

- [2] B. Bhargava and Z. Ruan. Site recovery in replicated distributed database systems. In *6th IEEE Conference on Distributed Computing Systems*, pages 621–627, May 1986.
- [3] R. G. Casey. Allocation of copies of a file in an information network. In *AFIPS Conference Proceedings*, pages 617–625, 1972.
- [4] D. Davcev and W. A. Burkhard. Consistency and recovery control for replicated files. In *Proc. 10th ACM Symposium on Operating System Principles*, pages 87–96, 1985.
- [5] B. Gavish and H. Pirkul. Computer and database location in distributed computer systems. *IEEE Transactions on Computers*, C-35(7):583–590, July 1986.
- [6] D. K. Gifford. Weighted voting for replicated data. In *7th ACM SIGOPS Symposium on Operating Systems Principles*, pages 150–159, December 1979.
- [7] C. L. Huang and V. O. K. Li. Regeneration-based multiversion dynamic voting scheme for replicated database systems. In *6th IEEE Conference on Data Engineering*, pages 370–377, 1990.
- [8] S. Jajodia and D. Mutchler. Dynamic voting. In *ACM SIGMOD Conference*, pages 227–238, 1987.
- [9] S. Jajodia and D. Mutchler. Integrating static and dynamic voting protocols to enhance file availability. In *4th IEEE International Conference on Data Engineering*, pages 144–153, 1988.
- [10] S. Jajodia and D. Mutchler. A pessimistic consistency control algorithm for replicated files which achieves high availability. *IEEE Transactions on Software Engineering*, 15(1):39–46, Jan 1989.
- [11] S. Jajodia and D. Mutchler. Dynamic voting algorithms for maintaining consistency of a replicated database. *ACM Transactions on Database Systems*, 15(2), June 1990.
- [12] D. E. Long and J. L. Carroll. Regeneration protocols for replicated objects. In *5th IEEE Conference on Data Engineering*, pages 538–545, 1989.
- [13] D. E. Long and J. L. Carroll. The reliability of regeneration-based replica control protocols. In *9th IEEE Conference on Distributed Computing Systems*, pages 465–473, 1989.
- [14] S. Mahmoud and J. S. Riordan. Optimal allocation of resources in distributed information networks. *ACM Transactions on Database Systems*, 1(1):66–78, March 1976.
- [15] H. L. Morgan and K. D. Levin. Optimal program and data locations in computer networks. *Communications of ACM*, 20(5):315–322, May 1977.
- [16] J. F. Paris. Voting with witnesses: A consistency scheme for replicated files. In *6th IEEE Conference on Distributed Computing Systems*, pages 606–612, 1986.
- [17] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its Eden implementation. *IEEE Transactions on Software Engineering*, 14(7):936–945, July 1988.
- [18] R. V. Renesse and A. S. Tanenbaum. Voting with ghosts. In *8th IEEE Conference on Distributed Computing Systems*, pages 456–462, 1988.
- [19] R. Tewari. *Robustness in Replicated Databases*. PhD thesis, Rutgers University, 1990.
- [20] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

owned by site A . The state of the system after site A recovers is:

A'_O	B_O	C_O	D'_O	E'_O
$v_O = 13$	$v_O = 13$	$v_O = 13$	$v_O = 13$	$v_O = 8$
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, B,$ $C, D)$	$(A, B,$ $C, D)$	$(A, B,$ $C, D)$	$(A, B,$ $C, D)$	$(A, B,$ $C, D, E)$

The data object copy at site A is converted to a virtual copy in accordance with the recovery module, since $GT(O) = 2$.

State VIII (After site E recovers)

After site E recovers, all sites can communicate with each other and all sites integrate with each other, forming a non-partitioned network. The state of the system at this point is:

A'_O	B_O	C_O	D'_O	E'_O
$v_O = 14$				
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, B,$ $C, D, E)$				

This example traces our algorithm for one complete cycle starting from a non-partitioned state and continuing through several site failures. The recovery of sites is further traced until all sites have recovered. Notice that due to the value of generation threshold chosen ($GT(O) = 2$), we end the cycle with three virtual copies at sites A, D and E , even though we started with two virtual copies at sites D and E . This illustrates the self adaptive nature of the algorithm.

6 Conclusion and Future Research

In this paper, we propose an algorithm that utilizes regeneration in distributed systems to improve the availability of replicated data. Specifically, our algorithm has the following advantages over a pure regeneration strategy or a pure dynamic voting strategy:

1. The RVC algorithm automatically maintains system availability using the generation threshold $GT(O)$, so that manual intervention is avoided for all but total failure. Thus, it is self-adaptive to changes in system configuration by regenerating data objects at other sites when there is a danger of losing the last real copy of the data object. On the other hand when sites recover, some real copies may be converted back to virtual copies in an effort to obtain the optimal initial allocation.

2. Unlike pure dynamic voting algorithm, the RVC algorithm makes a majority partition available for all possible cases of site failures. Our algorithm ensures that the last copy is lost only when the last site of the current majority partition fails. This leads to improved availability as compared to the pure dynamic voting algorithm.

3. The RVC algorithm ensures that data objects are regenerated whenever the number of real copies falls below the $GT(O)$ for that object. In the case of network partitioning this eliminates the possibility that a partition is current, but does not have a current copy of the data object, as long as at least one site in the partition is operational.

Future research in this area needs to address the issue of determining the vector $GT(O)$ using as inputs the communication costs, a statistical profile of the database and availability/reliability considerations. Another issue that needs further consideration is the determination of the number of virtual copies of each data object that should be created as well as their location. Finally, an important question is: at which site should the data object be regenerated when the associated threshold $GT(O)$ has been crossed. We are currently investigating these issues.

Acknowledgments

We would like to thank the anonymous referees for their careful reading of the paper and suggestions that helped to improve the quality of the paper. We acknowledge constructive comments and suggestions by Dr. Bharat Bhargava of Purdue University and Dr. David Rozenstein of Rutgers University. Financial support from the GSM Research Resources Committee, Rutgers University, is acknowledged by Nabil Adam and Rajiv Tewari. Summer research support from Temple University for 1990 is acknowledged by Rajiv Tewari.

References

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

the $GT(O)$ unspecified at this point, but one technique for determining it is to set $GT(O)$ to the number of copies of data object O to attain a minimum tolerable availability in the system.

State I (initial state)

The initial state of the system has version numbers equals to six and, as depicted by the update vectors, all five sites that have a copy of the data object are in communication with each other.

A_O	B_O	C_O	D'_O	E'_O
$v_O = 6$				
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, B,$				
$C, D, E)$				

State II (after two updates)

This state depicts the system after two updates have been successfully propagated. The update vectors show that each site can communicate with all other sites.

A_O	B_O	C_O	D'_O	E'_O
$v_O = 8$				
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, B,$				
$C, D, E)$				

State III (After site E fails and one update takes place)

The state of the system after site E 's failure is:

A_O	B_O	C_O	D'_O	E'_O
$v_O = 9$	$v_O = 9$	$v_O = 9$	$v_O = 9$	$v_O = 8$
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, B,$				
$C, D)$	$C, D)$	$C, D)$	$C, D)$	$C, D, E)$

The virtual copy at site E is not included in the write quorum, and the update vectors of all the operational sites reflect the set of sites that participated in the last update to data object O . The version number of E remains 8, indicating that it has become out-of-date. It should be noted that if site E had failed, and there was no update at that point in time, the state information of all the other sites would not have been updated instantaneously merely on failure of E . The state information would be updated at the next update request to the data object O .

State IV (After site B fails and one update takes place)

The state of the system after site B 's failure is:

A_O	B_O	C_O	D'_O	E'_O
$v_O = 10$	$v_O = 9$	$v_O = 10$	$v_O = 10$	$v_O = 8$
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, C,$	$(A, B,$	$(A, C,$	$(A, C,$	$(A, B,$
$D)$	$C, D)$	$D)$	$D)$	$C, D, E)$

Now consider some hypothetical partitioning, say A/CD' . CD' would be the majority partition, since it has 2 out of three current copies (real and virtual). For partitioning AC/D' , AC would be the majority partition. Hence for any possible partitioning of the functional sites, we can find a majority partition, thereby guaranteeing mutual exclusion.

State V (After Site A fails and one update takes place)

The state of the system after site A 's failure is:

A_O	B_O	C_O	D_O	E'_O
$v_O = 10$	$v_O = 9$	$v_O = 11$	$v_O = 11$	$v_O = 8$
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, C,$	$(A, B,$	(C, D)	(C, D)	$(A, B,$
$D)$	$C, D)$	(C, D)	(C, D)	$C, D, E)$

Since the generation threshold ($=2$) has been crossed, D' is upgraded to a real copy by copying information from site C . Now, if a partition occurs, C and D are both partitions containing one real copy. The tie for majority partition will be broken in favor of the lexicographically largest copy, in this case, C . Thus, site C will be the majority partition. We have thus shown that read and write operations are always performed in a majority partition. A majority partition can always be obtained down to the last copy. At any state, for any possible partitioning, the topological information about the network carried in the U_i vector enables the collection of votes to form a majority partition.

We now continue the example to illustrate site recovery and integration.

State VI (After site B recovers)

Suppose, site B recovers (and runs the function *recover*) and finds that it can communicate with C, D . Then site B determines that it is in a majority partition, and it proceeds to integrate by copying the object's information from C or D , giving:

A_O	B_O	C_O	D'_O	E'_O
$v_O = 10$	$v_O = 12$	$v_O = 12$	$v_O = 12$	$v_O = 8$
$U_A^O =$	$U_B^O =$	$U_C^O =$	$U_D^O =$	$U_E^O =$
$(A, C,$	$(B, C,$	$(B, C,$	$(B, C,$	$(A, B,$
$D)$	$D)$	$D)$	$D)$	$C, D, E)$

The threshold ($GT = 2$) is reached, hence the lexicographically lowest copy (i.e. at D) is converted into a virtual copy.

State VII (After site A recovers)

Site A recovers (and an update arrives) and runs the function *recover*, described later, for each data object

Figure 2: The Function Majority

```

function majority( $O$ : object,  $i$ : site_id, var  $C$ : set_of_sites):
    boolean;
begin
     $C \leftarrow$  set of sites that communicate with this site
    for each  $i \in C$  begin
        read  $v_i, U_i$ 
    end
    let  $v_{max} = \max\{v_i \mid i \in C\}$ 
    for each  $i \in C$  begin
        if  $v_i = v_{max}$  then
            include  $i$  in set  $S$ 
        end
    end
    let  $U = U_i$  for some  $i \in S$ 
    if ( $\text{card}(S) > 1/2 \text{ card}(U)$ ) or
       ( $\text{card}(S) = 1/2 \text{ card}(U)$  and  $\max_{i \in U}(v_i) \in S$ ) then
        majority := true
    else
        majority := false
    end; {majority}

```

Figure 3: The DO_READ Module

```

procedure DO_READ( $O$ : object)
begin
    if majority( $O$ , site,  $C$ ) then
        select any real copy  $i \in S$ 
        read from  $i$ 
        commit( $C$ )
    else
        abort( $C$ )
    end; {DO_READ}

```

where, v_O indicates the version number of object O , and U_A^O indicates the update site vector of object O at site A . This vector contains a list of the sites that participated in the last update to object O .

A proof of the correctness of the RVC algorithm and more related details can be found in [19].

5 An Example

For the purpose of illustrating the detailed operation of the RVC algorithm, consider a five site network with the value of the generation threshold for the given data object O , $GT(O)$ assumed to be set to 2 for illustration purposes. We have left the procedure for determining

Figure 4: The DO_WRITE Module

```

procedure DO_WRITE( $O$ : object)
begin
    if majority( $O$ , site,  $C$ ) then
        for each data object  $i \in S$ 
            if  $i$  is a real copy
                perform the write
                update state information
            else if  $i$  is a virtual copy
                update state information;
        commit( $C$ )
    else
        abort( $C$ )
    end; {DO_WRITE}

```

Figure 5: The DO_FAILURE Module

```

procedure DO_FAILURE( $i$ : site_id);
begin
    for each data object  $O \in i$ 
        if ( $O$  is a real copy) and ( $GT(O) < k$ ) then
            determine the site  $j$ ,
            to regenerate data object  $O$ 
            upgrade a virtual copy of  $O$  to
            real copy at site  $j$ 
    end; {DO_FAILURE}

```

Figure 6: The DO_RECOVER Module

```

procedure DO_RECOVER( $i$ : site_id);
begin
    for each data object  $O \in i$ 
        if majority( $O, i, C$ ) then
            if  $O$  is a real copy
                if site  $i \in T(i, O)$  {site  $i$  is in optimal alloc.}
                    copy contents of  $O$  from any  $i \in S$ ;
                    update state information;
                    commit( $C$ );
                else {site  $i$  is not in optimal alloc.}
                    convert site  $i$ 's copy to virtual copy
                    update state information
                    commit( $C$ );
            else if  $O$  is a virtual copy
                update state information;
                commit( $C$ );
            else { $O$  is not in a majority partition}
                abort( $C$ );
    end; {DO_RECOVER}

```

3 Design Objectives of the RVC Algorithm

The design objectives for the algorithm are:

1. The algorithm should be applicable to geographically distributed architecture.
2. The algorithm should be able to handle Network partitioning as well as site failure.
3. A *read quorum* should require only one current real copy of a data object. The other virtual copies comprising the quorum could have state information, but no associated data.
4. A *write quorum* should require a minimum of one current real copy of a data item. The other copies could be virtual copies.
5. Virtual copies can be upgraded to real copies according to a prespecified protocol, whenever the number of real copies drops below a specified *generation threshold*.
6. The algorithm should be truly distributed. That is, each partition should be able to decide autonomously whether it is a majority partition based on the state information associated with the data objects in that partition.

4 Description of the RVC Algorithm

We first present the algorithm's major steps followed by a detailed description of the algorithm.

Step 1: When a read request for a data object O is received at a given site, the DO_READ module is executed by that site, which then runs the function *majority* to check if it is in the current majority partition with respect to the data object O . If it is, the read request is satisfied by a real copy in that partition. Otherwise, the request is turned down. A quorum made up of at least one real copy will be allowed.

Step 2: When a write request for a data object O is received at a given site, that site will execute the DO_WRITE module. If the site determines that it is in a current majority partition, the update will be propagated to each site that has a real copy and

the associated state information will be updated. For those sites that have virtual copies, only the state information will be updated.

Step 3: When a site detects the failure of another site, it runs the DO_FAILURE module. For each data object O belonging to the failed site (this information is maintained in the allocation table $T(O, j)$), if the copy at the failed site was a real copy and the generation threshold $GT(O)$ has been crossed. The site at which the data object is regenerated can be determined based on communication cost minimization considerations, node utilization considerations or network reliability considerations depending on the organizational priorities. This module ensures that no data object loses its last real copy, unless there are no more sites to regenerate on.

Step 4: When a site recovers, the DO_RECOVER module is executed by that site. This module first determines whether the site is in the current majority partition with respect to *each* data object stored at that site. For each data object O that has a current majority partition it determines if the $GT(O)$ of that data object has been exceeded. If so, real copies of the data object are converted to virtual copies in accordance with the vector $T(O, j)$ of initial file allocations. Otherwise, the data object O is made current and its state information is updated. The objective of this module is to try to move towards the optimal file allocation stored in the vector $T(O, j)$ for each data object O .

Descriptive code for the RVC algorithm is provided in Figures 2 through 6. Figure 2 describes the *majority* function, which determines whether a given site i belongs to a majority partition with respect to a data object O . The other modules in the algorithm use function *majority* to initiate the collection of a majority of current sites in the present partition.

The notation used in the description is as follows. Suppose the distributed system consists of sites (A, B, C, \dots) . Each data object could be replicated at one or more of the sites. If a data object O has a real copy at site A , then we denote it by A_O . A virtual copy at site A is denoted by A'_O . The state information associated with each data object consists of:

$$v_O = \text{integer}$$
$$U_A^O = (A, B, C, D, E)$$

Figure 1: Motivation for Dynamic Voting with Virtual Copies

Regeneration will also work to convert real copies to virtual copies, when the need arises, e. g. when sites are recovering, resulting in a surfeit of real copies. This will result in a self-regulating (and self-adaptive) system that monitors the number of real and virtual copies of all data objects in the system, and seeks to maintain a predefined level of the number of real and virtual copies in the system.

The motivation for our algorithm can be described with reference to Figure 1. We consider two cases: site failures, and network partitioning. For illustrative purposes, we consider a network with seven sites, numbered one through seven. Assume that the file allocation algorithm determines that a copy of the file should be located at sites 1, 3, 5 and 7.

Consider the case of site failures which is depicted in the first diagram of Figure 1 and suppose that sites start failing in the order of 1, 3, 5 and 7. Even though there has been no partitioning, and sites 2, 4 and 6 are operational, the availability of the file went down to zero (since not even one copy of the file is accessible). At this point any voting based algorithm will not allow operations to continue in any partition. Our proposed algorithm, which is referred to as Regeneration with Virtual Copies (RVC) and is described in Section 4 avoids this problem by keeping virtual copies at sites

2, 4, and 6 and having these virtual copies participate in the voting process. Thus, as soon as sites start to fail, data object copies will be selectively regenerated at other sites. In the context of systems subject to site failures only, we observe that the last copy to fail will be a real copy, and a copy is available as long as at least one site in the network is operational.

In the case of network partitioning, depicted in the second diagram of Figure 1, we start with the initial configuration having data object copies at sites 1, 3, 5 and 7. Suppose that the network partitions into sites 1 and 3 in one partition and sites 5 and 7 in the second partition. The dynamic voting algorithm with linearly ordered copies (where ties are broken in favor of the lexicographically lowest numbered copy) will allow updates and reads in the partition containing sites 1 and 3. If another partition occurs, resulting in having sites 1 and 3 in different partitions, processing will be allowed only in the partition containing sites 1 and 4. The danger here is that if site 1 fails, dynamic voting class of algorithms will allow processing in none of the partitions. It is this shortcoming of the voting class of algorithms that we wish to overcome through our proposal.

The RVC algorithm allows processing to continue by regenerating the virtual copy kept at site 4 to a real copy. Thus, unlike currently available dynamic voting schemes, even if site 1 fails, site 4 will, under the RVC algorithm, still allow operations to continue.

It should be noted that regeneration will lead to sub-optimal allocation of files, since the new allocation obtained by regeneration will not be the same as the optimal solution determined by the file allocation algorithm. However, this sub-optimal mode of operation is preferable to having no data object copies available at all. As sites recover, and/or communication links are repaired, the system can be restored to its original “optimal” configuration.

To summarize, our work combines the advantages of the dynamic voting approach with the regeneration approach to achieve a selective regeneration policy that satisfies consistency constraints. Our algorithm would be effective in such applications as military tracking files, version management of graphics files in CAD/CAM systems, and other image tracking applications involving infrequent updates to relatively large data objects (i.e. data objects requiring large storage). In such environments the RVC algorithm can work in conjunction with a file allocation algorithm to optimize system performance and availability.

of the replicated data object. A partition is defined as a *majority* partition if it contains a majority of the current copies. Associated with each copy at a site i is another integer called the update site cardinality, SC_i . Based upon the values of VN_i and SC_i , certain rules are defined for reading and updating data objects. Rules for merging of data objects are also specified. A recent modification of this algorithm can be found in [11].

Other algorithms based on voting are: **Voting with Witnesses**[16] and **Voting with Ghosts**[18]. The voting with witnesses algorithm proposes data object copies called *witnesses* that can attest to the state of a data object by maintaining state information in the form of the version number. Witnesses can take part in the collection of read quorums. The voting with *ghosts* algorithm proposes data object copies that again carry only state information, but take part in write quorums. The objective of these algorithms is to improve the availability of the DCS.

A different approach to maintaining replicated databases has been adopted by proponents of a technique called regeneration, which is quite similar to file migration. Regeneration in distributed computing systems has been recently suggested in [17, 13, 12, 7] as a mechanism for maintaining consistency of replicated data in distributed computing systems. Distributed databases and distributed file service mechanisms are examples of distributed computing systems.

The algorithm proposed in [17], is an example of a regeneration algorithm that implements a replicated directory system. Such a system allows the selection of arbitrary objects to be replicated, the choice of the number of replicas of each object, and the placement of copies on machines (that are assumed to have independent failure modes). The replication level is restored by automatically replacing lost copies (due to node crashes) on other active sites. The algorithm uses a read one write all strategy (ROWA); if some copies are found to be inaccessible, new replicas are created to replace them. Some limitations of this algorithm are: it is not applicable to network partitioning; it is proposed for distributed systems on a local area network, but not for distributed systems on a wide area network; and it requires additional storage overhead due to replicated directories that are used to point to current copies.

Any mutual consistency protocol that implements mutual exclusion, including voting and dynamic voting involves data transfer over the network. This data transfer is necessary since all copies that participate in an update to any data object have to be made cur-

rent. If the database is distributed over a wide area network, this will involve large data transfers over long distances. This is contrary to the philosophy of distributed database design, which advocates minimizing data transfer over the network as one of its primary objective. This problem is further compounded if updates are numerous. At every update, large data transfer will be required in order to keep the copies of each data object synchronized. Hence, a regeneration approach in wide area distributed computing systems has to minimize data transfer by regenerating *only when absolutely necessary*. In this paper we propose a consistency control algorithm that selectively utilizes regeneration, and also accommodates the case of network partitioning. Our algorithm provides greater availability than previous voting based algorithms by incorporating selective regeneration.

2 Motivation for the Proposed Algorithm

In wide area network environments, the key to using regeneration is to use it selectively. Thus, we propose a policy of selective regeneration through a mechanism that is referred to as *generation threshold* ($GT(O)$). Initial allocation of data objects to nodes of the DCS can be performed by a file allocation algorithm such as the algorithms proposed in [3, 15, 14, 5, 19]. $GT(O)$, therefore represents the optimal number of file copies to be maintained in the DCS for each data object. We maintain an allocation table $T(O, j)$, whose row indices represent data objects and column indices represent the sites at which data object O has been initially allocated. The table $T(O, j)$ can be a zero-one matrix, where an element T_{Oj} is set to 1 if site j has a copy of the data object O , and 0 otherwise.

Dynamic voting as proposed in [4, 8, 10] requires real copies of data objects at all sites at which the data object is replicated. We propose to integrate *Virtual Copies* with dynamic voting. This is achieved by maintaining some copies of each data object as virtual copies, which contain only state information and participate in the voting process both for read and write quorums. We also impose the constraint that each quorum must contain at least one full copy of the data object.

Regeneration in our protocol refers to converting a virtual data object copy to a real data object copy, whenever the generation threshold has been crossed. This ensures that the last copy to fail is a real copy.

Regeneration with Virtual Copies for Replicated Databases

Nabil R. Adam

Rajiv Tewari

Graduate School of Management
Rutgers University
Newark, NJ 07102

Computer & Information Sciences
Temple University
Philadelphia, PA 19122

Abstract

We consider the consistency control problem for replicated data in a distributed computing system (DCS) and propose a new algorithm to dynamically regenerate copies of data objects in response to node failures and network partitioning in the system. The DCS is assumed to have strict consistency constraints for data object copies. The new algorithm combines the advantages of voting based algorithms and regeneration mechanisms to maintain mutual consistency of replicated data objects in the case of node failures and network partitioning. Our algorithm extends the feasibility of regeneration to DCS on wide area networks, and is able to satisfy user queries as long as there is one current partition in the system.

1 Introduction

In a distributed computing environment, two types of failures may occur: the processor at a given site may fail (referred to as site failure), and communication between two sites may fail (referred to as communication link failure). When a site fails, processing at that site stops and the contents of the volatile storage are destroyed. Communication links may fail due to such reasons as noise in the link, or temporary link malfunction.

Link failures may result in *network partitioning*, isolating the network into two (or more) connected components, with nodes within a given component being able to communicate with one another but not with nodes within other components. If we represent the distributed computing system by a network where nodes are sites and arcs are links, then partitioning divides the operational sites into two or more components. Each component is referred to as a *partition*. Since these components cannot communicate, mutual consistency

of replicated data can be preserved only if user requests are allowed to be processed at one, and only one partition. Such a partition is referred to as the *majority partition*. Consistency control algorithms ensure that user requests are processed in such a manner that mutual consistency of all data objects is preserved when site failures and/or network partitioning occur.

Consistency control algorithms can be classified according to whether they can handle site failures only, or both site failures and network partitioning. Two representative algorithms that can handle site failures only, are ROWAA (Read-One-Write-All-Available) [2] and Available Copies[1]. These algorithms function by reading *any* available copy of the data object (preferably local) and writing into all available copies. Under these algorithms a user query can be always satisfied as long as at least *one* copy of the desired data object is available. Updates of a data object, in this case, will always be satisfied as long as the last copy of a data object is not lost.

Voting algorithms for preserving mutual consistency of replicated data by mutual exclusion have been proposed by Thomas[20] and Gifford[6]. Voting type of algorithms can handle network partitioning in addition to site failures. This is achieved by a process of quorum collection which lets a network partition decide autonomously whether it constitutes a majority of current copies of a data object. If so, query and update operations would be allowed to proceed. The voting algorithms proposed in [20, 6] utilize a static quorum, which results in a limited database availability.

The dynamic voting algorithm proposed by Jajodia and Mutchler [8] and later refined in [9, 10], defines a *version number*, VN_i , of a copy, which counts the number of successful updates to the data object. The *current version number* is the maximum of the version numbers of all copies of a data object. A copy is current if its version number equals the current version number