MUTUAL EXCLUSION IN PARTITIONED DISTRIBUTED SYSTEMS

Daniel Barbara
Hector Garcia-Molina

Department of Computer Science

CS-001

July, 1985

MUTUAL EXCLUSION IN PARTITIONED DISTRIBUTED SYSTEMS

Daniel Barbara

Hector Garcia-Molina

Department of Computer Science

Princeton University

Princeton, New Jersey 08544

ABSTRACT

A network partition can break a distributed computing system into groups of isolated nodes. When this occurs, a mutual exclusion mechanism may be required to ensure that isolated groups do not concurrently perform conflicting operations. We study and formalize these mechanisms in three basic scenarios: where there is a single conflicting type of action; where there are two conflicting types, but operations of the same type do not conflict; and where there are two conflicting types, but operations of one type do not conflict among themselves. For each scenario, we present applications that require mutual exclusion (e.g., name servers, termination protocols, concurrency control). In each case, we also present mutual exclusion mechanisms that are more general and that may provide higher reliability than the voting mechanisms that have been proposed as solutions to this problem.

1.INTRODUCTION

In a distributed computing system, a partition occurs when one or more groups of nodes become isolated from the rest of the system due to a failure of the communications network. All the operational sites in the same group can communicate with each other but they cannot communicate with sites in other groups. In many systems it is impossible to differentiate between a single site failure and a partition (noticeably SDD-1 [BERN77],[BERN78],[HAMM80]), so "logical" network partitions are much more common that what one may expect.

A reliable distributed system must be able to cope with network partitions. This usually means that an isolated group of nodes must attempt to continue performing the task the system was originally assigned. However, the isolated group must do this without knowing how many other active groups there exist and what they may be doing. After a partition, the various groups cannot decide on a common strategy to follow.

In many applications, it becomes necessary to provide a *mutual exclusion* mechanism that works in spite of these catastrophic failures. Specifically, we have a restricted operation that must be executed by at most one of the groups. For instance, in a naming system that generates unique names, we probably do not want isolated groups to concurrently generate different names for the same object. So, if a group is going to perform the naming, it must be able to guarantee that no other group is performing this activity. This mutual exclusion has to be enforced without communication between groups.

Another problem were mutual exclusion arises is in controlling access to replicated data items in a distributed database, where executing transactions can read and write these items. Many concurrency control algorithms have been proposed for this problem, and a good survey of them can be found in [BERN81]. Basically, before committing, a transaction must ensure itself the control over all the copies of all the data items involved, henceforth excluding any other competing transaction from changing the values of those items or reading data values that are to be immediately updated. When partitions can occur, a transaction must also exclude transactions in other groups from executing updates that conflict with its updates.

Traditionally, mutual exclusion in partitioned distributed systems has been implemented by using *voting mechanisms.* That is, each node is a-priori assigned a number of *votes*, and only the group with a majority of the total votes is allowed to perform a restricted operation [e.g., DAVI82,GIFF79,THOM79, GARC82]. Notice that the nodes do not need to identify the presence of partitions before updating data or generating a new name: they merely have to collect enough votes from other sites to insure a majority.

The mutual exclusion problem in the naming and the concurrency control applications is similar but not identical. In naming, there is a single restricted operation: in concurrency control there are two: reading and writing the data. Transactions that simply read data can execute concurrently in different groups. Update transactions must exclude other competing updates and reads. Furthermore, there are other applications (to be discussed in this paper) where mutual exclusion follows different rules from the ones in these examples. Thus, a number of natural questions arise: How are the mutual exclusion scenarios related? How are the solutions for each scenarios related? Is voting the only way to achieve mutual exclusion? How should votes be assigned to nodes in order to increase the "reliability" of the system, i.e., to reduce the likelyhood that during a partition no group has a majority of votes?

The goal of this paper is to answer these and other related questions. In Section 2, we develop a framework for studying and classifying the various mutual exclusion scenarios. We also briefly summarize various applications where mutual exclusion arises, and show how they fit in our framework. The rest of the paper (Sections 3,4 and 5) is devoted to studying particular scenarios, discussing techniques for analyzing and improving the reliability of mutual exclusion mechanisms. (The scenario of Section 3 has been studied in an earlier paper [GARC85]; here we summarize the main results for this case.)

2. A FRAMEWORK FOR MUTUAL EXCLUSION IN DISTRIBUTED SYSTEMS

In a distributed system, nodes perform tasks or execute transactions, and it is these operations that have mutual exclusion requirements.

Definition 2.1 *Mission*: A *mission* is an activity undertaken by a node or group of nodes. Missions are classified into *types*. It is a priori decided whether each pair of types is *compatible* or *mutually exclusive*: two missions of compatible types can be executed concurrently by isolated groups of nodes while mutually exclusive ones cannot.○

For instance, during a partition, a group of nodes undertakes the mission of updating the database. Other groups in the system may try to undertake the same mission at the same time.

Definition 2.2 *Competitive mission type:* We say that a mission type is *competitive* if every pair of missions of this type are mutually exclusive. ○

Definition 2.3 *Non competitive mission type:* We say that a mission type is *non competitive* if every pair of missions of this type are not mutually exclusive. ○

The update mission is an example of a competitive mission type, and the read mission of a non competitive one.

In this paper we only study scenarios with one or two mission types. Scenarios with three or more mission types can always be broken into the fundamental one and two mission type subproblems, so we do not cover them. With this in mind we can make a simple classification of mutual exclusion scenarios.

Definition 2.4 *Unilateral mutual exclusion scenario* A mutual exclusion scenario is unilateral if it comprises only one type of mission. ○

Of course, the mission type in an unilateral scenario must be competitive (otherwise there would not be a mutual exclusion problem at all).

Definition 2.5 *Bilateral mutual exclusion scenarios:* A mutual exclusion scenario is bilateral. if it contains two types of missions. ○

Observe that a bilateral scenario where both mission types are competitive is equivalent to a unilateral scenario with a competitive mission type (i.e., in both cases, all missions exclude each other). Thus, we are left with only two bilateral cases: (a) no competitive mission types. and (b) one competitive type. If we add to this the unilateral scenario with competitive type. we have the three basic scenarios that we will consider in this paper: the unilateral case is covered in

Section 3, the bilateral with no competitive types in Section 4, and the bilateral with one competitive type in Section 5.

In closing this section we survey some of the most common mutual exclusion problems for distributed systems, and classify them using our terminology.

Unilateral scenarios

Restricted Updating

This is a classical example of a unilateral scenario. The mission type undertaken by each group is to try to remain operative, i.e., to perform updates on a replicated database. The system does not distinguish between read-only and update transactions; thus, read-only transactions are handled as if they were updates.

As discussed in the introduction, the best known solution to achieve mutual exclusion in this scenario is to a priori assign a number of *votes* (or points) to each node with a copy of the database, and a group whose members have a majority of the total votes is allowed to perform the restricted operation [e.g., DAVI82, THOM79, GARC82]. Mutual exclusion is achieved because at most a single group can have a majority of votes at a time. It is possible that at a given time no group has a majority and can perform the operation. There seems to be no way to avoid this problem. Even giving one node all votes does not help since that node may fail.

Notice that this problem and its solution can be generalized to the case where data is not fully replicated. For instance, consider a system with 4 nodes, *a*, *b*, *c* and *d*, and a database consisting of 3 fragments. Fragment *X* is replicated at nodes a, b and c; fragment Y is only located at site a; and fragment Z is found at all four nodes. Here, three independent sets of votes are used, each to control access to a fragment. The X votes are distributed among a, b and c; the Y votes to a; and the Z votes to all nodes. (It is also possible to give X and Y votes to nodes without a copy.) If a transaction wishes to access fragments X and Y, it must secure a majority of the X and Y votes. In this case there are 3 competitive mission types, but each type is compatible with the other. As stated earlier, for analysis the problem can be decomposed into 3 independent unilateral scenarios (one for each fragment), and this is how we will study it in this paper.

Also notice that a transaction can never rely on the fact it is in a group with a majority of votes (the group can disintegrate at any time), so it must secure itself the necessary votes before committing. Thus, a protocol for identifying groups after a partition is not necessary. However, for ease of presentation, we will speak in this and other scenarios of a "majority group" as if it were the entity that executes the transactions.

A Directory Server

Consider a directory server that must be unique in the system because different objects with the same name or different names for the same object are undesirable. Here again, during a partition, at most one group should have an active server, and this mutual exclusion can be achieved through voting.

Reliable Computation

In an N-modular redundant system, nodes may fail and yield incorrect or even misleading results. The computation being performed is replicated at N sites, and if nodes with a majority of votes agree on a result, it is considered correct [e.g., LAMP82,DOLE82,LYNC82].

Here every node or group of nodes has the mission of forcing its result over the rest of the system. This is the only mission type in this scenario. Nodes that can communicate with each other and have the same result cooperate towards the same mission and are considered to be in the same "partition". Two groups of nodes with different results are engaged in mutually exclusive missions.

Bilateral Scenarios with one competitive mission

The Concurrency Control Problem

Transactions are composed by read and write operations that access the database concurrently. Two operations conflict if they try to access the same data item and at least one of the operations is a write. Concurrency control is the activity of controlling the relative order of conflicting operations so the final result is equivalent to performing the transactions in serial order. We can distinguish between two types of conflicts. If transaction T_i issues a read over the item x and transaction T_j issues a write over the same item, we talk about a *read-write* (*rw*) conflict. On the other hand, if both transactions try to perform a write over x, we talk about a *write-write* (*ww*) conflict. Many concurrency control mechanisms have been proposed [BERN81]. Here we will present two to illustrate. (Others will be discussed in Section 5.) In the first, each item (or set of items) is controlled by a lock manager at a single site. The managers for different items can be at different sites, and usually the manager is located at a node with a copy of the item. The lock manager can issue read locks that give a transaction the right to read the item, and write locks which give read/write privileges. As a transaction executes, it must obtain the appropriate locks before accessing data. If a partition makes a required manager inaccessible, then the transaction must wait. If the transactions and manager follow the locking rules, including two-phase locking (after releasing a lock, a transaction cannot request additional locks), then it can be shown that the execution of the transaction is equivalent to some serial execution.

Gifford [GIFF79] devised and implemented a different type of mechanism, one that uses voting for concurrency control. A total of T votes are distributed among the nodes containing a copy of the database. If a transaction wishes to read and update the database, it must collect a write quorum of w votes; if it only wishes to read, it collects a read quorum of r votes. If $r + w > T$ and $2w > T$, then conflicts are avoided. At most one group can be updating the database at a time; reads do not exclude each other, but one reader excludes any writers. As we have described it, the mechanism treats the entire database as a single unit. However, in this case too we can fragment the database and have independent vote assignments for each fragment.

In our terminology, a transaction embarks on a series of read or update missions, one for each item (or fragment) it accesses. In an update (read) mission, the node where the transaction runs attempts to gain control of an item and its copies for updating (reading). Missions on different items are compatible, so we concentrate on a single item and its two mutually exclusive mission types: non competitive reads and competitive updates.

Bilateral Scenarios with no competitive missions

Termination protocols

When a failure occurs during the processing of a transaction, the protocol used for committing the transaction might be in different states at the different sites where the transaction runs. A possibly different protocol that tries to terminate the incomplete transaction at the operational sites must be invoked. These types of protocols are known as termination protocols [SKEE81]. Faced with the problems of dealing with partitions in the network, Skeen [SKEE82] developed a class of termination protocols called *quorum based* protocols.

Under a quorum based protocol, the fate of a transaction in an isolated group is decided by collecting a majority of votes. A transaction must collect a commit quorum of V_c votes before it is committed by any site. Otherwise it must collect an abort quorum of V_a votes before it is aborted. Of course $0 < V_a, V_c \leq V$, where V is the total number of votes in the system. To prevent groups that are isolated from each other from independently deciding to abort and commit, we must have $V_a + V_c > V$.

The abort and commit agreements can be thought as missions undertaken by the groups. Obviously, they are mutually exclusive. This scenario is therefore a bilateral scenario in which both missions are non competitive. It does not matter if two groups decide to commit the transaction.

3. UNILATERAL SCENARIOS

In this section we treat the problem of achieving mutual exclusion in scenarios with only one mission. These can be viewed as problems in which we have a restricted operation that should be performed by at most one group of the system at any given time. The mutual exclusion has to be enforced without communication between groups.

As discussed in Section 2, one way to implement mutual exclusion for this scenario is by means of voting. A second solution to this mutual exclusion problem was suggested by Lamport in 1978 [LAMP78], but because it *appears* to be so similar to vote assignment, it has received little attention. The idea is to a-priori define a set of groups that may perform the restricted operation. Each pair of groups should have a node in common to guarantee mutual exclusion. For example, if we have nodes a,b, and c we may define the set $\{\{a,b\},\{b,c\},\{a,c\}\}$. Nodes a and b can together perform the operation, knowing that neither group $\{b,c\}$ or $\{a,c\}$ can be formed. Notice that this set of groups is equivalent to assigning 1 vote to each node (or n votes to each node).

The assignment of votes or the choice of set of groups can have a critical effect on the reliability of a distributed system. Consider for example, a system with nodes a,b,c, and d and an assignment that gives one vote to each. This seems like a natural choice because it gives each node equal weight. Since 3 votes are needed for a majority, this is equivalent to the set of groups

$$S = \{\{a,b,c\},\{a,b,d\},\{a,c,d\},\{b,c,d\}\}$$

But now, consider an assignment that gives node a 2 votes and the rest a single vote. The majority is still 3, so this is equivalent to

$$R = \{\{a,b\},\{a,c\},\{a,d\},\{b,c,d\}\}$$

Set R and its associated vote assignment is clearly superior to S because all groups of nodes that can operate under S can operate under R, but not vice versa. For instance, a and b can form a group under R but not under S. So, if the system splits into groups $\{a,b\}$ and $\{c,d\}$, there will be one active group under R but none under S. So clearly, no system designer should *ever* select set S (or its equivalent vote assignment), in spite of the fact it seems "natural."

We use the term "*R dominates S*" to mean that R is always superior to S. Obviously, we want to ignore dominated sets (or vote assignments). But even if we do, we must still select one of the non-dominated sets, and this is no easy task. In our example, which of the nodes should get the 2 votes? Or should we give one node 1 vote and the rest 3 votes? Or 4, 3, 2 and 2 votes?

There are many choices, but many are duplicates. For example, giving a 4 votes, b 3 votes, and c and d 2 each, yields exactly the same set of groups that was given by R. So again, in the selection process we want to ignore duplicate vote assignments.

The concept of set of groups proves to be useful, not only because it facilitates the analysis of vote assignments, but because it is a more "powerful" mechanism for mutual exclusion, as we will see in a moment. In the rest of this section we summarize the most important properties of vote assignments and sets of groups. These properties will be useful in understanding how these mechanisms operate and for selecting "good" assignments or sets of groups. A more detailed discussion of these properties appears in [GARC85]; theorems and lemmas will be stated here without proof.

We start by defining sets of groups and domination. Notice that to avoid confusion we are referring to sets of nodes as *groups*. Sets of groups are thus sets of sets of nodes. In dealing with mutual exclusion, we do not want to have sets like $\{\{a\},\{b,c\}\}$ or $\{\{a\},\{a,b\}\}$. The former set would allow two different groups to be active at the same time, while in the latter the group $\{a,b\}$ is redundant.

We will use the term *coterie* (from the French) to refer to the sets of groups that are "well formed." (According to Webster's dictionary, a coterie is a "close circle of friends who share a common interest... A set refers to a group, usually larger and, hence, less exclusive than a coterie.")

Definition 3.1 *Coterie.* Let U be the set of nodes which compose the system. A set of groups, S, is a *coterie* under U iff

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

(ii) (Intersection property) If $G, H \in S$, then G and H must have at least one common node.

(iii) (Minimality) There are no $G,H \in S$ such that $G \subset H$.

When U is understood, we will drop it from the discussion. ○

Note that not all nodes must appear in a coterie. For instance, $\{\{a\}\}$ is a coterie under $\{a,b,c\}$.

Definition 3.2 *Domination for Coteries.* Let R, S be coteries (under U). R *dominates* S iff $R \neq S$ and for each $H \in S$ there is a $G \in R$ such that $G \subseteq H$. (We say that G is the group that dominates H.) ○

Definition 3.3 *Dominated and Non-Dominated Coteries.* A coterie S (under U) is *dominated* iff there is another coterie (under U) which dominates S. If there is no such coterie, then S is *non-dominated* (ND for short). ○

As discussed earlier, a dominated coterie should not be used because there is a coterie that provides more protection against partitions. For instance, the coterie

$$\{\{a,b,c\},\{c,d,e\}\}$$

should be replaced by $\{\{c\}\}$, and the coterie

$$\{\{a,b\},\{b,c\}\}$$

should be replaced by

$$\{\{a,b\},\{a,c\},\{b,c\}\}$$

The next theorem gives us a way to check if a coterie is ND without enumerating all other coteries. This will be useful later on.

Theorem 3.1 Let S be a coterie under U. S is dominated iff there exists a group $G \subseteq U$ such that

(i) G is not a superset of any group in S.

(ii) G has the intersection property. That is, for all $H \in S$, $G \cap H \neq \emptyset$.○

Checking domination of coteries seems to be a hard problem. The best algorithm that we know at this point is the one suggested by Theorem 3.1. It generates all the possible subsets of the universe of n nodes, and for each one, checks if it can be added to the coterie. This algorithm is clearly exponential in n. The worst case complexity of an algorithm to check for non-domination is, however, an open problem.

It is clear that given a vote assignment we can always find the corresponding coterie. It is enough to find the tight majority groups (groups for which the deletion of any element breaks the vote majority). The set of tight majority groups is clearly a coterie since it satisfies the properties of definition 3.1. Thus, the fundamental question is whether every coterie can be represented by a vote assignment. If this were true, then coteries and vote assignments would be equivalent concepts. The following theorem settles this question.

Theorem 3.2 There are ND coteries such that no vote assignment corresponds to them.○

The following coterie is an example of such a coterie:

$$S = \{\{a,b\}\{a,c,d\},\{a,c,e\},\{a,d,f\},$$
$$\{a,e,f\},\{b,c,f\},\{b,d,e\}\}$$

If we try to set up linear inequalities to describe this coterie as a vote assignment (e.g., votes(a) + votes(b) $\geq$ Majority of votes), we arrive at a contradiction.

Theorem 3.2 tells us that coteries are a more powerful concept than vote assignments. That is, in some systems a coterie like the one in the previous example could actually yield the best reliability characteristics, and there would be no way to enforce the same groups with votes. However, vote assignments have some advantages over coteries, namely they are easier to implement. The Venn diagram of Figure 1 summarizes the classes of coteries.

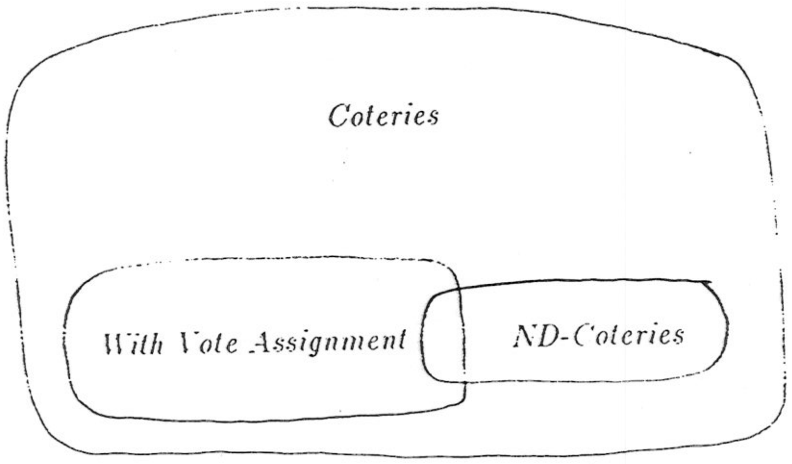

Figure 1

Another important question concerns the number of possible ND coteries for a system with n nodes.

Theorem 3.3 The value $2^{2^{cn}}$ for some constant c, is a lower bound of the number of ND coteries under a universe of n nodes. ○

Theorem 3.3 shows unequivocally that any attempt to exhaustively enumerate ND coteries is hopeless for large n. Thus, it will not be possible to conduct an exhaustive search for the optimal coterie for a system. However we have made two important observations:

(1) The number of ND coteries for small systems (5 nodes or less) is small. This is interesting since these will be likely systems in practice. (Remember that each node has a copy of the database or a potential name server.) Figure 2 shows the ND coteries for systems with 5 nodes or less. For example, for a 4-node system there are 3 classes: (a) give all the votes to a single node (e.g., $\{a\}$); (b) give three of the nodes an equal number of votes (e.g., $\{ \{a,b\},\{a,c\},\{b,c\} \}$); and (c) give all the nodes an equal number of votes, except for one node that receives an extra vote ($\{ \{a,b,c\},\{a,d\}, \{b,d\},\{c,d\} \}$). Also observe that for 5 or less nodes all coteries have equivalent vote assignments.

(2) Even for large systems, we can enumerate a subset of the class of ND coteries that includes those with vote assignment. This class results in a more manageable number of choices.

With these results we can narrow down the number of choices that must be considered by a system designer. To select one of the remaining vote assignments or coteries, we need metrics that evaluate the "reliability" provided by each choice. For example, we could use one of the following two metrics:

a) **Node Vulnerability:** The minimum number of node failures that produces a state where no group is active (e.g., updating the database).

for 1 $node\ systems = [\{\ \},\ \{\{a\}\}]$

for 2 $node\ systems = [\{\ \},\ \{\{a\}\}]$

for 3 $node\ systems = [\{\ \},\ \{\{a\}\},\ \{\{a,b\},\{a,c\},\{b,c\}\}]$

for 4 $node\ systems = [\{\ \},\ \{\{a\}\},\ \{\{a,b\},\{a,c\},\{b,c\}\},$

$\{\{a,b,c\},\{a,d\},\{b,d\},\{c,d\}\}]$

for 5 $node\ systems = [\{\ \},$

$\{\{a\}\},$

$\{\{a,b\},\{a,c\},\{b,c\}\},$

$\{\{a,b,c\},\{a,d\},\{b,d\},\{c,d\}\},$

$\{\{a,b,c\},\{b,d\},\{c,d\},\{b,c,e\},\{a,d,e\}\},$

$\{\{a,b,c\},\{c,d\},\{b,c,e\},\{a,d,e\},$

$\{a,c,e\},\{b,d,a\},\{b,d,e\}\},$

$\{\{a,b,c\},\{b,c,e\},\{a,d,e\},$

$\{a,c,e\},\{b,d,a\},\{b,d,e\},$

$\{a,b,e\},\{c,d,a\},\{c,d,b\},\{c,d,e\}\},$

$\{\{a,b,c,d\},\{a,e\},\{b,e\},\{c,e\},\{d,e\}\}]$

Figure 2

b) **The steady state probability that the system is active:** It is the probability that the system has an active group of nodes, given the failure characteristics of each component in the system.

To illustrate the use of these two metrics, consider the system of Figure 3. Assume that for every node, the steady state probability of being active is equal to p, while the links are perfectly reliable (they are always functioning). We can compare the two ND Coteries for this 3 node system, that is the *singleton* coterie $\{a\}$ and the *uniform* coterie $\{\ \{a,b\},\{a,c\},\{b,c\}\ \}$, using the two metrics defined above.

The node vulnerability of the singleton coterie is one (if node a is down, the database cannot be updated), while that of the uniform coterie is 2 (two failures are needed to disrupt the system).

As for the probability that the system is active, P, it will be p under the singleton coterie, and

$$P = p^3 + 3p^2(1 - p)$$

under the uniform coterie. (at least two of the nodes have to be active). For $0.5 < p < 1.0$, it will be best to use the uniform coterie.

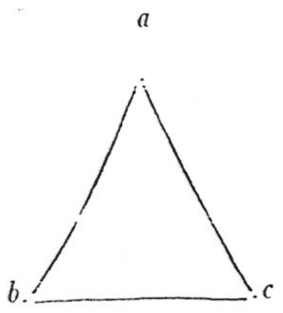

Figure 3

4. BILATERAL SCENARIOS WITH NO COMPETITIVE MISSIONS

In this section we study bilateral scenarios in which neither mission is competitive. Let us consider termination protocols once again.

Quorum protocols [SKEEN82] assume an a-priori distribution of votes among the nodes. A transaction must collect a commit quorum of V_c votes before it is committed, or an abort quorum of V_a before it is aborted, by any group under a partition. (Of course, if any of the sites in the group is already in commit or abort state, the transaction is committed or aborted). To prevent two unconnected groups from taking contradictory decisions, we must have $V_a + V_c > V$.

Skeen observes that the selection of the vote distribution and the values of V_a and V_c affect the performance of the algorithm, and suggests several ways to optimize these values. However, the selection of V_a is not as independent from the selection of V_c as it may seem at first.

Let us consider the following example: a three node system with nodes a, b and c and a total of V votes. Assume that we select $V_c = V$ (a total commit quorum). This means that the only group that can commit transactions is the one containing all votes. Let w_i be the number of votes that node i receives. Consider the following cases:

1 $w_a = w_b + w_c$ and $V_a = w_a$. Then the groups that can abort a transaction are $\{a\}$ and $\{b,c\}$. (Any other group is a superset of one of them.)

2 $w_a = w_b = w_c$ and $V_a = V$. Here $\{a,b,c\}$ is the only quorum group for aborting a transaction.

3 $w_a = V$ and $w_b = w_c = 0$, obviously $V_a = V$. And $\{a\}$ is the only quorum group for abortion.

4 $w_a = w_b = w_c$ and $V_a = w_a$. Here the groups are $\{a\},\{b\},\{c\}$.

It is easy to see that under a partition, if we have a group able to decide on aborting the transaction under cases 1 through 3, that same group will be able to do so under case 4. (Observe that any of the abort groups of cases 1 through 3 is contained in at least one group of case 4.) Therefore, given that one selects $V_c = V$, it does not make sense to have either choices 1,2 or 3 for the abort quorum. In other words, our choices for quorums seem to be restricted, just like the choices for coteries were restricted to ND ones. Notice also that the groups in case 4 correspond to the minimal transversals of the commit quorum group $\{a,b,c\}$. The result is not a coincidence and can be generalized easily. We need a few definitions before doing so. The first one is a generalization of the concept of coteries.

Definition 4.1 *Quorum set:* A *quorum* set is the set of groups that can complete a mission. More formally, let U be the set of nodes which compose the system. A set of groups, Q, is a quorum set under U iff

(i) $G \in Q$ implies that $G \neq \emptyset$, and $G \subset U$.

(ii) (Minimality) There are no $G,H \in Q$ such that $G \subset H$. ○

We take the name of quorum sets from the example of quorum based protocols. Notice that definition 4.1 is similar to the definition of coteries, except that the intersection property has been dropped because the missions are not competitive. For instance, in quorum protocols two different groups can be committing the transaction at the same time.

The definition of domination used for coteries (Definition 3.2) is also applicable in the case of quorum sets.

Definition 4.2 *Quorum domination:* We say that a quorum set Q *dominates* Q_1 when all the groups in Q_1 are supersets of some group in Q. ○

As an example, if:

$$Q_1 = \{ \{a,b\},\{c,d\} \}$$
$$Q = \{ \{a\},\{c\} \},$$

Then, Q dominates Q_1.

In bilateral scenarios, we have two quorum sets, one for each mission type. In the example of quorum protocols we talk about the *commit quorum set* or the *abort quorum set*.

Both quorum sets must be designed in such a way that any group in one of them precludes every group of the other from being formed. In this way we ensure the mutual exclusion between the two missions. Thus, if a commit (abort) quorum is going to be used in conjunction with an abort (commit) quorum, every group in one must intersect every group in the other.

Definition 4.3 *Complementary quorum set:* Given a quorum set, a *complementary quorum set* is another quorum set, such that every group in one intersects every group in the other. ○

For example if we have:

$$Q_c = \{ \{a,b\},\{c,d\} \}$$

as the commit quorum set, then

$$Q_a = \{ \{a,c\},\{a,d\},\{b,c\},\{b,d\} \}$$

is a suitable choice for the abort quorum set.

Given a quorum set for one of the missions, there are several choices for the other. Among them, one of the choices is particularly important and it is the subject of our next two definitions.

Definition 4.4 *Transversal:* A transversal of a quorum set Q under U is defined to be a set $T \subset U$ such that for every group $G \in Q$, $G \cap T \neq \emptyset$. A *minimal transversal* is a transversal such that no proper subset of it is a transversal. ○

Definition 4.5 *Antiquorum set:*. An ***antiquorum set*** is the set of minimal transversals of a quorum set○

For instance, in our example above, Q_a is the antiquorum of Q_c. We will denote the antiquorum of a quorum Q as Q^{-1}.

We are now ready to show the importance of the antiquorum.

Theorem 4.1 Given a quorum set Q, any complementary quorum set is dominated by Q^{-1}.

Proof: Let R be any complementary quorum set for Q with $R \neq Q^{-1}$. Let $J \in R$ and $J \notin Q^{-1}$. Since $J \cap G \neq \emptyset$ for all $G \in Q$, J is a transversal of Q. Now, since Q^{-1} contains all the minimal transversals of Q, there exists a $H \in Q^{-1}$ such that J is a superset of H. Since J was not in Q^{-1}, then J must be a superset of H. Since this is true for all J in R, then Q^{-1} dominates R.○

Theorem 4.1 implies that once the quorum set is chosen, the best complementary set to chose will be the antiquorum. Hereafter, we will use the antiquorum as the understood choice.

Definition 4.6 *Quorum agreement.* The set $q = \{Q, Q^{-1}\}$ is called the *quorum agreement.*○

Observation 4.1: There are quorum agreements in which neither the quorum or the antiquorum is a coterie○

The quorum agreement $q = \{ Q_c, Q_a \}$ presented earlier is an example of this type of quorum agreement.

As we have mentioned, coteries are a subclass of quorum sets. In fact, they restrict the behavior of the system in the following way.

Observation 4.2. Whenever one of the quorum sets in a bilateral scenario is a coterie, at most one group will be able to complete the corresponding mission. That is, the mission type is forced to be competitive.○

For instance, if the commit (abort) quorum is a coterie, only one group at most will be able to commit (abort) the transaction during a partition.

Theorem 4.2 Every bilateral mutual exclusion scenario implemented using voting techniques will result in at least one of the quorum sets being a coterie.

Proof: Let $q = \{ Q, Q^{-1} \}$ be the quorum agreement. Let N be the set of nodes in the system and $u(a)$ be the votes that node a has. Every group G in Q must obey:

$$\sum_{a \in G} u(a) \geq V_c$$

similarly for every J in Q^{-1}:

$$\sum_{a \in J} u(a) \geq V_a$$

Now, since $V_a + V_c > V$ and $V = \sum_{a \in N} u(a)$ we have that either V_a or V_c (or both) are a majority (i.e., greater than half of the votes). Therefore the groups must intersect.○

The following is an important corollary:

Corollary 4.1 Every quorum agreement that does not have a coterie does not have a corresponding vote assignment.

Again the quorum agreement of our first example, is also an example for corollary 4.1. However, there are other quorum agreements that do not have vote assignments. It is enough to consider any coterie without a corresponding vote assignment. (Theorem 3.2.)

As a consequence of Theorem 4.2 we can state that every quorum protocol devised by Skeen corresponds to a quorum agreement in which at least one of the two components (quorum or antiquorum) is a coterie. Figure 4 presents the classes of quorum sets.

If we restrict ourselves to voting mechanisms, we will not be able to implement true bilateral scenarios in which none of the missions is competitive. Voting limits our power, forcing one of the missions to compete with itself. All the quorum based protocols devised by voting mechanisms force the commit or abort to be limited to one group in the system. Aside from this, the more general concept of quorum sets allows choices that might very well be better choices under some metrics.

To illustrate, consider the system of Figure 5, and the quorum agreement:

$$Q = \{ \{a,b\},\{c,d\} \}$$
$$Q^{-1} = \{ \{a,d\},\{b,c\},\{a,c\},\{b,d\} \}$$

which is *not* implementable by voting schemes (by Theorem 4.2). It represents a very simple and efficient quorum protocol in which to commit (abort) a transaction, a node needs only to communicate with a single, immediate neighbor. For example, node a can abort the transaction if it can communicate with b and it can abort it if it can communicate with d.

When a quorum set is a coterie, we call its antiquorum an *anticoterie*. An

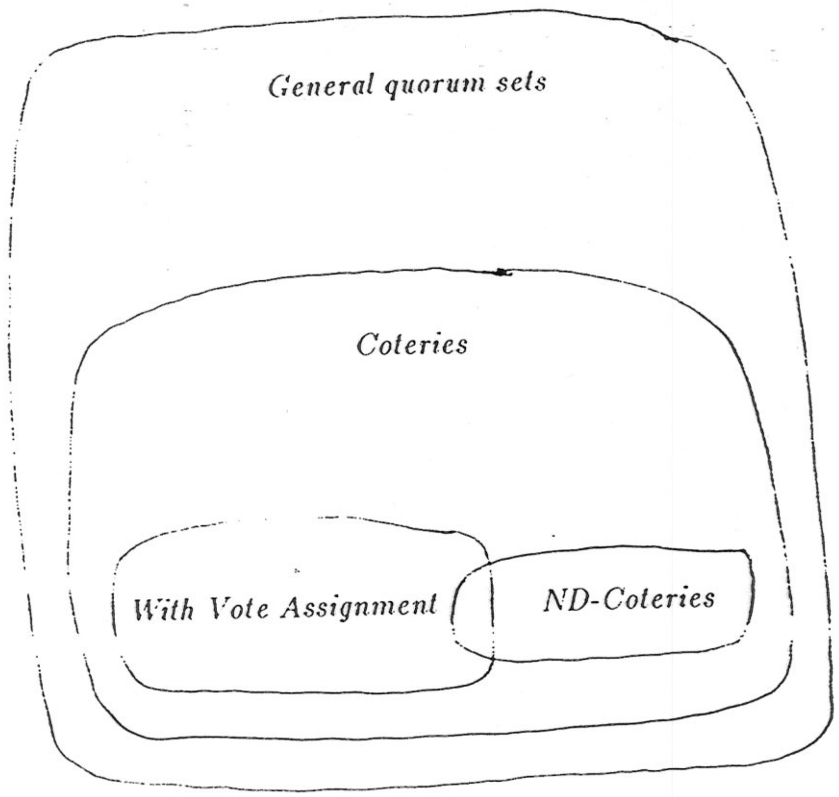

Figure 4

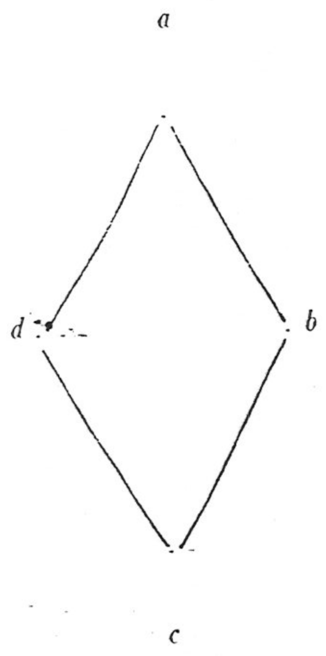

Figure 5

anticoterie is not necessarily a coterie. The following example illustrates the case.

$$Q = \{ \{a,b\},\{a,c\} \}$$
$$Q^{-1} = \{ \{a\},\{b,c\} \}$$

Notice that Q is not an ND-Coterie, in fact, if it were, the corresponding anticoterie would be a coterie also. The following example illustrates the point.

$$Q = \{ \{a,b\},\{a,c\},\{b,c\} \}$$
$$Q^{-1} = \{ \{a,b\},\{a,c\},\{b,c\} \}$$

We can prove that this is the case in general and moreover, if both quorum sets are coteries, they must be the same ND-Coterie.

Theorem 4.3 If Q is an ND-Coterie then $Q^{-1} = Q$.

Proof: From the definition of ND Coterie, it is easy to see that every group in Q is a minimal transversal and is hence in Q^{-1}. To show that Q^{-1} has no additional groups, consider some $J \in Q^{-1}$ such that J is not a superset of any group in Q. By Theorem 3.1, Q is then dominated, which contradicts the statement of the Theorem. Thus, $Q = Q^{-1}$.○

Theorem 4.4 If in the quorum agreement $q = \{Q,Q^{-1}\}$ both of the quorum sets are coteries, they must be the same ND-Coterie.

Proof: Assume that both quorum sets are coteries but Q is not an ND-Coterie. Then according to Theorem 3.1 there must be a group, G not in Q that has the intersection property and is not a superset of any group in Q. Therefore G should be a member of Q^{-1}. Notice also that $\overline{G}$ has the same properties: if $\overline{G}$ did not intersect with some group $J \in Q$, then G would be a superset of J. Similarly, if $\overline{G}$ were a superset of some $J \in Q$ then G would not intersect with J. Therefore, $\overline{G}$ must also be a member of Q^{-1}. But, if G and $\overline{G}$ are members of Q^{-1}, then Q^{-1} is not a coterie as was assumed. Thus, both are ND-Coteries and according to Theorem 4.3 they are the same coterie○

Theorem 4.4 shows that there are only three possible situations:

- Both the quorum and antiquorum are the same ND-Coterie. In this case we are in the presence of an scenario that behaves as unilateral.
- One of the two quorum sets is a coterie, but not an ND-Coterie, while the other is not a coterie. This scenario is a bilateral one, in which the set that is a coterie is implementing a competitive mission type.
- Neither set is a coterie. This is again a bilateral scenario, but both mission types are not competitive.

In the rest of this section we discuss the classification of quorum agreements in terms of their performance over some metric.

For a given quorum set Q, if we can find another quorum Q_1 such that Q_1 dominates Q, then under every partition in which a group of Q is able to operate, a group in Q_1 will also be able to do so (but not vice versa). Can we extend this definition of domination to quorum agreements? That is, can we find a way of saying that quorum agreement q_1 "dominates" q? One idea would be to define domination in terms of $Q \cup Q^{-1}$ and $Q_1 \cup Q_1^{-1}$. However, mixing the members of Q and Q^{-1} does not make sense since they refer to different missions.

A second approach would be to say that q_1 dominates q if Q_1 dominates Q and Q_1^{-1} dominates Q^{-1}. This idea does not work either because of the following theorem.

Theorem 4.5 Given a quorum agreement $q = \{Q, Q^{-1}\}$ it is not possible to find another quorum agreement $q_1 = \{Q_1, Q_1^{-1}\}$ such that Q_1 dominates Q and Q_1^{-1} dominates Q^{-1}.

Proof: Assume that Q_1 dominates Q and Q_1^{-1} also dominates Q^{-1}. Recall that Q_1^{-1} is formed by the minimal transversals of Q_1 and Q^{-1} by the minimal transversals of Q. Thus, all the groups in Q_1^{-1} are also transversals of Q. Now since Q_1^{-1} dominates Q^{-1}, we can have two situations:

- There is a group in Q_1^{-1} that is smaller than any of those in Q^{-1} and still is a transversal of Q. Therefore, Q^{-1} does not contain all the minimal transversals of Q.
- All the groups of Q^{-1} are in Q_1^{-1}, but there exists a group G in Q_1^{-1} that is not in Q^{-1}. In this case, G is a minimal transversal of Q_1 and consequently of Q also. Thus, again Q^{-1} does not contain all the minimal transversals of Q. O

Theorem 4.5 shows that we can not do better than to dominate one of the quorum sets. The following theorem shows that we pay a price when one quorum set dominates the other.

Theorem 4.6 If $q = \{Q, Q^{-1}\}$ and $q_1 = \{Q_1, Q_1^{-1}\}$ are quorum agreements such that Q_1 dominates Q, then Q^{-1} dominates Q_1^{-1}.

Proof: Assume that Q_1 dominates Q. Every $G \in Q_1^{-1}$ is a minimal transversal of Q_1 and therefore a transversal of Q. Hence, G or a subset of G must be in Q^{-1}, implying that Q^{-1} dominates Q_1^{-1}. O

To illustrate these last results, consider the following quorum agreements:

$$q_1:$$
$$Q_1 = \{\{a,b,c,d\}\}$$
$$Q_1^{-1} = \{\{a\},\{b\},\{c\},\{d\}\}$$

$$q_2:$$
$$Q_2 = \{\{a,b,c\},\{a,b,d\},\{a,c,d\},\{b,c,d\}\}$$
$$Q_2^{-1} = \{\{a,b\},\{a,c\},\{a,d\},\{b,c\},\{b,d\},\{c,d\}\}$$

$$q_3:$$
$$Q_3 = \{\{a,b,c\},\{a,d\},\{b,c,d\}\}$$
$$Q_3^{-1} = \{\{a,b\},\{a,c\},\{a,d\},\{b,d\},\{c,d\}\}$$

$$q_4:$$
$$Q_4 = \{\{a,b,c\},\{a,d\},\{b,d\},\{c,d\}\}$$
$$Q_4^{-1} = \{\{a,b,c\},\{a,d\},\{b,d\},\{c,d\}\}$$

$$q_5:$$
$$Q_5 = \{\{a,d\},\{b,c\}\}$$
$$Q_5^{-1} = \{\{a,b\},\{a,c\},\{c,d\},\{b,d\}\}$$

We can establish the following order:

$$Q_4 \; dom \; Q_3 \; dom \; Q_2 \; dom \; Q_1$$
$$Q_1^{-1} \; dom \; Q_2^{-1} \; dom \; Q_3^{-1} \; dom \; Q_4^{-1}$$

Regarding Q_5, Q_5^{-1}, we can only compare it with Q_1, Q_1^{-1}, by stating:

$$Q_5 \; dom \; Q_1$$
$$Q_1^{-1} \; dom \; Q_5^{-1}$$

but this is expected since Q_1^{-1} dominates any antiquorum. In fact q_1 is the most trivial quorum agreement we can have. (For instance, in a quorum based protocol under any partition, it will never commit or it will never abort, depending on whom Q_1 represents). Notice that Q_5 does not dominate nor is it dominated by

Q_2, Q_3 or Q_4.

Notice that the selection problem for bilateral scenarios is more complicated as compared to the unilateral case. The number of choices increases, since the intersection property has been dropped. Also, deciding between two quorum agreements that dominate each other (like q_1 and q_5 for instance) implies favoring one of the two missions: aborts or commits.

One proposed metric for quorum protocols is the expected number of blocked sites, that is, the average number of sites that cannot resolve the fate of the transaction when a failure (partition) occurs. Cooper [COOP82] developed a technique to compute this number given that a *particular* partition has occurred. He uses the commit protocol description and assumptions about the communication delays and processing time to compute the probability that a site is in its window of uncertainty (it does not know whether to commit or abort a transaction) and is isolated from the groups that know how to handle the transaction. Using the components reliabilities, such a technique can be adapted to compute the expected number of blocked sites under all partitions, for a given quorum agreement.

5. BILATERAL SCENARIOS WITH ONE COMPETITIVE MISSION.

We study now of bilateral scenarios in which one of the missions must be competitive. According to Theorem 4.4, we are restricted to two possible situations:

- Both quorum sets are the same ND-Coterie, rendering a unilateral scenario.

- Only one of the quorum sets is a dominated coterie.

For both cases, quorums have two important advantages. They are a convenient mechanism for describing all bilateral mutual exclusion scenarios with one competitive mission. Furthermore, they can provide new mechanisms that are not implementable by voting and that may be more efficient in some systems. In the rest of this section we illustrate these points by considering the concurrency control problem for replicated data.

Two main techniques are widely recognized for concurrency control in distributed systems: *two-phase locking* (2PL) and *timestamping*. In 2PL, before accessing a data item, a transaction must own a *readlock* or a *writelock* on it. Two locks over the same item conflict if at least one of them is a writelock. In timestamping, transactions are given unique timestamps, which are used to resolve conflicting operations. Timestamp based algorithms are not able to cope with partitions, since they will either cause data to diverge in different groups, or will halt processing during a partition. Therefore we do not consider them here.

Depending on which locks a transaction must acquire before being able to commit, five locking techniques can be distinguished [BERN81,GIFF79]. We will describe them briefly, showing how a quorum agreement can represent each in a system where a data item is replicated at 4 nodes. We will use Q for the update quorum and Q^{-1} for the read quorum.

- *Basic Two-Phase Locking:* Here, when a transaction wants to write the data item, it must first get writelocks from all the copies. Therefore, it sends lock request to every copy of the item and waits for positive acknowledgments from all of them before committing the update. Notice that readers need only obtain a readlock from any of the copies, since this will preclude any writer from getting all the writelocks. The quorum agreement will be:

$$Q = \{ \{a,b,c,d\} \}$$
$$Q^{-1} = \{ \{a\},\{b\},\{c\},\{d\} \}$$

- *Centralized 2PL:* Before accessing the data, locks must be obtained at a central site (the same site for all items). The quorum agreement will be (assuming a is the central site):

$$Q = \{ \{a\} \} = Q^{-1}$$

- *Primary Copy 2PL:* For each data item, one copy is designated as the primary copy and all the appropriate locks must be obtained there. The quorum agreement will be (assuming a is the primary copy for the item in question):

$$Q = \{ \{a\} \} = Q^{-1}$$

- *Voting 2PL:* In this technique, each node is given a number of votes. Transactions request locks from all the copies and wait for a majority of positive answers before accessing the data item. A possible quorum agreement could be:

$$Q = \{ \{a,b,c\},\{a,d\},\{b,d\},\{c,d\} \} = Q^{-1}$$

- *Gifford's Voting:* This is a generalization of the previous cases. As discussed in Section 2, a read quorum consists of r votes and a write quorum of w votes, with $2w$ and $r + w$ being greater than the total votes, T. This is essentially the same as Skeen's mechanism, with the writes being forced to be the competitive mission type. This voting mechanism can describe the previous four (e.g., in the last quorum a, b, c have 1 vote, d has 2, and $r = w = 3$). If $r < w$, then we arrive at mechanisms not covered by the previous four cases For instance, with this same vote assignment but with $r = 2$. $w = 4$ we get

$$Q = \{ \{a,b,d\},\{a,c,d\},\{b,c,d\} \}$$
$$Q^{-1} = \{ \{d\},\{a,b\},\{a,c\},\{b,c\} \}$$

Note, incidentally, that not all choices for voting are good. Specifically, we can easily show that if $r = T + 1 - w$, then we get a good quorum agreement, else the read quorum will not be anticoterie of the write coterie. For example. with the same assignment as before, if $r = 3$, $w = 4$, $T = 5$, we get

$$Q_w = \{ \{a,b,d\},\{a,c,d\},\{b,c,d\} \}$$
$$Q_r = \{ \{a,b,c\},\{a,d\},\{b,d\},\{c,d\} \} \neq Q_w^{-1},$$

a very poor choice (e.g., we do not need to lock at $\{a,d\}$ to read, locking at $\{d\}$ is enough).

Bernstein and Goodman have decomposed the concurrency control problem into two synchronization mechanisms that can be treated separately: one for read/write synchronization and one for write/write synchronization. Both synchronizations need not be performed at the same time. If transactions delay actual changes to the database until the end (saving them in their private storage), rw synchronization can be performed as the transaction runs, saving the ww synchronization until the time the transaction is ready to commit.

In the pure locking case. they exhibit 12 different methods by combining the first four techniques explained above. Again. quorums are useful to describe these hybrid mechanisms. To illustrate, we will describe a method (suggested by Bernstein and Goodman) that uses basic 2PL for rw synchronization and voting 2PL for ww synchronization. To carry on both types of synchronization, we distinguish between rw writelocks (for rw synchronization) and ww writelocks (for ww synchronization). An rw writelock only conflicts with readlocks on the same item, while a ww writelock only conflicts with other ww writelocks over the same item. The rw synchronization is performed in the following way. To write data item x, a transaction request rw writelocks from every copy of x and waits for a positive acknowledgment from every one of them. To read the item x, the transaction must acquire a readlock from any of the copies. The ww synchronization is guaranteed by forcing transactions to collect a majority of ww writelocks before actually writing x. Using Q_{rw} and Q_{rw}^{-1} for the *rw* synchronization and Q_{ww} for the *ww* synchronization, the algorithm can be described in terms of quorums as follows:

$$Q_{rw} = \{ \{a,b,c,d\} \}$$
$$Q_{rw}^{-1} = \{ \{a\},\{b\},\{c\},\{d\} \}$$
$$Q_{ww} = \{ \{a,b,c\},\{a,d\},\{b,d\},\{c,d\} \}$$

In our terminology, this decomposition corresponds to breaking a bilateral scenario in two separate scenarios: a bilateral scenario (rw) and a unilateral one (ww). Recall that in the original bilateral scenario one of the missions (the update mission) was competitive. For the bilateral scenario that results after the decomposition this is not the case. The writes do not have to exclude each other in the rw synchronization scenario. They have a separate way of enforcing the mutual exclusion among themselves in the ww synchronization scenario. Using the example above, the rw writelocks are used to enforce the rw synchronization, while the ww writelocks do the same for the ww synchronization. However, in this example, the bilateral scenario is competitive, and this is unnecessary.

The discussion above can be summarized in the following observation:

Observation 5.1 Any bilateral scenario in which there is a competitive mission can be decomposed in two separate scenarios: a bilateral scenario without competitive missions and a unilateral scenario, provided that both scenarios can be synchronized independently.○

The hybrid locking mechanism we have presented has certain advantages in conventional database processing [BERN81], but from the point of view of partitions, it has no real advantages over, say, a basic 2PL mechanism. In both cases, readers must lock at one site and writers must lock at all four sites. The fact that the writers perform different types of locking (rw or ww) at the different sites does not matter: all four sites must be available.

However, now that we can make the rw synchronization missions of the bilateral scenario truly noncompetitive (by using quorum agreements without corresponding vote assignments), we can construct hybrid mechanisms that do have advantages under partitions and that lie outside the framework of Bernstein and Goodman.

For example, suppose that we implement rw synchronization with the quorum agreement:

$$Q_{rw} = \{ \{a,b\},\{c,d\} \}$$
$$Q_{rw}^{-1} = \{ \{a,c\},\{a,d\},\{b,c\},\{b,d\} \}$$

and ww synchronization with

$$Q_{ww} = \{ \{a,b,d\},\{a,c\},\{b,c\},\{c,d\} \}$$

As discussed in Section 4, the quorum agreement $\{Q_{rw}, Q_{rw}^{-1}\}$ may be well suited for systems like the one in Figure 4. Here, a transaction originating at node a (b) that tries to read an item must acquire the readlock at its own site plus a readlock at b (a). A transaction originating at node c (d) must do the same at node d (c). Similarly, the writes must secure the rw writelocks at two neighboring nodes. This mechanism requires minimum communication, since a node only needs to send messages to one neighbor. When a transaction completes, it ensures exclusion with other writers by locking at one of the groups in Q_{ww}. If the transaction had obtained rw locks at $\{a,b\}$, it probably will request ww locks at $\{a,b,d\}$. This is a larger group of nodes, but communication among its members is required for a shorter period of time, only while the transaction commits.

Quorums can provide interesting non-voting concurrency control mechanisms, even if synchronization is not decomposed. For example, the quorum

$$Q = \{ \{a,b,c\},\{a,d,e\},\{c,d,e\} \} \quad (writes)$$
$$Q^{-1} = \{ \{a,c\},\{a,d\},\{b,d\},\{c,d\},\{c,e\},\{e,b\},\{e,a\} \} \quad (reads)$$

cannot be implemented with voting ($v(a) + v(d) + v(e) \geq w$ and $v(a) + v(b) + v(d) < w$ imply that $v(b) < v(e)$. However, $v(a) + v(b) + v(c) \geq w$ and $v(a) + v(c) + v(e) < w$ imply that $v(b) > v(e)$.) For the system of Figure 6, this may be an excellent mechanism since read locks only have to be acquired at a single neighbor, and write locks at two neighbors.

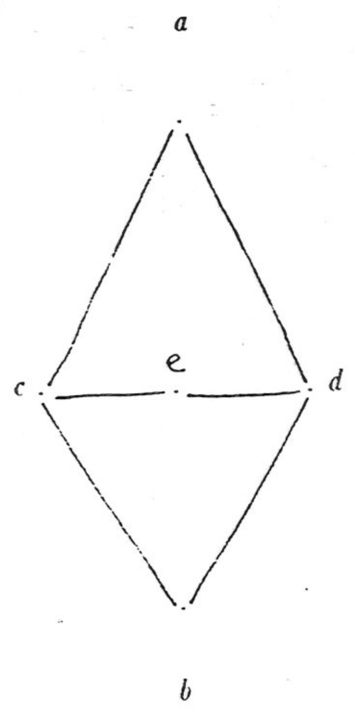

Figure 6

Incidentally, with four or less nodes, all quorum agreements with a competitive mission can be represented by voting. This example shows that it is not the case for 5 or more node systems. In the unilateral scenario, we saw in Section 3 that it requires 6 or more nodes to obtain an ND Coterie without a vote assignment. Finally, in the bilateral scenario with no competitive missions, all quorum agreements cannot be implemented with voting.

The examples above show how concurrency control mechanisms can be studied using quorum sets. In fact the use of quorum sets, provides a more general framework in which new techniques can be defined. They may also be helpful in selecting a good strategy for concurrency control. For example, Smith and Decitre [SMIT84] presented an analysis of voting algorithms using the probability that read or write quorum collection fails. Their goal was to compare several choices of r, w and the votes under this metric. Quorum sets may be used in this same way.

6. CONCLUSIONS

We have presented a framework for studying mutual exclusion scenarios in distributed systems. In section 3 we introduced the concept of coteries. Coteries proved to be a more powerful tool than vote assignments for two reasons. First, there are assignments that do not have a corresponding vote assignment. Second, they are much more structured, lending themselves to formal analysis, and helping in the selection of a good mutual exclusion mechanism.

In the next section we generalized the concept of coteries for scenarios that have more than one mission. Doing so, we discovered that quorum sets are also more powerful than exclusion mechanisms based on voting, and are useful in the selection process. For quorum based termination protocols, for instance, our results aid in the selection of the parameters V_c and V_a. We see that once V_c and the votes for each site are chosen, the quorum set has been determined. According to Theorem 4.1 the best choice for the complementary quorum set is the antiquorum. This simple result helps to trim down the choices drastically.

We presented a new class of quorum based protocols that cannot be implemented using voting mechanisms. Perhaps among those protocols we can find the optimum ones for specific situations. The following idea is worth pursuing. Assume that in a given system of 5 nodes the most probable partition is $(a,b),(c,d,e)$. Assume also that we decide to favor the aborts over the commits.

That is, we decide that the situations in which the groups are ready to abort a transaction are more probable than those in which they can commit a transaction. Then, we can use the following protocol:

$$Q_a = \{ \{a,b\},\{c,d,e\} \}$$
$$Q_c = \{ \{a,c\},\{a,d\},\{a,e\},\{b,c\},\{b,d\},\{b,e\} \}$$

That is, we select the groups in the abort quorum to be the most common groups during a partition, and the commit quorum as the corresponding antiquorum.

In Section 5 we studied the problem of bilateral scenarios in which one of the missions is competitive. We showed how existing concurrency control mechanisms for this case can be studied with quorums, and how novel ones can be developed.

7. REFERENCES

[ALSB76] P.A. Alsberg, G.G. Belford, J.D. Day, and E. Grapa, "Multi-Copy Resiliency Techniques," Center for Advanced Computation, CAC 202, University of Illinois, Urbana, May 1976.

[BARB84a] D. Barbara, and H. Garcia-Molina, "Optimizing the Reliability Provided by Voting Mechanisms," *Proc. Fourth International Conference on Distributed Computing Systems*, May 1984, pp. 340-346.

[BARB84b] D. Barbara and H. Garcia-Molina, "The Vulnerability of Voting Mechanisms," *Proc. Fourth Symposium on Reliability in Distributed Software and Database Systems*, October 1984.

[BARB85] D. Barbara and H. Garcia-Molina, "Evaluating Vote Assignments with a Probabilistic Metric," to appear in: *Proc. Fifteenth IEEE*

International Symposium on Fault-Tolerant Computing, June 1985.

[BERN77] P.A. Bernstein, D. W. Shipman, J. B. Rothnie, N. Goodman, "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The General Case)", TR CCA-77-09, Computer Corporation of America, December 1977.

[BERN78] P. A. Bernstein, J.B. Rothnie, N. Goodman, C.A. Papadimitriou, "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)", *IEEE Transactions on Software Engineering*, Vol. 4, No. 3, May 1978, pp. 154-167.

[BERN81] P.A. Bernstein, and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 185-221.

[COOP82] E. C. Cooper, "Analysis of Distributed Commit Protocols", *ACM International Conference on Management of Data*, June 1982, pp. 175-183.

[DAVI82] S. Davidson, "Evaluation of an Optimistic Protocol for Partitioned Distributed Database Systems", Technical Report 299, Department of Electrical Engineering and Computer Science, Princeton University, May 1982.

[DAVI84] S. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in a Partitioned Network: A Survey", Technical Report 320, Department of Electrical Engineering and Computer Science, Princeton University, Nov. 1984.

[DOLE82] D. Dolev, and S. Strong, "Polynomial Algorithms for Multiple Processor Agreement", *Proc. 14th ACM Symposium on Theory of Computing*, 1982, pp.401-497.

[ESWA76] K.P. Eswaran, J. Gray, R.A. Lorie, and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", *Communications ACM*, Vol. 19, 11, pp. 624-633.

[GARC85] H. Garcia-Molina, and D. Barbara, "How to Assign Votes in a Distributed System," to appear in *Journal of the ACM*. (An earlier version appeared as Technical Report 311, Department of Electrical Engineering and Computer Science, Princeton University, March 1983.)

[GIFF79] D.K. Gifford, "Weighted Voting for Replicated Data", *Proceedings Seventh Symposium on Operating System Principles*, December 1979, pp. 150-162.

[GRAY79] J. Gray, "Notes on Database Operating Systems", *Operating Systems: An Advance Course*, R. Bayer et al. editors, Springer-Verlag, 1979, pp. 393-481.

[HAMM80] M. Hammer, and D. Shipman, "Reliability Mechanisms for SDD-1: A System for Distributed Databases," *Transactions on Database Systems*, Vol. 5, No. 4, December 1980, pp. 431-466.

[LAMP78] L. Lamport, "The Implementation of Reliable Distributed Multiprocess Systems", *Computer Networks*, Vol.2, 1978, pp. 95-114.

[LAMP82] Lamport, L., Shostak, R., and Pease, M., "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and*

Systems, Vol. 4, No. 3, July 1982, pp. 382-401.

[LYNC82] N. Lynch, M. Fischer, and R. Fowler, "A Simple and Efficient Byzantine Generals Algorithm", *Proceedings Second Symposium on Reliability in Distributed Software and Database Systems*, July 1982, pp.46-52.

[PARK80] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, and C. Kline, "Detection of Mutual Inconsistency in Distributed Systems", *Proceedings of the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks*, February 1980, pp. 172-183.

[ROTH77] J.B. Rothnie and N. Goodman, "A Survey of Research and Development in Distributed Database Management", *Proceedings Third VLDB Conference*, Tokyo, 1977, pp. 48-62.

[SKEE81] D. Skeen, and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System", *Proceedings Fifth Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1981, pp. 129-142.

[SKEE82] D. Skeen, "A Quorum-Based Commit Protocol", *Proceedings Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, February 1982, pp. 69-80.

[SMIT84] W. Smith, and P. Decitre, "An Evaluation Method for Analysis of the Weighted Voting Algorithm for Maintaining Replicated Data", *Proceedings of the Fourth International Conference on Distributed Computing Systems*, May 1984, pp. 494-502.

[THOM78] R.H. Thomas, "A Majority Consensus Approach to Concurrency Control", *ACM Transactions on Database Systems*, Vol. 4, Num. 2, June 1979, pp. 180-209.

[WRIG84] D.D. Wright, "Managing Distributed Databases in Partitioned Networks", Ph.D. Thesis, Department of Computer Science, Cornell University, 1984.