based on voting are also briefly presented.

1 Introduction

By keeping several physical copies of the same logical data distributed over different sites it is possible to benefit from all properties of distributed computing, such as better performance, improved fault tolerance, load distribution and scalability, to name a few. However, *data replication* also makes any access to the data more costly, for in order to guarantee consistency of the logical data, an update performed to any copy must be reflected in all further accesses to the data. That means that eventually all copies must be notified of the update, and some accesses may not be possible until the propagation of the new value is completed. Therefore, while replication may offer all the benefits of distributed computing, it needs a control protocol – in general not simple to implement and always somewhat costly – to synchronize all accesses to the copies, so as to maintain the logical data in a consistent state.

In this paper several of the replica control protocols found in the literature are described and compared; to facilitate their description, the protocols are grouped according to their basic characteristics. The paper is organized as follows. In the remaining of this section some basic concepts and early protocols for maintaining consistency are presented. Section 2 presents protocols based on simple voting, and section 3 presents protocols which use an underlying logical structure to provide better data availability. Some protocols not based on voting are described in section 4.

*Institut National des Télécommunications, rue Charles Fourier, 91011 Evry Cedex, France (on leave from DCC-IMECC-UNICAMP); work supported by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), grant 94/3667-3.

†Depto. C. Computação, IMECC, UNICAMP, Cx. Postal 6065, 13081-97 Campinas, SP, Brazil.

1.1 System model

The underlying system is assumed to be a collection of autonomous computers which do not share a global memory, communicating only by exchanging messages through a communication network. Each computer, also called a *node* or *site*, is composed by a processor and a local memory. The communication network is composed of *links* which connect the nodes.

Nodes and communication links are supposed to be *fail-stop*, i.e., in case of failure they simply stop working until some repairing action is taken; they do not present spurious or malicious behavior. Failure in nodes or links may cause the system to be divided into distinct groups of nodes called *partitions*; nodes from one partition can communicate with nodes from the same partition but not with nodes from other partitions. The partition of a system is a condition particularly difficult to manage in distributed systems: when a node cannot be contacted it is general impossible to determine whether the node has failed or is operational but contained in another partition. In the analysis of the protocols presented in this paper, it is considered that a user is able to communicate with each node of the system with probability ρ $(0 < \rho < 1)$. That is, at any instant there is a probability $(1 - \rho)$ that a node cannot be contacted due to node or link failures.

1.2 Data replication

One piece of information, referred as the *logical data*, is replicated by storing N copies in different nodes of the system, one copy at each node. Copies of the same logical data are also referred as *replicas*. Each copy carries a *version number*, that is incremented at every update and can be compared to determine which copy holds the most recent information. In this paper we consider that *reading* and *writing* are the only two operations performed on the replicated data, with the usual conflict definition: a write operation conflicts with any other write or read operation. However, all results are readily extended to any set of more complex operations based on the replicated object semantics, as long as conflict rules can be defined for the access operations.

Data replication is a task managed by the system administrator, who must decide how many copies to use and where to locate them. This decision depends on factors such as data demand, number of participating sites and amount of space available at each site. That information, once established, must be made available exclusively to the *replica control protocol*, which is responsible for mapping the accesses to the logical data requested by users into accesses to the physical copies.

It is assumed that each node in the system contains a *replication manager*, responsible for running the replica control protocol, as well as serving as an interface to users. To access the replicated data, a user contact its local replication manager. Depending on the replica control protocol used, the local replication manager may contact a copy, a set of copies, or other replication managers before delivering the service to the user.

The *data availability* is defined as the probability that the data can be accessed, and reflects the degree of fault-tolerance of a given protocol. In most replica control protocols the read and write operations have different data availability. The data availability for

an operation is expressed as a function the node availability ρ, and is used as a metric to compare different protocols.

1.3 Consistency

The notion of consistency considered in a replicated system comprises two aspects [Lampson et al. 1983]: *mutual consistency* and *internal consistency*. Internal consistency refers to the information contained in each replica and corresponds to the usual notions of atomicity and semantic integrity to which it is submitted non-replicated data. Mutual consistency refers to the fact that copies represent the same single logical data, and users must perceive the data consistent despite eventual disparities in the copies. The correctness criteria normally used for verifying mutual consistency is *one-copy-serializability* [Bernstein and Goodman 1983], by which a series of operations on copies of a logical data must have access to the same values as if the operations were applied to a single copy of the (non-replicated) data. When read and write are the only operations allowed, the notion of one-copy-serializability translates to the requirement that any read must obtain the value produced by the most recent write. All replica control protocols described in this paper are focused in maintaining the mutual consistency between replicas, in the assumption that the internal consistency will be maintained by some concurrency control protocol (see [Barghouti and Kaiser 1991] for a review of such protocols).

Most of the existing replica control protocols follow the strategy of only allowing a read or write operation to be performed on the replicated data if permission from certain sets of copies, called *quorums*, are previously obtained. If quorums are defined so that quorums for every two conflicting operations have at least one copy in common, it is guaranteed that (1) by using the version number a read operation can always choose the most recent copy, and (2) no two write operations will occur independently, thus excluding the possibility that copies diverge.

The number of messages exchanged by a replica control protocol is proportional to its quorum sizes. For that reason quorum sizes are also used a metric, together with data availability, to compare different replica control protocols.

Two approaches, *optimistic* and *pessimistic*, have been used by replica control protocols to deal with partition failures [Davidson et al. 1985]. In the optimistic approach, users in different partitions are allowed to read and update copies; as the failures are corrected and partitions merge, the copies are compared and procedures to make them consistent are started. Such procedures in general involve cancelling or compensating actions. In the pessimistic approach, only users from one partition, called *main partition*, are allowed to access the data. Other partitions are blocked until failures are repaired and their copies can get the correct data value from copies in the main partition.

The terms optimistic and pessimistic used to designate the two approaches refer to the expected frequency of failures and the expected time for repairing them. In the optimistic approach it is assumed that failures are infrequent and when they do occur they are promptly repaired. It is assumed also that the momentary existence of more than one value for the logical data does not cause great concern, nor does it incur in high cost correction procedures. On the other hand, the pessimist approach assumes that failures may be

frequent, the partitioning may persist for an indeterminate period of time, and consistency among copies is vital at every instant. In practice, examples of the optimistic approach are rarely found, mainly due to the infeasibility of transactions being cancelled; this is specially true for transactions which involve external actions such as delivering bank notes, or emitting air tickets. For that reason, the majority of replica control protocols in the literature follows the pessimistic approach, priorizing consistency in detriment of availability. That high proportion of pessimistic protocols is also reflected in this survey, where almost all protocols presented are pessimistic. Optimistic protocols are briefly described in section 4.2.

1.4 Early protocols

This section presents a brief review of some early protocols, which are described in an order that reflects the distribution of control in the protocol. The first ones have a more centralized control; the last ones have a more distributed control.

1.4.1 Primary copy [Stonebreaker 1979]

In this protocol, one of the copies is designated the *primary* copy, being responsible for controlling all requested accesses. A write operation is always performed on the primary copy, which propagates the new value to the other copies; a read operation can be performed on any copy, as long as the primary copy has allowed it. The primary copy allows a read operation on a copy if it has already propagated the last update to that copy. In case of system partitioning, the main partition will be the one containing the primary copy; only users in this partition can access the data.

The disadvantages of this protocol are the necessity of electing a new primary copy in case of failure of the current one, and the fact that the primary copy is a potential network bottleneck. Yet another problem arises if it is not possible distinguish the failure of a node from failure of the communication system. In this case, a communication failure may be taken for a failure of the primary copy, with a new primary copy being elected while the current one is still active. Also, failure of the primary copy may be taken for a communication failure, rendering the data unavailable.

1.4.2 ROWA [Traiger et al. 1982]

A simple approach to implement a replica control protocol is to map a write operation on the logical data into a write operation in every one of its copies. In that way, a read operation can be performed in any of the copies, preferentially in the nearest one. This approach is known as the *read-one-write-all* (or ROWA) protocol. Although simple, the ROWA protocol has the undesired characteristic of not tolerating failures of any node, since failure of a single copy would prohibit any further write operation. Read operations, on the other hand, have an optimum availability, being possible as long as at least one copy can be contacted.

1.4.3 Accessible copies [Bernstein and Goodman 1984]

This protocol modifies the ROWA protocol such that only the copies which can be accessed (i.e, copies stored in nodes which have not failed) are updated by a write operation. A list containing the copies which were operational at the last write operation must kept in all nodes and updated when necessary, to prevent a read operation to access a copy with an old value. The management of this list is assumed to be performed by the underlying operating system, which is responsible for maintaining it updated in case of failures and recoveries. Even though more robust than the ROWA protocol, the accessible copies protocol does not cope with partition failures, since it relies on the knowledge of which nodes failed and which nodes are working, and that is impossible when the system is partitioned.

1.4.4 Token [Minoura and Wiederhold 1982]

This protocol is similar to the primary copy protocol, but here the primary copy is identified by the possession of a special object called *token*. The token can be passed freely to other nodes, allowing the primary copy to change site also in the absence of failures. In case of partitioning, the main partition is the one containing the token.

Compared to the primary copy protocol, the token protocol offers a better access time, since the bottleneck problem is reduced, and the primary copy can be moved to places in the network where accesses are more frequent. However, the token may be lost due to communication failures, in which case a complex procedure for safely re-generating the token must be started.

1.4.5 Majority voting [Thomas 1979]

This was the first protocol with totally distributed control; it is also more tolerant to partition failures than the ones presented above. In this protocol, read and write operations are performed on quorums consisting of a majority ($\lfloor N/2 \rfloor + 1$) of the N copies. This guarantees that one-copy serializability is attained, and that in the case of system partitioning there will exist only one main partition. The data is available while there is a majority of copies that can be contacted within one partition.

Compared to the protocols already presented, the only disadvantage of the majority voting protocol is the penalty it imposes on read operations, previously performed in one single copy of the data. And although it is tolerant to some partition failures, there are situations in which a majority of the copies is operational, but the system is partitioned in such way that there is no partition with a majority of copies; in that case, the data is unavailable.

1.4.6 Weighted voting [Gifford 1979]

In this protocol, a certain number of votes is assigned to each copy, and quorums for read and write operations are formed by any sets of copies whose sum of votes are at least r and w, respectively. Let denote v the total number of votes assigned to the copies. To guarantee

one-copy-serializability the values of r and w must be defined such that

$$r + w > v \quad \text{(a)}$$

$$2w > v \quad \text{(b)}$$

Constraint (a) makes certain that there is a non-null intersection between any quorum for a read operation and any quorum for a write operation, thus guaranteeing that any read quorum will contain an updated copy. Constraint (b) prevents that two write operations occur independently. In case of partition, these constraints assure also that only one main partition will exist.

The flexibility in defining the values for r and w makes this protocol a generalization of some protocols described above. By assigning one vote to each copy, and by choosing $r = 1$ and $w = N$, the weighted voting protocol is made to correspond to the ROWA protocol; by choosing $r = w = \lfloor N/2 \rfloor$ it is made to correspond to the majority voting protocol. It is also possible to use this flexibility to take advantage of specific network characteristics, assigning a greater number of votes either to more reliable nodes (in an attempt to increase data availability) or to more commonly accessed nodes (in an attempt to diminish traffic on the network). Furthermore, all protocols based on voting reviewed in this paper either incorporate or can be extended to incorporate the idea of assigning more than one vote to each copy.

For a small number of copies, majority and weighted voting are effective and have the advantage of being very simple to implement. They are also the base for a variety of derived protocols, described in the next section. For that reason, the generic term *simple voting* will be used hereafter to designate these protocols.

The availability of an operation in the simple voting protocol is expressed as the probability that a quorum of q copies, out of a total of N copies, is obtained. Assuming each copy possesses one single vote and is accessible independently with probability ρ, the data availability in the simple voting protocol is given by [Ahamad and Ammar 1989]:

$$AV(N, q, \rho) = \sum_{j=q}^{N} \binom{N}{j} \rho^j (1-\rho)^{N-j}.$$

2 Protocols based on simple voting

This section describes several protocols which improve, in different aspects, the majority voting protocol. Section 2.1 presents protocols which aim at reducing the space needed for storing the replicated data; section 2.2 presents protocols which attempt to increase the data availability by adapting quorum sizes to changes in the system configuration.

2.1 Reducing the storage space

One of the costs associated with the use of replication is the larger amount of memory needed to store the replicated data: a replicated data with N copies needs N times more space

than a non-replicated data. The main purpose of the protocols presented in this section is to modify the simple voting protocol in order to benefit from the properties offered by a replicated system with N copies but occupying physically a space equivalent to only M copies ($M \leq N$).

2.1.1 Voting with witnesses [Pâris 1986]

This protocol proposes the reduction of the storage space by substituting some copies by simple records containing, instead of the data value, just the current version number. Each of these records is called a *witness*. Using such a scheme, a logical data with N copies is composed of M normal copies and $N - M$ witnesses.

As it is done with conventional copies, witnesses are assigned a specific number of votes, and participate normally in the voting process. There is however an additional constraint in the formation of quorums, namely that every quorum must contain at least one current copy of the data, in order to avoid quorums composed of only witnesses, or composed of witnesses and old copies. The need for this additional constraint is obvious, since a witness cannot be used for read or write operations, as it does not hold the value of the data.

As an example, consider a replicated data with three copies and two witnesses, each entity (copy or witness) possessing one vote. Suppose that two entities are not accessible, and the version numbers of the other three entities are 5, 5 and 8. In this scenario the version numbers of the entities which are not accessible must be equal to 8: they cannot be less than 8, since the eighth write operation must have involved at least three entities; and they cannot be greater than 8, since they would need the agreement of some other entity to perform a ninth write operation. Therefore, the possible configurations for the remaining three entities in this case are the following (witnesses are denoted by an underline):

[5, 5, 8], [5, 5, 8], [5, 5, 8], [5, 5, 8], [5, 5, 8], [5, 5, 8] and [5, 5, 8].

Three of the seven configurations do not contain an up-to-date copy, although they do contain an up-to-date witness: [5, 5, 8], [5, 5, 8] and [5, 5, 8]. In these three configurations the access to the data would be blocked due to the additional constraint on the voting process. However, in [Pâris 1986] it is presented a statistical analysis showing that such situations are relatively rare and do not have much impact on the data availability for this protocol when compared to the simple voting.

The voting with witnesses protocol produces better results when the number of copies is small. For example, suppose it is needed to replicate a very large data (the number of copies therefore must be kept to a minimum), so that the system can tolerate at least one node failure. Using the majority voting protocol three copies are needed, with read and write quorums of 2 copies, in order to provide that degree of fault tolerance. If two copies and a witness are used instead, the same degree of fault tolerance is achieved, with a significant reduction on the needed storage space. It must be noted however that using witnesses may also decrease the data availability, for example when the number of copies is large and the number of witnesses is near the size of the quorums. In that case, the probability of forming quorums composed only of witnesses, or witnesses and outdated copies, is increased.

Besides reducing the storage space, the use of witnesses can improve the response time of write operations, since updating a witness involves only incrementing its version number. Furthermore, as witness have negligible storage costs, they can be created or reallocated more easily than conventional copies. A new form of witness, which can be stored in volatile memory, was introduced as an extension to the protocol in [Pâris 1990].

2.1.2 Voting with fragmentation [Agrawal and El Abbadi 1990b]

In contrast with the voting with witness protocol, this protocol reduces the storage space by decomposing the logical data into fragments which are replicated only partially. More specifically, the logical data is divided into N fragments and the replication of these fragments is made such that

1. each fragment is stored into M nodes ($M \leq N$), and
2. each node stores M distinct fragments.

Therefore the data is distributed over N nodes but, as each fragment is copied into only M of those nodes, the total space occupied is equivalent to M copies. The set of fragments stored in a node is called a *segment*. The minimum number of segments needed to retrieve the data (that is, the number of segments that is guaranteed to contain all N fragments) is denoted by S. It is easy to verify that $S = N - M + 1$. Figure 1 shows an example in which a replicated data has three copies stored over five nodes ($M = 3$, $N = 5$, and therefore $S = 3$). Note that the data could be reconstructed using segments s_1 and s_4, or s_2 and s_5, but that is not true for all pairs of segments. On the other hand, any three segments contain all fragments and suffice to reconstruct the data.

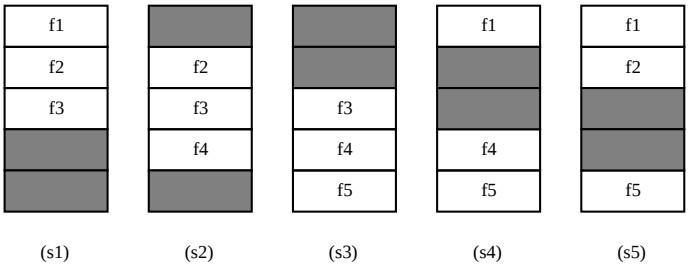

Figure 1: Fragmented voting

To implement this protocol a version number is assigned to each segment. Quorum sizes for read and write operations, denoted respectively by q_r and q_w, refer to the number of segments that must be contacted so that the operation can be performed. The values of q_r and q_w are defined with the constraints

$$S \leq q_r \leq N \quad \text{(a)}$$

$$max(S, \lfloor N/2 \rfloor + 1) \leq q_w \leq N \quad \text{(b)}$$

$$N + S \leq qr + qw \leq 2N \quad \text{(c)}$$

Constraint (a) ensures that any read operation will have access to all N data fragments. Constraint (b) guarantees that data which is updated can be completely reconstructed (by requiring that any write operation will be performed on at least S data fragments), and that two write operations cannot occur concurrently. Finally, constraint (c) guarantees that any pair of read/write quorums will have at least one segment in common, i.e., at least one copy of each of the N data fragments will be common to the two quorums; as a consequence, it is guaranteed that any read operation will be performed in up-to-date segments.

When $N/2 < M \leq N$, the voting with fragmentation protocol requires write quorums of the same size as the majority voting protocol, but read quorums comparatively larger. This is due to constraint (c), which requires at least S copies in common for any pair of read and write quorums, instead of only one common copy required in majority (single fragment) voting. That constraint can be relaxed by integrating a propagation mechanism to the system [Wuu and Bernstein 1984]. To implement the propagation mechanism, each node must maintain a *log* of events which correspond to read and write operations on the replicated data. All communication in the system is then made exclusively through (a) mapping of access requisitions, and their corresponding values, into events recorded locally in the log, and (b) interchanging log copies between the nodes. Upon receiving a copy of a log from another node, a node records in its local log the events not yet known and discard events known by all nodes (easily detected as the node receives logs from all other nodes). In that way it is possible to weaken constraint (c) so that only one segment in common is required for the intersection of read and write quorums:

$$N + 1 \leq q_r + q_w \leq 2N \qquad \text{(d)}$$

The guarantee that the data will be totally reconstructed is based on the fact that the log contains events relative to the logical data as a whole, and not only about the segment stored in a determined node. In [Agrawal and El Abbadi 1990a] it is shown that segments contacted using restriction (d) together with the information contained in the log suffice to reconstruct correctly the data.

In the worst case, the log size can increase up to the equivalent space of one whole copy of the data (when all data contents have been generated by write operations whose events are still part of the log). Therefore, when write operations are performed more frequently than log copies are exchanged, the space needed can reach N times the original data size (and in this case the protocol presents no gain compared to non-fragmented majority voting). When data is seldom modified, log sizes are smaller, and in the best case it is possible to obtain the advantages of a replicated data with N copies using the space corresponding to M times the size of the original data.

2.1.3 Voting with coded fragmentation [Agrawal and Jalote 1995]

This protocol extends the idea of voting with fragmentation by the use of a file coding scheme in which splitting and joining is done by using n vectors in a m dimensional vector space, such that any m of these n vectors are linearly independent. In the description below, a *word* is an integer in the range $0 \ldots 2^l - 1$, where l is the number of bits in a word, and all operations are in module p, for a prime integer $p > 2^l$.

The data to be replicated is arranged into a matrix FM, in which the words of the data are packed row-wise, m words in each row. The vectors of dimension m chosen for splitting the data are denoted $a_i = (a_{i1}, a_{i2}, \ldots, a_{im})$, $(1 \leq i \leq m)$. Each of these vectors is used as a column of the splitting matrix SM; i.e., $SM_{i,j} = a_{j,i}$. The fragments to be distributed are the columns of the coded file matrix $CF = FM \times SM$. That is, the ith column of CF is designated as the ith fragment. Therefore, a group of m consecutive bytes in the original data is represented as one word in each of the n fragments.

The original data can be reconstructed with at least m fragments. Let us denote $r_1, r_2, \ldots, r_m$ the fragments available. To reconstruct the data, it is necessary to construct matrix SM' such that column j of SM' is column r_j of SM. Note that SM is an $m \times m$ matrix which is invertible, since its m columns are linearly independent m-dimensional vectors. Another matrix, CF', representing the m fragments available, is contructed, such that $CF'_{ij} = CF_{ir_j}$. Therefore, since $FM \times SM' = CF'$ the data can be reconstructed:

$$FM = CF' \times SM'^{-1}.$$

Fragmenting (reconstructing) the data requires n (m) additions and n (m) multiplications per word of the original data.

The authors show that, for an (m, n, N) system, the read quorum size varies from $max(m, r)$ to $N - n' + m$, and the write quorum size varies from $max(n', w)$ to $max(n', w, N - n + n')$, where r and w are quorum sizes for read and write operations for simple voting in a system with N nodes, and n' $(n' \geq m)$ is the minimum number of nodes a write operation updates.

2.2 Increasing the data availability

Although tolerant to a certain number of failures from nodes and links, the simple voting protocol does not allow any access to the data if the minimum number of copies required in the quorums cannot be contacted. In principle, considering a replicated data with N copies, an operation which requires Q copies can be performed while there is at least Q copies that can be contacted – resulting in a tolerance to up to $N - Q$ nodes which store a copy. That is not true, however, when partitions are considered, since Q copies may be available but distributed in the partitions so that no partition includes more than $Q - 1$ copies.

The higher the quorum size, the lesser will be the probability that it can be formed in presence of failures, and therefore the data availability decreases when components fail, since quorums in that case become proportionally larger. The basic idea of the protocols reviewed in this section is to alter dynamically the size of the quorums when failures are detected in order to prevent situations in which the data is unavailable due to system partition.

2.2.1 Virtual partitions [El Abbadi et al. 1985]

This protocol extends the accessible copies protocol (section 1.4.3) to include a layer to manage changes in the system configuration. It uses a voting mechanism to cope with partition failures and provides an efficient read operation by maintaining converging views about which nodes are operational within the current partition.

The protocol is composed of two layers: a layer which manages the views shared by the nodes (or Virtual Partitions, VPs) and a layer which manages the copies. The layer which manages the Virtual Partitions is responsible for maintaining a consistent view of which nodes can communicate. The layer which manages the copies is responsible for controlling the access to the replicated data and to map the operations on the logical data into accesses to the copies.

When a node detects any change in the system configuration (caused by failure or recovery of a component), it initiates the first phase of a protocol to create a new VP, sending to all other nodes an invitation to join the new VP. If a majority of the nodes answers to the invitation, the initiator starts the second phase of the VP creation, sending the attributes of the new VP to the nodes which answered. The attributes of a VP are: (i) an identifier, generated uniquely by the initiator, which serves to ascertain that each node will join only one VP at a time; (b) a set of potential members, composed by the set of nodes which answered to the invitation; and (c) a set of actual members of the VP (initially, the potential members set and actual members set are equal).

The VP identifier and the potential members set are static, established in the first phase of the protocol, being the same to all potential members. The set of actual members is dynamic and, due to failures, may vary from one node to another. A potential member may leave the actual members set and try to initiate a new VP when it suspects that some change in the topology has occurred. Changes in the current topology are detected by another protocol in the VP management layer, which send messages probes periodically.

The protocols in the copy management layer are responsible for verifying the data *accessibility*. A data is considered accessible to node i if a majority of copies is stored in nodes included in the same VP as i. Note that the verification of accessibility is performed without any external communication. The mapping of logical operations into physical operations follows rules similar to the ones in the ROWA protocol: to perform a read operation, a user at node i verifies if the data is accessible, and in that case it sends a read request to a copy stored in any node from the same VP as i. To perform a write operation, a user at node i verifies if the data is accessible, and in that case it sends a write request to all copies stored in nodes included in the same VP as i.

The guarantee that only one actual partition will proclaim itself the main partition comes from the fact that it is necessary a majority of nodes for the creation of a new VP and for the data to be considered accessible in a VP.

2.2.2 Generalized virtual partitions [El Abbadi and Toueg 1989]

This protocol modifies the previous one by eliminating the VP management layer and by generalizing the process of forming the quorums. Let v denote a VP; in the protocol with generalized virtual partitions the replicated data is considered accessible for reading or writing in v if A_r or A_w copies, respectively, are stored in nodes included in v. These values, called *accessibility limits*, are determined following the constraint $A_r + A_w > N$, where N is the total number of copies. This restriction guarantees that any set of copies used in a read access always contains an up-to-date copy. Notice that by eliminating the constraint $2A_w > N$ it is possible that the data is updated in more than one partition. For

each VP v a read quorum $q_r(v)$ and a write quorum $q_w(v)$ are defined. Those quorums are used in mapping logical operations into physical accesses to copies, and must satisfy the following relations (where $n(v)$ denotes the number of nodes belonging to v which contain a copy of the data):

$$q_r(v) + q_w(v) > n(v) \quad \text{(a)}$$

$$2q_w(v) > n(v) \quad \text{(b)}$$

$$1 \leq q_r(v) \leq n(v) \quad \text{(c)}$$

$$A_w \leq q_w(v) \leq n(v) \quad \text{(d)}$$

Constraints (a) and (b) guarantee the copy consistency within a VP; constraints (c) and (d) make the quorum sizes defined for a VP compatible with the accessibility limits for the data. Notice that this generalized version of the virtual partitions protocol corresponds to the original protocol when $A_r = A_w = \lfloor N/2 \rfloor + 1$, $q_r = 1$ and $q_w = n(v)$.

The elimination of the VP management layer present in the original protocol comes from the realization that the two phase protocol used for the creation of a new PV is redundant, in the sense that any available transaction facility can be used instead. Therefore, VP are treated as normal objects, subject to concurrency control, and are created using specific transactions. A transaction for creating a VP is responsible for inviting the other nodes and, in case a majority of nodes agree in creating a new VP, establish the attributes of the new VP and send it to the potential members.

Three approaches are suggested to determine when to create a new VP: a *greedy* approach, in which a new VP is created as soon a change in the network topology is detected; an *object-driven* approach, in which a new VP is created only if high priority objects currently unavailable will be available in the new VP; and a *demand-driven* approach, in which a new VP is created only if high priority transactions that cannot be run in the current configuration will be able to run in the new VP.

2.2.3 Dynamic vote assignment [Barbara et al. 1989]

In this protocol, when failures are detected the votes assigned to copies in the main partition are modified, in an attempt to diminish the probability that new quorums cannot be formed in case further failures occur. Suppose a replicated data with four copies A, B, C and D, having initially the distribution of votes $v_A = v_B = v_C = 1$ and $v_D = 2$. Under this vote distribution, any set of three copies, or any set of two copies where one of the copies is D, has a majority of votes and therefore can be used as a quorum. Consider that the communication network fails so that the system is divided in two partitions {A, B, C} and {D}. At this point, partition {A, B, C} holds a majority of votes and is the main partition. If a new failure causes copy C to be isolated in a new partition, none of the three partitions {A, B}, {C}, and {D} holds a majority of votes and the data will be blocked. This blocking situation could be prevented, however, if the vote assignment in the main partition were modified before the second partition failure occurred. For example, the new vote distribution could be $v_A = v_B = v_C = 5$ (v_D would remain unchanged as D does not belong to the main partition). In that case, after the second partition failure {A, B} would still have a majority of votes (10 over a total of 17), and blocking of the data would have

been avoided. The new main partition {A, B} could again make a new vote distribution $v_A = 15$ and $v_B = 5$, which would tolerate a further partition failure that isolated all four copies – the main partition would then be {A}, with 15 votes out of 27.

This dynamic vote assignment may be performed in two ways: *majority consensus*, in which all nodes in the main partition agree on a new vote distribution, and *autonomous*, in which each node decides when and how to change its own number of votes, independently of other nodes. In this paper only the autonomous approach is presented. The majority consensus approach can be implemented by electing a coordinator (see [Garcia-Molina 1982] for election algorithms) which, from information sent by other nodes about their view on the current topology, can choose an optimum vote distribution [Barbara and Garcia-Molina 1987; Kumar and Segev 1988; Cheung et al. 1989], a topic that is outside the scope of this survey.

Two protocols for autonomous dynamic vote assignment are presented. In the first protocol, the nodes can only increase their votes; in the second, nodes can either increase or decrease their votes. A node which decides to change its number of votes is called the *initiator*; the other nodes, which will be informed of the change by the initiator, are called *participants*. These protocols must be executed when it is detected that a node has left or re-joined the current main partition. When a node leaves the main partition the remaining nodes must increase their votes; let i be the excluded node and v_i the number of votes of the copy stored in i. In order to guarantee that further partition failures will be tolerated, the number of votes of the main partition must be increased by at least $2v_i$ votes: v_i to compensate for the votes lost by the exit of i plus v_i votes to cover for the increase in the total number of assigned votes. Similarly, when a node re-joins the main partition, it must increase its number of votes, or the other nodes in the main partition must decrease their own votes.

Protocol to increase the number of votes

In this protocol, each node maintains an array V containing the number of votes of all copies of the data (including its own). The protocol for the initiator and participants is as follows:

Initiator: Demands the number of votes from other nodes and updates v with values from the arrays received. If it obtains a majority of answers (under the current vote assignment), it sends the new (increased) number of votes of its own copy to the participants, and waits a confirmation message. It the coordinator obtains confirmation from a majority of participants (still under the current vote assignment), it makes the change in its number of votes permanent in v.

Participant: Upon request, send its current array V to the initiator; record the change of votes of the initiator in V and send a confirmation.

Protocol to increase and diminish the number of votes

The protocol is divided in two parts, one to perform the increase in the number of votes, another to perform its decrease. This protocol requires that every component in the array of votes V contains, besides the number of votes, a corresponding *vote version number*. The vote version number is used by the initiator to determine which information is more recent

when its number of votes assigned to a node j differs from the information received from some other node. When a node decides to increase its number of votes it does the following:

Initiator: Demands the number of votes from other nodes and updates V with the arrays received. If it obtains a majority of answers (under the current vote assignment), it sends the new (increased) number of votes of its own copy to the participants, and waits a confirmation message. It the coordinator obtains confirmation from a majority of participants (still under the current vote assignment), it increments by one its version number and makes the change in its number of votes permanent in V.

Participant: Upon request, sends its current array V to the initiator. When it receives the new number of votes, it increments by one the version number of the initiator, records the change of votes in v and sends a confirmation.

To decrease its number of votes a nodes does as follows:

Initiator: Demands the number of votes from other nodes and updates V with the arrays received. Sends the new (decreased) number of votes of its own copy to the participants, increments by one its version number and changes its number of votes in V.

Participant: Upon request, sends its current array V to the initiator; records the change of votes of the initiator, together with its version number, in V.

It must be noted that the correct formation of quorums while the number of votes is changing dynamically is guaranteed by the above protocols, as shown in [Barbara et al. 1989].

2.2.4 Dynamic voting [Jajodia and Mutchler 1990]

In this protocol a quorum is formed by a majority of the 'current copies', i.e., a majority of the copies which participated in the last write operation. As the number of current copies may vary due to failures, the quorum size is dynamically modified according to the evolution of changes in the system configuration.

Two additional attributes, besides the version number and number of votes, are associated to each copy: the primary node and the cardinality. The *cardinality* expresses the total number of votes of nodes which participated in the most recent write operation on the data (the total number of nodes if nodes hold one vote each). The *primary node* identifies the node which will be used as a tie-break when the current main partition is divided in two partitions with the same number of votes. In that case, the partition which includes the primary node will be considered the main partition (the primary node is obviously expendable when the cardinality of the main partition is odd). The primary node may be determined by an election or, wherever possible, by imposing a linear order on the nodes.

The protocol will be presented through an example. Suppose a replicated data composed of five copies stored in nodes A, B, C, D and E, with nodes being assigned one vote each. The nodes are ordered by their names (A > B > C > D > E), so that the primary node

can be easily determined when necessary. The table in Figure 2a represents the system configuration at a point where no failures have occurred.

At this point, consider that node C starts a write operation and determines that it can only communicate with nodes D and E. Since C belongs to the main partition (which includes three copies of the last five that participated in the last write operation), the quorum is obtained, and the new configuration is shown in Figure 2b.

Node	A	B	C	D	E
Version Number	1	1	1	1	1
Cardinality	5	5	5	5	5
Primary Node	–	–	–	–	–

(*a*)

Node	A	B	C	D	E
Version Number	1	1	2	2	2
Cardinality	5	5	3	3	3
Primary Node	–	–	–	–	–

(*b*)

Node	A	B	C	D	E
Version Number	1	1	3	3	2
Cardinality	5	5	2	2	3
Primary Node	–	–	C	C	–

(*c*)

Node	A	B	C	D	E
Version Number	1	1	7	7	2
Cardinality	5	5	2	2	3
Primary Node	–	–	C	C	–

(*d*)

Figure 2: Example of dynamic voting

Suppose node C initiates another write operation and detects that now it can only communicate with node D. As C and D form a majority in relation to the last performed write operation, the quorum is obtained and the operation can be performed. Note that in the simple voting protocol that would not be the case, since C and D do not hold a majority over the total number of copies. The configuration changes again, but notice that in this case, as the cardinality is even, a primary node is needed, so node C is chosen based on the ordering previously agreed, as shown in Figure 2c. Suppose nodes C and D proceed to change the data four more times, and then a new partition failure is detected, leading to the configuration of Figure 2d. In that case, the partition composed uniquely by node C can still access the data.

This example shows that failures in nodes and communication links may occur in such a way that the dynamic voting protocol can still perform operations which would be blocked by normal voting protocols. However, the opposite is also true. Resuming the previous example, suppose node C fails and the remaining nodes are re-grouped into one single partition. In that case, the dynamic voting protocol would block any access to the data, while the simple voting protocol would not, since the newly formed partition {A, B, D, E} has a majority of votes. In [Jajodia and Mutchler 1990] it is shown, by stochastic analysis, that in general the dynamic voting protocol provides better availability than the simple voting protocol.

Finally, two approaches are suggested to allow the use of different sizes for read and write quorums: storing in every node a table with the best values for quorum sizes for each possible cardinality, or letting the quorum sizes be assigned by the node which initiates the

write operation.

3 Protocols based on logical structures

This section describes several replica control protocols whose quorum sizes are smaller than those required by the weighted voting protocol. The basic idea behind these protocols is to organize the copies into a logical structure, such as a grid or a tree, and then derive quorums from the position of the copies in the structure. In general, quorums obtained this way cannot be obtained by any vote distribution in the simple voting protocol and their quorum sizes do not grow linearly with the increase in the number of copies.

As an example of using a structure to derive smaller quorums suppose the copies are organized into a logical ring. To make the example simpler, suppose also that the total number of copies, N, is even, and that the copies are numbered consecutively in the ring from 1 to N, as shown in Figure 3. Define a write quorum as being either (i) the set of all odd-numbered copies plus any even-numbered copy, or (ii) the set of all even-numbered copies plus any odd-numbered copy. In that case, a read quorum can be defined as any two adjacent copies in the ring. Note that a write quorum has a non-null intersection with any other write or read quorums, thus guaranteeing one-copy-serializability.

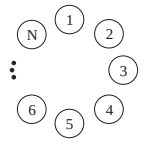

Figure 3: Copies logically organized in a ring

Compared to the majority voting protocol, in which both read and write quorums are of size $\lfloor N/2 \rfloor + 1$, this 'ring protocol' offers a good alternative: the read quorum size is fixed at only two copies, independently on the value of N, and the write quorum has the same size as in the majority voting protocol. However, as it is the case with all protocols based on logical structures, the smaller read quorum size for this protocol incurs some penalty on the data availability. In the best case the write operation tolerates the same number of failures as the majority voting protocol, that is, $\lfloor N/2 \rfloor$ copies. But in the worst case two failures (one odd-numbered copy and one even-numbered copy) suffice to block any further write operations.

This 'ring protocol' exemplifies the basic characteristic of the protocols presented in this section. These protocols use different structures and different voting rules to offer several forms of compromise between quorum sizes and data availability.

3.1 Grid quorum [Cheung et al. 1992]

In this protocol, the N copies of the data are organized into a rectangular grid with m lines and n columns. Figure 4 shows an example with 12 copies organized into a 3×4 grid.

A read quorum is formed by at least one copy from every column of the grid, and a write quorum is formed by a read quorum plus all copies from at least one column. It is

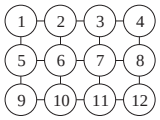

Figure 4: Copies logically organized in a grid

easy to verify that a write quorum has a non-null intersection with any other write or read quorum. Examples of read quorums for the grid shown in Figure 4 are {1, 6, 3, 12} and {9, 10, 11, 4}; examples of write quorums are {1,5,9,6,3,12} and {5, 2, 7, 4, 8, 12}.

Read quorums have size m and write quorums have size $m + n - 1$. When $m \approx n$, read and write quorum sizes are approximately $\sqrt{N}$ and $2\sqrt{N} - 1$, respectively. Considering that each node is independently accessible with probability ρ, the data availability for read operations is given by

$$A_r(\rho) = [1 - (1 - \rho)^m]^n,$$

where $(1-\rho)^m$ is the probability that all copies in a column are inaccessible, and $1-(1-\rho)^m$ is the probability that at least one copy in a column is accessible.

The data availability for write operations is calculated by subtracting, from the probability of forming a read quorum, the probability that at least one copy, but not all copies, is accessible in all columns:

$$A_w = [1 - (1 - \rho)^m]^n - [1 - \rho^m - (1 - \rho)^m]^n.$$

In [Kumar et al. 1993] the grid protocol was extended in two ways. Firstly, in the new protocol some positions in the grid are allowed to be left unoccupied. That is, the structure used is an incomplete grid. Secondly, in the extended protocol a read quorum is formed either by one copy from every column (as in the original grid protocol) or with all copies from one column.

The advantage of the incomplete grid protocol is that, besides providing better availability (particularly for read operations), it permits replicated data with a greater choice of total number of copies.

3.2 Hierarchical voting [Kumar 1991]

This protocol uses a tree structure to represent a hierarchy of groups of copies. Vertices at the lowest level (level 0) correspond to copies of the replicated data; vertices at higher levels represent groups of vertices of the level immediately below. In general, a vertex at level i ($1 \leq i \leq m$) in the hierarchy represents a logical group of l_i vertices from level $i-1$; the total number of copies in the structure is therefore $\prod_{i=1}^{m} l_i$. Figure 5 shows an example of a two-level hierarchy ($l_1 = l_2 = 3$) with 9 copies at level 0.

A read quorum is formed if permission is obtained from the root vertex at the top level m. The root vertex will grant permission if it can obtain permission from at least r_m of its children at level $m-1$; each of those children will in turn grant permission it can obtain permission from at least r_{m-1} of its own children at level $m-2$, and so on, until permissions

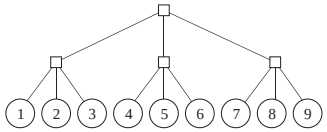

Figure 5: Copies logically organized in a two-level hierarchy

are granted directly by the copies at level 0. Read access permission is then propagated bottom-up until it reaches the root vertex, resulting in a read quorum of size $\prod_{i=1}^{m} r_i$.

A write quorum is formed in a similar way, defining for each level i the minimum number of w_i vertices from level $i-1$ which must grant permission in order that the permission is granted at level i.

So that quorums of conflicting operations intercept each other, the values of r_i and w_i for each level i ($1 \leq i \leq m$) in the hierarchy must be constrained in a similar way to the weighed voting protocol:

$$r_i + w_i > l_i \qquad \text{(a)}$$

$$2w_i > l_i \qquad \text{(b)}$$

Choosing $r_1 = 1$, $r_2 = 2$, $w_1 = 3$ and $w_2 = 2$ for the two-level hierarchy shown in Figure 5, examples of read quorums are $\{1, 4\}$, $\{6, 7\}$ and $\{2, 8\}$, and examples of write quorums are $\{1, 2, 3, 4, 5, 6\}$, $\{1, 2, 3, 7, 8, 9\}$ and $\{4, 5, 6, 7, 8, 9\}$. Choosing $r_1 = r_2 = w_1 = w_2 = 2$, examples of both read and write quorums are $\{1, 2, 4, 5\}$, $\{2, 3, 7, 8\}$ and $\{5, 6, 7, 9\}$.

In [Kumar 1991] it is shown that the greater the number of levels in the hierarchy the smaller the resulting quorum sizes. For the number of levels in the hierarchy to be the largest possible, the number of children must be kept to a minimum. As the smaller number of vertices that can satisfy constraints (a) and (b) and still tolerate a failure is three, the ideal structure is a hierarchy where $l_i = 3$ and $r_i = w_i = 2$ for all levels. In this ternary tree, $m = \log_3 N$, and both read and write quorums have size 2^m; as $2^{\log_3 N} = N^{\log_3 2}$, both quorum sizes are $O(N^{0.63})$.

The availability for the hierarchical voting protocol is calculated by generalizing the availability of the simple voting protocol to all levels of the hierarchy. Therefore, the availability of the read operation is given by

$$A_r(i) = \begin{cases} \rho & (i = 0) \\ AV(l_i, r_i, A_r(i-1)) & (i \geq 1) \end{cases}$$

Similarly, the data availability for a write operation is given by

$$A_w(i) = \begin{cases} \rho & (i = 0) \\ AV(l_i, w_i, A_w(i-1)) & (i \geq 1) \end{cases}$$

3.3 Hierarchical grid [Kumar and Cheung 1991]

One problem in the grid quorum protocol is the low availability for write operations when the total number of copies N is large, since the probability of obtaining permission from all copies in one column decreases fast when the number of copies in the columns increases. To lessen that problem, the hierarchical grid protocol organizes the copies into a structure which combines the idea of the grid quorum and the hierarchical voting. In that way, the hierarchical grid protocol increases the data availability for write operations maintaining the quorum sizes $O(\sqrt{N})$.

Similar to hierarchical voting, the hierarchical grid protocol organizes the copies into a hierarchy of l levels in which the copies are the objects at the lowest level (level 0), and the objects at higher levels are groups of objects at the level immediately below. A logical object at level 1 is defined as a grid $m_1 \times n_1$ composed of $m_1.n_1$ copies of the replicated data. In general, a logical object at level i ($1 \leq i \leq l$) represents a grid of dimensions $m_i \times n_i$ composed of $m_i.n_i$ objects from level $i-1$. The total number the copies in the structure is therefore given by $\prod_{i=1}^{l} m_i.n_i$. Figure 6 shows an example of a replicated data with 16 copies organized into a hierarchical grid where $m_1 = n_1 = m_2 = n_2 = 2$.

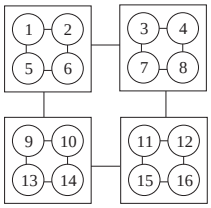

Figure 6: Copies logically organized in a two-level grid hierarchy

Besides the traditional read and write operations, the hierarchical grid protocol defines a third operation on the data, called *blind-write*. A blind-write operation conflicts with any other read or write operation, but does not conflict with another blind-write operation. This new operation is actually never performed on the data, being introduced only as a way of defining the quorums for write operations.

A read quorum is formed by obtaining, at the top level l in the hierarchical grid, read access permission from at least one object at level $l-1$ from each of the columns of the grid $m_l \times n_l$. Similar to hierarchical voting, obtaining read permission from an object at level $l-1$ means forming a read quorum in the grid of dimensions $m_{l-1} \times n_{l-1}$ corresponding to that level. In general, obtaining read access permission from an object at level i ($1 \leq i \leq l$) in the hierarchical grid consists of obtaining the same permission from at least one object of level $i-1$ of each of the columns in the grid of size $m_i \times n_i$ corresponding to the object at level i. At the lowest level, 0, the access permission is given directly by the copies. In the hierarchical grid shown in Figure 6 the sets {1, 6, 7, 8}, {1, 2, 11, 12} and {9, 14, 15, 16} are examples of read quorums.

A blind-write quorum is also formed recursively, level by level, by obtaining blind-write access permission at level $l-1$ from all objects of some column in the grid $m_l \times n_l$ of level l. In the hierarchical grid of Figure 6 {1, 5, 10, 14}, {3, 7, 11, 15} and {2, 6, 9, 13} are

examples of blind-write quorums.

Finally, a write quorum is formed by the union of a read quorum and a blind-write quorum. The set of copies {1, 5, 6, 7, 8, 10, 14} is an example of write quorum for the hierarchical grid of Figure 6.

The data availability in the hierarchical grid protocol for the read and blind-write operations are calculated level by level, in a way similar to the original grid protocol, and are expressed by the following recurrences:

$$A_r(i) = \begin{cases} \rho & (i = 0) \\ [1 - [1 - A_r(i-1)]^{m_i}]^{n_i} & (i \geq 1) \end{cases}$$

$$A_b(i) = \begin{cases} \rho & (i = 0) \\ 1 - [1 - [A_b(i-1)]^{m_i}]^{n_i} & (i \geq 1) \end{cases}$$

Before calculating the availability for the write operation two probabilities, regarding a grid of dimensions $m_i \times n_i$ at level i, are expressed: the probability of obtaining a read access permission, given by

$$P_r^{m_i,1}(i) = [A_r(i-1)]^{m_i},$$

and the probability of obtaining a write access permission, given by

$$P_w^{m_i,1}(i) = [A_b(i-1)]^{m_I} - [A_b(i-1) - A_w(i-1)]^{m_i},$$

which is the difference between the probability of forming a blind-write quorum in all objects of a column, and the probability of forming only blind-write quorums (and not write quorums).

The availability for the write operation is finally calculated as the difference between the probability of obtaining read access permission from all columns of the grid and the probability of obtaining only read access permission (and not write access permission) from all columns of the grid, being expressed by the recurrence

$$A_w(i) = \begin{cases} \rho & (i = 0) \\ [P_r^{m_i,1}(i)]^{n_i} - [P_r^{m_i,1}(i) - P_w^{m_i,1}(i)]^{n_i} & (i \geq 1) \end{cases}$$

In [Kumar and Cheung 1991] it is shown that the data availability for the hierarchical grid protocol using a hierarchical grid with l levels is greater than the data availability in the original grid protocol. The data availability for the hierarchical grid protocol approaches asymptoticly 1 as l increases, while in the original grid protocol the write availability approaches asymptoticly 0 as N increases.

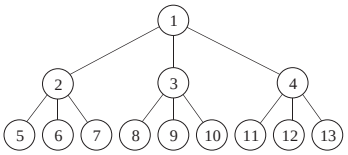

Figure 7: Copies organized logically in a tree with $h = d = 3$

3.4 Tree quorum [Agrawal and El Abbadi 1992b]

The tree quorum protocol also uses a tree for arranging logically the copies, but as opposed to hierarchical voting, the copies are placed not only as leaves, but in all vertices of the tree. The structure adopted is a complete tree with height h and degree d (number of children of each vertex), so the total number of vertices is $N = (d^h - 1)/(d - 1)$. Figure 7 shows an example of a tree in which $h = 3$ and $d = 3$, for a total of 13 copies.

Quorums are formed recursively, including the root copy and w of its children, and for each child included w of its own children, and so on, up to a depth l. The quorum formed is denoted by the pair $<l, w>$. When forming a quorum, if a copy at depth h' is not accessible it is substituted by w quorums of depth $l - h'$ and width w formed from children of the unaccessible copy. The recursion finishes successfully when the depth of the quorum to be formed is zero. It is not possible to form a quorum when the height of the remaining tree is less than the required depth; in that case, access permission is denied.

Read and write quorums are denoted respectively by the pairs $<l_r, w_r>$ and $<l_w, w_w>$. Similar to the simple voting protocols, the width and depth of the quorums must obey certain restrictions to guarantee that quorums for conflicting operations intercept each other:

$$l_r + l_w > h \quad \text{(a)}$$

$$w_r + w_w > d \quad \text{(b)}$$

$$2l_w > h \quad \text{(c)}$$

$$2w_w > d \quad \text{(d)}$$

Consider for example the tree shown in Figure 7, and suppose that read quorums are chosen as $< 1, 2 >$ and write quorums are chosen as $< 3, 2 >$. In this example, in the best case a read quorum is formed by the root copy alone. If the root is not accessible, a quorum formed by any two of the root's children may be used instead, for example {2, 3} or {3, 4}. If two or more of the root's children is also not available, these children can be substituted by two of their children. In that way, if copies 1, 2 and 3 are not accessible, a read quorum can be formed using copy 4 and two of the children of either copy 2 or 3, for example {4, 5, 6} or {4, 8, 10}. Finally, if copies 1, 2, 3 and 4 are not accessible, a read quorum can still be formed by two children of at least two copies among 2, 3 and 4, resulting in for example quorums {5, 6, 8, 9}, {6, 7, 12, 13} and {8, 10, 11, 13}. A write quorum of dimension $< 3, 2 >$ must include the root and two of its children together with two of their own children. Examples of write quorums are {1, 2, 3, 5, 6, 8, 9}, {1, 2, 4, 6, 7, 12, 13} and {1, 3, 4, 9, 10, 11, 13}. Similarly, choosing $q_r = q_w = <2, 2>$ for the tree in Figure 7, any

of the sets {1, 2, 3}, {1, 3, 4}, {1, 4, 5, 6} and {2, 4, 6, 7, 12, 13} would represent a read or a write quorum.

The size of a read quorum varies from $[(w_r)^{l_r} - 1]/(w_r - 1)$ in the best case, when all copies in the quorum are from the higher levels of the tree, to $(w_r)^{l_r-1}.[(w_r)^{l_r} - 1]/(w_r - 1)$ in the worst case, when the copies are from the lower levels. The availability is calculated in a similar way to the hierarchical voting protocol, using the availability for the simple voting protocol. The following recurrence expresses the availability for a quorum of dimension $< l, w >$ in a tree with height h and width w (the first term represents the case where the root is accessible and the second term represents the case where the root could not be contacted):

$$A_h(l,w) = \begin{cases} \rho & (h = 1) \\ \rho[AV(d,w,A_{h-1}(l-1,w))] + (1-\rho)[AV(d,w,A_{h-1}(l,w))] & (h > 1), \end{cases}$$

Two interesting cases of the tree quorum protocol are worth examining in more detail. In the first case, designated QA1, read quorums have dimension $< 1, (d+1)/2 >$ and write quorums have dimension $< h, d/2 + 1 >$. In the second case, designated QA2, the read and write quorum dimensions are respectively $< 1, d >$ and $< h, 1 >$. In the QA1 protocol a read quorum is formed by a single copy when no failures occur. If the root is not accessible, though, the quorum size increases to $(d + 1)/2$. If more copies are not accessible, the read quorum size may increase up to $[(d+1)/2]^h$ (note that in general $[(d+1)/2]^h < N/2$). Considering that 50% of the accesses are made using the root copy, the average read quorum size is $(h+1)/2$; since $h = \log_d[N(d-1)+1]$, this means that read quorum sizes are $O(\log N)$. Write quorums, on the other hand, are of fixed size $[[(d+1)/2]^h - 1]/[(d-1)/2]$.

For the QA2 protocol, a read quorum is also composed of a single copy when there are no failures, but its size may increase up to d^{h-1} copies when only leaves are accessible. Write quorums, on the other hand, are formed by any path from the root to a leaf copy, and have size h, which means they are $O(\log N)$.

3.5 A unified view: the general protocol

The grid, hierarchical grid and hierarchical voting protocols are said to be *symmetric* because all copies have the same probability of being accessed by the protocol. The tree quorum protocol is *asymmetric* in the sense that nodes higher in the hierarchy (nearer to the root) tend to be more accessed. In [Mendonça and Anido 1994] it was described a protocol which was shown to be a generalization of all symmetric protocols based in logical structures. This *general protocol* uses the same structure as the hierarchical voting protocol presented in Section 3.2, that is, a hierarchy in which a vertex at level i ($1 \leq i \leq m$) represents a logical group of l_i vertices from level $i - 1$; vertices at the lowest level (level 0) represent copies of the replicated data. The basic idea of the general protocol is the realization that symmetric structures such as a grid are only another way of viewing a kind of hierarchical voting. For example, the read quorum defined for the grid protocol ("one copy of each column") can be seen as a voting process composed of two-levels: in the first level, the voting takes place within each column, and the quorum for this level is one copy; in the

second level, the voting takes place among the columns, and the quorum for that level is "all columns".

The quorums in the general protocol are defined using the concept of *virtual* operations. Virtual operations, similarly to the blind-write operation described in the hierarchical grid protocol (Section 3.3), are not actually available to users. They are just a mechanism to provide more flexible read and write quorums. Two virtual operations, a and b, are defined for the protocol. Virtual operation a conflicts with virtual operation b and, commutatively, virtual operation b conflicts with operation a. Once the virtual operations have been defined, it is easy to derive quorums which guarantee one-copy-serializability for read and write operations. A read quorum is formed directly by either a quorum for virtual operation a or a quorum for virtual operation b. A write quorum is formed by the union of a quorum for virtual operation a and a quorum for virtual operation b. That way, it is guaranteed that any write quorum will always have a non-null intersection with any other write or read quorum.

Quorums for virtual operations a and b are formed recursively, starting from the root vertex at level m. The root vertex will allow virtual operation a if it can obtain permission from at least qa_m of its children at level $m-1$; each of these children will in turn grant permission if it can obtain permission from at least qa_{m-1} of its own children at level $m-2$, and so on, until permission is asked directly to the copies at level 0. Access permission then propagate bottom-up until it reaches the root vertex. To guarantee that quorums for virtual operations a and b intercept each other, $qa_i + qb_i > l_i$ for all levels i. To simplify the notation, the values l_i, qa_i and qb_i $(1 \leq i \leq m)$ will be denoted respectively by the arrays $L = (l_1, l_2, \ldots, l_m)$, $Q_a = (qa_1, qa_2, \ldots, qa_m)$ and $Q_b = (qb_1, qb_2, \ldots, qb_m)$.

A grid with x lines and y columns can be emulated in the general protocol using a 2-level hierarchy where $L = (x, y)$, $Q_a = (1, y)$ and $Q_b = (x, 1)$. For example, the hierarchy shown in Figure 8, in which $L = (3, 4)$, can used by the general protocol to emulate the grid of Figure 4 by making $Q_a = (1, 4)$ and $Q_b = (3, 1)$. Examples of quorums for virtual operation a are $\{1, 2, 3, 4\}$ and $\{1, 6, 3, 12\}$; examples of quorums for virtual operation b are $\{1, 5, 9\}$ and $\{3, 7, 11\}$. Defining a read quorum to be formed by virtual operation a, the general protocol generates the same read and write quorums as the the grid protocol. Note that the use of virtual operations allows the general protocol to define more read quorums than the grid protocol, since a read operation can be performed also by a quorum of Q_b, which, in the case of the asymmetric grid of the example, is smaller than the original read quorum.

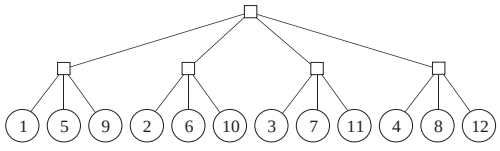

Figure 8: The grid of Figure 4 viewed as a two-level hierarchy

Since the general protocol is in fact an extension to the hierarchical voting protocol, quorums generated by the latter can also be generated by the former. The general protocol, however, allows more flexibility in forming the quorums, because it does not need the

constraint $2w_i > l_i$ present in the original hierarchical voting protocol.

Similar to the grid protocol, the hierarchical grid protocol can be simulated by the general protocol through the use of a hierarchy of $2k$ levels defined by $L = (x_1, y_1, x_2, y_2, \ldots, x_k, y_k)$, with $Q_a = (1, y_1, 1, y_2, 1, \ldots, 1, y_k)$, and $Q_b = (x_1, 1, x_2, 1, \ldots, x_k, 1)$. Figure 9 shows the hierarchical grid of Figure 6 organized as a 4-level hierarchy in which $L = (2, 2, 2, 2)$; by making $Q_a = (1, 2, 1, 2)$, $Q_b = (2, 1, 2, 1)$, the general protocol defines the same quorums obtained by the hierarchical grid protocol.

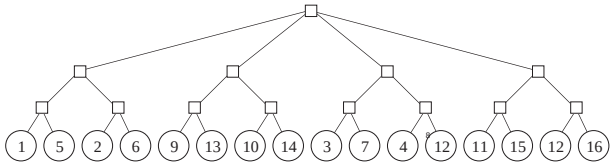

Figure 9: The hierarchical grid of Figure 6 viewed as a four-level hierarchy

The following recurrence expresses the availability for virtual operation a, A_a, in the general protocol:

$$A_a(i) = \begin{cases} \rho & (i = 0) \\ AV(l_i, qa_i, A_a(i-1)) & (i \geq 1) \end{cases}$$

Availability for virtual operation b, A_b, is calculated in the same way. The availability for a read operation, A_r, is therefore given by

$$A_r = min(A_a, A_b).$$

Availability for a write operation is calculated one level at a time. The write availability for level i in the hierarchy is given by the difference between the probability of forming the largest quorum of that level, $max(qa_i, qb_i)$, and the probability of forming *only* that quorum, i.e., the probability of not getting write access permissions from the minimum number, $max(qa_i, qb_i)$, of vertices required at level $i-1$ (among the children of a vertex at level i). So, write availability, A_w, in the general protocol is expressed by:

$$A_w(i) = \begin{cases} \rho & (i = 0) \\ AV(l_i, qa_i, A_a(i-1)) & \\ \quad -AV(l_i, qa_i, A_a(i-1) - A_w(i-1)) & \\ & (i \geq 1 \; and \; qa_i \geq qb_i) \\ AV(l_i, qb_i, A_b(i-1)) & \\ \quad -AV(l_i, qb_i, A_b(i-1) - A_w(i-1)) & \\ & (i \geq 1 \; and \; qa_i < qb_i) \end{cases}$$

The general protocol has been extended in [Mendonça and Anido 1995] to generalize also asymmetric structures such as the tree used in the tree quorum protocol.

4 Other approaches

Several other solutions, which do not use voting or logical structures, have been proposed to the problem of maintaining the consistency of replicated systems. The purpose of this section is to present a general overview of some of these other approaches.

4.1 Generic solutions

The first solution presented in this section aims at generalizing the pessimistic protocols, increasing the flexibility of forming quorums by eliminating the voting process. Instead of voting, the solution consists in pre-calculating all possible combinations of quorums. The second solution presented in this section describes a way of implementing that same flexibility by using a voting process.

4.1.1 Coteries [Garcia-Molina and Barbara 1985]

This solution proposes that the groups of copies which compose the quorums are pre-defined by the system administrator, thus eliminating the necessity of a voting process for every data access. The set of groups of copies which can be used as quorums is called a *coterie*.

A coterie is formally defined as follows. Let C be the set of copies of the replicated data. A set S of groups of copies is a coterie over C if and only if

(i) $G \in S$ implies that $G \neq \emptyset$, and $G \subseteq C$,

(ii) if $G, H \in S$ and either G is a read quorum candidate and H is a write quorum candidate, or both G and H are write quorum candidates, then G and H must have at least one copy in common,

(iii) there are no two groups $G, H \in S$ such that $G \subset H$.

As an example, consider a replicated data with six copies represented by the set C={1, 2, 3, 4, 5, 6}. One possible coterie over C would be the sets R={{1, 3, 5}, {2, 4, 6}} and W={{1, 2, 3, 4}, {3, 4, 5, 6}, {2, 3, 5, 6}}, where R and W denote respectively the groups of copies candidates to read and write quorums.

Once a coterie for the set of replicated copies is defined, at each access to the data the replica control protocol simply requests the required access from the copies in a candidate quorum group. If all copies in that group grant permission, the access can be performed. If one or more copies of the chosen group cannot be contacted, the protocol attempts to obtain the permission from another group, idealy using the access permissions already obtained. The process is repeated until either a quorum is successfully formed or all candidate groups in the coterie have been tried; in the last case access to the data is denied.

It is easy to verify that all quorums generated by a voting mechanism constitute a coterie. The inverse however is not true, i.e., there are coteries which cannot be generated by any vote assignment in the simple voting protocol, as shown in [Garcia-Molina and Barbara 1985].

Although coteries provide more flexibility in forming quorums, the number of possible groups of candidate quorums is exponential in the total number of copies. When the number of copies is small (up to five) it is feasible to enumerate all possible coteries and to choose the one which provides the greater availability. For a system with more than five copies, the enumeration of all coteries is not practical. In [Tang and Natarajan 1989] it is proposed a scheme in which the search for the best coterie is solved using linear programming techniques, so that the coteries approach can be used in a system with up to 10 copies.

4.1.2 Multi-dimensional voting [Ahamad et al. 1991]

In this scheme both the votes assigned to the copies and the quorums are arrays of dimension k. For a replicated data with N copies the vote assignment $V_{N,k}$ is an array $N \times k$ where $v_{i,j}$ represents the number of votes of copy i at dimension j ($v_{i,j} \geq 0$; $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, k$).

There are two levels of requirements in multi-dimensional voting: votes and dimensions. At the vote level the number of votes obtained at one dimension must be greater or equal to the quorum required at that dimension. At the dimension level, the number of dimensions that must obtain the required quorum must be greater or equal to a parameter l. The requirement that quorums be obtained in l from a total of k dimensions is denoted by the pair (l,k). Note that (1,1) is therefore equivalent to simple (one-dimension) voting.

Answers		*Vote Assignment*					*Votes Received*				
Copy 1	$\longrightarrow$	2	0	2	2	$\longrightarrow$	(2	0	2	2)	
		3	1	0	0						
Copy 3	$\longrightarrow$	0	1	0	2	$\longrightarrow$	(0	1	0	2)	
Copy 4	$\longrightarrow$	1	0	1	1	$\longrightarrow$	(1	0	1	1)	
		1	0	1	1						
		0	1	2	2						
						+					
							(3	1	3	5)	*Vote Result*
						−	(5	3	5	5)	*Quorum Array*
							f	*f*	*f*	*s*	*Quorum obtained in one dimension only*

Figure 10: Multi-dimensional voting for six copies using four dimensions

Figure 10 shows an example of multi-dimensional voting for a replicated data with six copies. In the example, only the votes from copies 1, 3 and 4 have been received for a given data access requisition. These three arrays are added and the resulting array is compared to the quorum array (5, 3, 5, 5). The comparison is performed for each dimension, and the results s and f represent respectively success and failure. In the example only one dimension obtained the required quorum. A data access operation with requirement (1,4) would therefore be allowed; access operations with requirement (i,4) would not be allowed

for i=2, 3, and 4.

In [Ahamad et al. 1991] it is shown that any coterie can be generated by multi-dimensional voting. It is also shown that multi-dimensional voting can be used to implement a generalization of the voting with fragments protocol, in which the number of fragments are not necessarily the same in all segments. Although extremely flexible, multi-dimension voting cannot be considered strictly as a protocol, since it does not define any rules as of what should be done, but only about how it can be done. In that way, it relies on some other voting scheme to define what are the quorums, and only implements, by voting, the quorums defined.

4.2 Optimistic approaches

As discussed in section 1.3, optimistic replica control protocols are those that allow, while the system is partitioned, access to the replicated data in more than one partition. This section describes two optimistic protocols, which utilize the same strategy for detecting inconsistencies (finding a cycle in a dependency relation), but use different approaches for the implementation of the strategy.

In both protocols, a coordinator for each partition is elected as soon as a partition failure is detected. The coordinator is responsible for maintaining information about accesses to the replicated copy while the system is partitioned, and for managing the process of merging the partitions when the failure is repaired.

4.2.1 Transaction-based approach [Davidson 1984]

The coordinator of a partition maintains a graph in which vertices correspond to transactions executed in the partition, and edges correspond to dependency relations between the transactions. Two dependency relations are defined: "read-from", when a transaction read data produced from another transaction, and "read-before", when a transaction read data which was afterwards altered by another transaction.

When the communication between two partitions is re-established, one of the two coordinators constructs a global graph of precedence with the information from both partitions, and *interference edges*, which correspond to dependency relations between transactions in the two partitions, are added to the global graph. An interference edge represents the fact that if a transaction read some data in one partition that transaction must precede any other transaction which has updated the data in the other partition. Copy inconsistency is then detected by the presence of cycles in the resulting graph.

To eliminate the inconsistency the cycle is broken by cancelling one or more transactions. The commit of transactions must therefore be postponed during system partitioning until the partitions merge. Obviously, the best strategy to break the cycles is to minimize the transactions that must be cancelled. Unfortunately, the problem of breaking cycles in a graph while minimizing the number of vertices removed is shown to be NP-complete in [Davidson 1984]. On the other hand, reasonably efficient strategies, such as breaking first smaller cycles, can always be applied.

4.2.2 Data-based approach [Ramarao 1989]

To each data d in a partition P it is associated a set, denoted *precedence-set(d)*, containing the identifiers of all data stored in P which *precede* d. For two data d_i and d_j stored in P, d_i is said to precede d_j if d_j is read in a transaction T executed in P so that either T updates d_i or T appears strictly after T' in some sequential order for transactions executed in P, where T' is a transaction which updates d_i. That way, when two partitions P and P' are regrouped, copies of the replicated data are not consistent if there are two data d_i and d_j such that d_i precedes d_j in P and d_j precedes d_i in P'. As in the case of the transaction-based protocol, the inconsistencies are eliminated by canceling some of the involved transactions.

4.3 Solutions not based on one-copy-serializability

All replica control protocols presented in the previous sections use one-copy-serializability as the basic correctness criterion. This section presents a brief overview of some protocols that adopt other correctness criteria, which maintain the integrity of the logical data even though one-copy-serializability is not preserved. The proposed correctness criteria explore semantic aspects of the applications in order to reduce the constraints normally imposed to the access of replicated data.

4.3.1 Commutativity [Kumar and Stonebraker 1988]

The semantic property exploited in this approach is *commutativity*, which captures the fact that some concurrent operations may be implemented in any order without altering the result [Weihl 1988]. It is suggested that transactions are pre-analysed so as to classify them in one of four classes: *C*-transactions, which commute with any other transaction; *PC*-transactions, which under certain integrity constraints are executed as *C*-transactions; *NC1*-transactions, which commute with each other and with *C*-transactions; and *NC2*-transactions, which commute only with *C*-transactions.

The protocol requires different quorums depending on the transaction class. *C*-transactions do not need to form any quorum to access the data, and therefore execute faster. *PC*-transactions, once they have their integrity constraint satisfied, also do not need to form quorums to access the data. Transactions in classes *NC1* and *NC2* must form quorums before each data access, since they do not commute. In that way, considering a replicated data with N copies, quorums nc_1 and nc_2 for transactions in classes *NC1* and *NC2* respectively, must be defined such that $nc_1 + nc_2 > N$.

4.3.2 ϵ-serializability [Pu and Leff 1991]

In this approach the new correctness criterion proposed, ϵ-serializability, allows a bounded (and temporary) degree of inconsistency in the system. To enforce it transactions are divided in two classes: update transactions (ETUs), which include at least one update operation, and query transactions (ETQs), which include only read operations. ETUs are executed following the one-copy-serializability criterion, while ETQs are not restricted to any constraints. The basic idea is to perform the updates caused by ETUs asynchronously,

which improves their execution time but may cause ETQs to access data that is not up-to-date. The inconsistency however will be temporary, as at the end of the ETUs, which are one-copy-serializable, the data will be consistent. The protocol provides a way to control the degree of inconsistency in ETQs so as to maintain it within a specified value ϵ; if $\epsilon = 0$, i.e., the degree of inconsistency is "none", all transactions, ETUs and ETQs, are one-copy-serializable. To implement this bounded inconsistency transactions are allowed to import and export a limited amount of inconsistencies.

4.3.3 M-ignorance [Krishnakumar and Bernstein, 1995]

This approach exploits the fact that in some systems a transaction needs not see the results of at most M previous transactions it would have seen if the execution was serialized. A typical example is an airline reservation system. Suppose an airplane has 300 seats, but an overbooking of 10 passengers is acceptable. Considering transactions which reserve a single seat, a 10-ignorant system can provide access to 10 concurrent transactions, instead of access to a single transaction in one-copy-serializability, and still preserve the integrity constraint $\mathsf{res\_seat} \leq 300$.

The basic difference between this approach and ϵ-serializability is that violations of the integrity constraints are defined as a function of the number of conflicting transactions that can execute concurrently.

4.3.4 Causality [Joseph and Birman 1986, Ladin et al. 1992]

The notion of *causal order* is widely used in in distributed systems [Lamport 1978]. It expresses the fact that some systems do not need to use a "physical time" order to be considered correct; a suitable "happened-before" order suffices. For example, if user u_1 and u_2, without any prior communication, send each an independent message to user u_3, it is in general irrelevant the order in which u_3 receives the messages. However, if u_1 sends two consecutive messages to u_3, it is reasonable to expect that u_3 receives them in the order they were sent. Furthermore, if user u_1 sends two consecutive messages, the first to u_2 and the second to u_3, and the receipt of this message causes u_3 also to send a message to u_2, it is reasonable to expect that u_2 receives the message from u_1 before it receives the message from u_3. In both cases the meaning of the second message may depend on the contents of the first one.

Two approaches have been proposed to implement causality as the correctness criteria in replica control protocols. Both approaches use a variation of the ROWA scheme, in which the execution time of a write operation is reduced by not blocking the user while the new value is disseminated to the other copies. Instead, the dissemination is executed asynchronously. Write operations therefore have faster response time, but read operations may access data in a way which is not one-copy-serializable. The approaches differ in the strategy to guarantee causality.

In [Joseph and Birman 1986], the replica control protocol relies on the support offered by an existing communication system, ISIS. The Isis communication model is based on the notion of processes groups; members of a group communicate using a reliable multicast

service. This service provides different multicast primitives, which permit different ordering constraints, from causal to total order (in total order, all members receive all messages in exactly the same order). In this approach, a timestamp is assigned to each read or write request by a user. A group is formed by all users and copies of a logical data; operations are multicast to all group members. By delivering the requests in the specified order (causal or total), the communication system guarantees that although operations are performed in only one replica, a copy will not execute a request that may break the order chosen (causal or total).

In [Ladin et al. 1992], another approach, denominated *lazy replication*, is described. It implements causality by using a specially designed protocol to achieve better performance. The protocol includes a concurrency control mechanism, based on quorum locks, and a dissemination mechanism, based on gossip messages (gossip messages are point-to point background messages that can be used to broadcast information).

5 Final remarks

When choosing a replica control protocol for an application several other factors, besides data availability and quorum sizes, must be taken into account. A system may have a sufficient number of operational nodes but the replica control protocol might spend too much time to obtain the required quorum. Also, a certain protocol P_1 may send more messages than another protocol P_2, but P_1 may be able to perform the exchanges of messages concurrently, while P_2 may have to perform the exchanges sequentially. Therefore characteristics such as load balancing and the size and number of messages exchanged are also important factors, as is the *data delay*, defined as the delay between the time the data was requested by the user and the time the access was performed.

Two metrics to compare replica control protocols which take into account the data delay have been proposed. In [Menascé et al. 1994], it is considered that the cause of the data delay is the message overhead the protocol imposes on the network system; node processing is assumed to be negligible. The metric proposed is defined as A/d, where A is the data availability and d is the average normalized data delay, i.e., the average data delay divided by the node to node communication time under zero network load. In [Rangarajan et al. 1993] a similar metric, called *capacity*, is proposed. Contrary to the previous approach, this metric considers that the cost of a message is not the channel bandwidth it requires, but the workload it imposes on the nodes in processing the received information. Denoting by w the average processing time *per node* to perform a successful operation, the (average) capacity of the system is defined as A/w. Both metrics are useful to compare different protocols, since they reflect the number of operations that can be successfully performed by the system in a unity of time. The higher the data availability and the lower the data delay, the better is the protocol.

Table 1 shows yet another characteristic of replica control protocols: the number of copies that may be inaccessible, without causing service disruption, for several of the protocols presented in this paper. It must be noted that, for protocols based on logical structures, the values in the table do not mean any set with that number of copies; rather, the values

Protocol	Read Access		Write Access	
	Best Case	Worst Case	Best Case	Worst Case
ROWA	$N-1$	$N-1$	0	0
Majority Voting	$\lfloor N/2 \rfloor$	$\lfloor N/2 \rfloor$	$\lfloor N/2 \rfloor$	$\lfloor N/2 \rfloor$
Grid	$N-\sqrt{N}$	$\sqrt{N}-1$	$N-2\sqrt{N}+1$	$\sqrt{N}-1$
Hierarchical Voting	$N-N^{0.63}$	$N^{0.63}-1$	$N-N^{0.63}$	$N^{0.63}-1$
Hierarchical Grid	$N-\sqrt{N}$	$\sqrt{N}-1$	$N-2\sqrt{N}+1$	$\sqrt{N}-1$
Tree Quorum	$N-1$	$\log N-1$	$N-\log N$	0

Table 1: Fault-tolerance, in number of copies, for some of the protocols presented

depend as well on the position of the failed copies within the structure.

References

Agrawal, D. and El Abbadi, A. 1990a. Storage efficient replicated databases. *IEEE Transactions on Knowledge and Data Engineering 2,* 3 (September), 342–351.

Agrawal, D. and El Abbadi, A. 1990b. The tree quorum protocol: An efficient approach for managing replicated data. In *Proceedings of the* 16^{th} *VLDB Conference*, pp. 243–254.

Agrawal, D. and El Abbadi, A. 1992. The generalized tree quorum protocol: An efficient approach for managing replicated data. *ACM Transactions on Database Systems 17,* 4 (December), 689–717.

Agrawal, G. and Jalote, P. 1995. Coding-based replication schemes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems 6,* 3 (March), 240–251.

Ahamad, M. and Ammar, M. H. 1989. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Transactions on Software Engineering 15,* 4 (April), 492–495.

Ahamad, M., Ammar, M. H., and Cheung, S. Y. 1991. Multidimensional voting. *ACM Transactions on Computer Systems 9,* 4 (November), 399–431.

Barbara, D. and Garcia-Molina, H. 1987. The reliability of voting mechanisms. *IEEE Transactions on Computers C-36,* 10 (October), 1197–1208.

Barbara, D., Garcia-Molina, H., and Spauster, A. 1989. Increasing availability under mutual exclusion constraints with dynamic vote reassignment. *ACM Transactions on Computer Systems 7,* 4 (November), 394–426.

Barghouti, N. S. and Kaiser, G. E. 1991. Concurrency control in advanced database applications. *ACM Computing Surveys 23,* 3 (September), 269–318.

Bernstein, P. A. and Goodman, N. 1983. The failure and recovery problem for replicated databases. In *Proceedings of the* 2^{nd} *Symposium on Principles of Distributed Computing*, pp. 114–122. ACM.

Bernstein, P. A. and Goodman, N. 1984. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems 9,* 4 (Dec.), 596–615.

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

Cheung, S. Y., Ahamad, M., and Ammar, M. H. 1992. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Transactions on Kowledge and Data Engineering 4,* 6 (December), 582–592.

Davidson, S. B. 1984. Optimism and consistency in partitioned distributed database systems. *ACM Transactions on Database Systems 9,* 3 (September), 456–481.

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

El Abbadi, A., Skeen, D., and Cristian, F. 1985. An efficient, fault-tolerant protocol for replicated data management. In *Proceedings of the* 4^{th} *Symposium on Principles of Database Systems*, pp. 215–228. ACM.

El Abbadi, A. and Toueg, S. 1989. Maintaining availability in partioned replicated databases. *ACM Transactions on Database Systems 14,* 2 (June), 264–290.

Garcia-Molina, H. 1982. Elections in a distributed computing system. *IEEE Transactions on Computers C-31,* 1 (January), 48–59.

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

Gifford, D. K. 1979. Weighted voting for replicated data. In *Proceedings of the* 7^{th} *Symposium on Operating Systems Principles*, pp. 150–162. ACM.

Jajodia, S. and Mutchler, D. 1990. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Transactions on Database Systems 15,* 2 (June), 230–280.

Joseph, T. A. and Birman, K. P. 1986. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Transactions on Computer Systems 4,* 1 (February), 54–70.

Krishnakumar, N. and Bernstein, A. J. 1994. Bounded ignorance: a technique for increasing concurrency in a replicated system. *ACM Transactions on Database Systems 19,* 4 (December), 596–625.

Kumar, A. 1991. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Transactions on Computers 40,* 9 (September), 996–1004.

Kumar, A. and Cheung, S. Y. 1991. A high availability $\sqrt{N}$ hierarchical grid algorithm for replicated data. *Information Processing Letters 40,* 6 (December), 311–316.

Kumar, A., Rabinovich, M., and Sinha, R. K. 1993. A performance study of a new grid protocol and general grid structures for replicated data. Tech. Rep. 93-03-02 (March), Department of Computer Science and Engineering, University of Washington.

Kumar, A. and Segev, A. 1988. Optimizing voting-type algorithms for replicated data. *Lectures Notes in Computer Science 303*, 428–442.

Kumar, A. and Stonebraker, M. 1988. Semantics based transaction management techniques for replicated data. In *SIGMOD*, pp. 117–125. ACM.

Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. 1992. Providing high availability using lazy replication. *ACM Transactions on Computer Systems 10,* 4 (November), 360–391.

Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21,* 7, 558–565.

Lampson, B. W., Paul, M., and Siegert, H. J. (Eds.) 1983. *Distributed Systems: Architecture and Implementation.* Spring-Verlag Berlin Heidelberg New York.

Menascé, D. A., Yesh, Y., and Kalpakis, K. 1994. On a unified framework for the evaluation of distributed quorum attainement protocols. *IEEE Transactions on Software Engineering 20,* 1 (November), 868–890.

Mendonça, N. C. and Anido, R. 1994. Using extended hierarchical quorum consensus to control replicated data: from traditional voting to logical structures. In *Proceedings of the* 27^{th} *Annual Hawaii International Conference on Systems Sciences, Minitrack on Parallel and Distributed Databases*, Maui, Hawaii, pp. 303–312.

Mendonça, N. C. and Anido, R. 1995. A general protocol for maintaining replicated data. Tech. Rep. INT-23/95, Institut National des Telecommunications, Evry - France.

Minoura, T. and Wiederhold, G. 1982. Resilient extended true-copy token scheme for a distributed database system. *IEEE Transactions on Software Engineering SE-8,* 3 (May), 173–188.

Pâris, J.-F. 1986. Voting with witnesses: A consistency scheme for replicated files. In *Proceedings of the* 6^{th} *Conference on Distributed Computing Systems*, pp. 606–612. IEEE.

Pâris, J.-F. 1990. Efficient voting protocols with witnesses. *Lectures Notes in Computer Science 470*, 305–317.

Pu, C. and Leff, A. 1991. Replica control in distributed systems: An asynchronous approach. In *SIGMOD*, pp. 377–386. ACM.

RAMARAO, K. V. S. 1989. Detection of mutual iconsistency in distributed databases. *Journal of Parallel and Distributed Computing 6*, 498–514.

RANGARAJAN, S., JALOTE, P., AND TRIPATHI, S. K. 1993. Capacity of voting systems. *IEEE Transactions on Software Engineering 19,* 7 (July), 698–706.

STONEBREAKER, M. 1979. Concurrency control and consistency of multiple copies of data in distributed INGRES. *IEEE Transactions on Software Engineering SE-5,* 3 (May), 188–194.

TANG, J. AND NATARAJAN, N. 1989. A static pessimistic scheme for handling replicated databases. In *SIGMOD*, pp. 389–398. ACM.

THOMAS, R. H. 1979. Majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems 4,* 2 (June), 180–209.

TRAIGER, I. L., GRAY, J., GALTIERI, C. A., AND LINDSAY, B. G. 1982. Transactions and consistency in distributed database systems. *ACM Transactions on Database Systems 7,* 3 (September), 323–342.

WEIHL, W. E. 1988. Commutativity-based concurrency control for abstract data types. *IEEE Transactions on Computers 37,* 12 (December), 1488–1505.

WUU, G. T. J. AND BERNSTEIN, A. J. 1984. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the* 3^{rd} *Symposium on Principles of Distributed Computing*, pp. 57–66. ACM.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*