POLICIES FOR DYNAMIC VOTE REASSIGNMENT

Daniel Barbara
Hector Garcia-Molina
Annemarie Spauster

September, 1985

CS-TR-010-85

POLICIES FOR DYNAMIC VOTE REASSIGNMENT

Daniel Barbara

Hector Garcia-Molina

Annemarie Spauster

Computer Science Department

Princeton University

Princeton N.J. 08540

1. INTRODUCTION

In distributed systems, voting is used commonly to provide a mutual exclusion mechanism that works under catastrophic failures like partitions. Each node is a priori assigned a number of votes, and only the group with a majority of votes is allowed to perform a restricted operation [e.g., DAVI82, GIFF79, THOM79, GARC82].

A classical problem in which voting has been widely used is the one of controlling access to replicated data items in distributed databases, where executing transactions can read and write these items. Basically, before committing, a transaction must have control over all 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

This work has been supported by NSF Grants ECS-8303146 and ECS-8351616, Naval Electronic Systems Command contract N00039-84-C0485, and from grants from IBM and NCR corporations.

partitions occur, a transaction must also exclude transactions in other groups from executing updates that conflict with its updates. For example, consider the four node system of Figure 1. Assume each node has a copy of a database and say the following vote assignment is in effect: $v_a = v_b = v_c = 1$ and $v_d = 2$. In this case, if nodes a and d can communicate, then they can run update transactions on the database. Since a and d together have a majority of votes (3), then no other group could possibly be in the same situation.

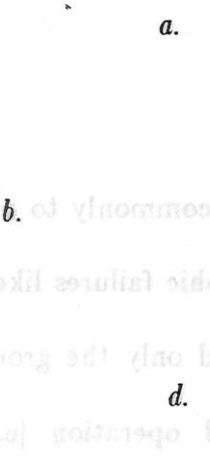


Figure 1

Since groups cannot communicate with each other during a partition, it is possible that at a given time no group has a majority of votes. (In this example, this can happen if, say, the three groups $\{d\}$, $\{a\}$, $\{b,c\}$ form.) There is no way to avoid this problem: even giving one node all the votes does not help since that node may fail. We say that under these circumstances, the system becomes

halted. The likelihood of such a state can be reduced by selecting a good vote assignment [BARB85a].

The likelihood of halting can also be reduced if we take advantage of the following idea: in the case of a partition that leaves some group with the majority of votes, let this group reassign the votes dynamically in order to increase its voting power so further partitions may be survived. Such dynamic vote reassignment is the topic of this paper.

To illustrate, consider once again the system of Figure 1 and vote assignment $v_a = v_b = v_c = 1$ and $v_d = 2$. Assume that a partition separates node d from nodes a, b and c. Nodes a, b and c can still collect a majority of votes (3), while d cannot. However, if a second partition occurs, separating node c from a and b, the system will be halted.

During the first partition, nodes a, b and c may opt for reconfiguring the votes in an effort to become more resilient to new partitions. For instance, a new vote assignment could be $v_a = v_b = v_c = 5$ and $v_d = 2$. In this way the second partition will find nodes a and b with 10 votes over 17, forming a majority group and the system will not be halted.

Of course, we must be very careful in switching from the first assignment to the second one, making sure that no node "mixes up" new and old votes. This will ensure that all nodes are eventually informed of the new assignment, and guarantee that at no time will two isolated groups think they have a majority. We will refer to the algorithms for making the vote transition as the *protocols.*

Just as important, we must select a good new assignment. Note that the second assignment in the example is just one of many possibilities, and each different assignment may give different protection against new partitions. We will use the term *policy* to refer to the mechanism for selecting the new assignment. In this paper we study various policies and compare the protection they give.

It is possible to distinguish two strategies for dynamic vote assignment:

- **Group consensus.** The nodes in the active group agree upon the new vote configuration in a coordinated fashion. The information used is global. Nodes outside the active group will not receive any votes under the new configuration.
- **Autonomous reconfiguration.** Each node makes its own decisions autonomously, without subordinating its new choice of votes to any other node. However, before a change is made final, a node must get the approval of the majority.

Group consensus can be achieved by using a coordinator. After a failure, the members of the majority group (if any) elect a coordinator. The coordinator collects information about the current system topology, and with a given policy determines a good assignment. The coordinator then distributes the new assignments to all group members. The new votes are tagged with a ***version number*** that distinguishes them from old votes. Nodes outside the majority group cannot be informed of their new votes (typically 0), but since their version number is out of date, their votes become obsolete automatically. The protocols for group con-

sensus are described in more detail in [GARC82], and the policies for selecting an assignment given a topology are studied in [BARB85a]. (Incidentally, group consensus can also be performed in a distributed fashion.)

Group consensus can select very good assignments, but unfortunately requires fairly tight coordination among the members of the group and good knowledge of the current system topology. On the other hand, autonomous reconfiguration is much simpler and more flexible. Each node decides independently how many votes it should have. The node does not need complete or accurate information about the state of the system. In a sense, the node makes educated guesses about the best number of votes to have, with its primary goal being to claim for itself all or part of the voting power of a node (or nodes) that have been separated from the majority group.

When a node wishes to change its votes, it follows a commit protocol to inform the rest of the nodes. The node can start voting with its new votes as soon as it receives acknowledgments of the change from nodes with a majority of votes, as this ensures that the change is eventually propagated to all nodes. The protocol is extremely simple (only one phase is required), although the proofs of correctness are not. The details can be found in [BARB85b].

The goal of this paper is to study the policies for autonomous vote reassignment and to answer the following key questions: Which dynamic policy is best? What price do we pay for having the simpler autonomous strategy? That is, does group consensus provide significantly higher protection against partitions? The policies are studied in section 2; the comparisons are made in section 3. Section 4

presents some conclusions.

2. VOTE CHANGING POLICY

In this section we consider several variations on autonomous reconfiguration. We can divide the strategies as follows:

- **Alliance techniques.** After a failure (or group of failures), for each node outside the active group, *all* the nodes in the active group increase their votes to supplant the failed node.
- **Overthrow techniques.** After a failure (or group of failures), for each node outside the active group, there will be *one* node in the active group supplanting it. A protocol is needed to decide which node will undertake this task. An a priori ordering of the nodes may suffice.

We will describe policies for these two main techniques.

2.1 The Overthrow Technique

Vote increasing under the overthrow technique is straightforward. Consider a system in which node x has gone down, while the rest of the nodes are still up. (This can be considered as a partition of the system into two groups, with x in one group and the rest of the nodes in the other.) Let v_x be the number of votes that node x has. Let TOT be the total number of votes in the system and MAJ the majority of votes. Assuming TOT is odd, $MAJ = \frac{TOT+1}{2}$. If node a is the node supplanting x, the new number of votes for a, v_a' will have to be such that it covers the voting power that a had before (v_a), plus the voting power of x, plus the increase in the total number of votes. If a increases its votes by $2v_x$, the total

number of votes will be $TOT' = TOT + 2v_x$ and $MAJ' = MAJ + v_x$. It can be shown that all the majority groups that used x can be formed using a instead:

- If a group G had V_G votes and contained a and x, the group $G' = G - \{x\}$, will have $V_G' = V_G + 2v_x - v_x \geq MAJ + v_x = MAJ'$. Therefore, G' is a majority group under the new vote assignment.
- If a group G had V_G votes and contained x, but not a, the group $G' = G \cup \{a\} - \{x\}$ will have $V_G' = V_G + v_a + v_x > MAJ'$.

Of course, any other group in which x does not participate may suffer loss of voting power and need the help of node a to complete the majority, but the basic goal of supplanting x is achieved.

Deciding which node should increase its votes for the node(s) that are no longer in the active group can be accomplished, for example, by ranking the nodes; the node in the majority group with the lowest ranking can take the extra votes. Or, a token passing mechanism can be used, where the node with the token takes the votes. In any case, the method does not need to be foolproof. If problems such as communication delays arise and nodes pick a vote value that is not accurate or several nodes supplant an excluded node, the worst that will happen is that the new assignment is not as good as it could be. It will not lead to more than one active group.

2.2 The Alliance Technique

There are many variations on the alliance technique. We describe three here. In general, we want to give each node a fraction of the voting power of a node that has been excluded from the majority group. As in the overthrow

technique, we want to be sure to give out at least $2v_x$ votes in the majority group, enough to counteract those votes that node x holds plus the number of votes node x could have contributed if it were in the active group. Of course, we can always assign a surplus of votes to each node. One possibility is to assign $2v_x$ votes to every member of the active group; or, we can assign v_x votes to each member of the active group, and assign $2v_x$ votes when there is just one node left. Another possibility is to spread the $2v_x$ votes out. Say N = the number of nodes in the majority group. Then, give each node in the active group $\left\lceil \frac{2v_x}{N} \right\rceil$ votes. If need be, N can be estimated by the nodes. This may not be as good as possible in terms of resilience to failures, but is certainly not dangerous. No matter what the strategy, we have to be careful of the case when there are only 2 nodes left in the majority group. In that situation, it is senseless to give each node the same number of votes, since if they lose communication with each other, their extra votes will only cancel each other out and no group may have a majority. Instead, it is better to pick one node and give it $2v_x$ votes.

2.3 Examples

We illustrate these techniques with an example. Consider again the system of Figure 1, but with initial vote assignment $v_a = 6$, $v_b = v_c = v_d = 5$. Assume that node a gets disconnected from the rest, leaving $\{b,c,d\}$ as the active group. Using the overthrow techniques, if we say node b is of lowest rank or has the token (depending on the strategy used), the new vote assignment will be

$$v_a = 6,\ v_b = 17,\ v_c = v_d = 5.$$

since we give node b $2v_a$ votes.

Now consider the three alliance techniques. If we give each node $2v_a$ votes we have

$$v_a = 6,\ v_b = v_c = v_d = 17.$$

If we give each node v_a votes we have

$$v_a = 6,\ v_b = v_c = v_d = 11.$$

If we give each node $\left\lceil \frac{2v_a}{N} \right\rceil$ votes, N = the number of nodes in the majority group, we have the assignment

$$v_a = 6,\ v_b = v_c = v_d = 9.$$

There are obvious differences between these assignments. For instance, the first two have node(s) with a higher number of votes than the last two. In general, the size of the votes will grow much faster using the first two techniques than in the last two. We will discuss how to handle this problem shortly. Also, the first assignment gave much power to one node, which can be a disadvantage if this node fails. Also important is the amount of message traffic these assignments incurred. Since only one node got votes in the overthrow case, this technique required fewer messages. But just looking at the assignments tells us little about their relative performance. This is just one topology and set of failures, certainly not representative. A later presentation of simulation results will give us a better means of comparison.

2.4 Balance of Power

Of course, whenever a node is excluded from the majority and other nodes increase their votes, the balance of voting power is disturbed. What we need are

techniques for maintaining the equilibrium of the system. There are two possibilities:

(1) A node that has been out of the active group can "catch up" when it returns to the active group, in other words, increase its votes.

(2) When a node that has been out of the active group returns, the node(s) that increased their votes because of its absence can relinquish them, i.e., decrease their votes.

2.4.1 Catch up Strategies

When a node x is separated from the active group each node in the active group takes on more votes, depending on the policy chosen for vote increasing and the value v_x. When node x returns to the active group it should increase its votes by at least as many as other nodes did when x was initially excluded. In addition, node x can keep track of the last majority group that it participated in. When it becomes part of an active group again, it can determine which nodes were in the previous majority group with x, but are not in the present majority group. These nodes caused vote increases since the last active group that included node x. Node x can then increase its votes as if it were present when those nodes were initially excluded. This allows node x to pick up most (but maybe not all) of the vote changes that occurred while it was excluded from the active group. It may not, for example, increase its votes for a node, y, that has failed after x failed, but repaired before x repaired.

We illustrate this strategy with an example. Referring again to Figure 1, suppose we have the initial vote assignment $v_a = 6$, $v_b = v_c = v_d = 5$. Assume

we are using the $2v_x$ alliance technique with catch up. Say once again that node a goes down and nodes b, c, d get 17 votes each as in the example of Section 2.3. Then, say node b fails and nodes c and d take on 34 more votes each yielding the assignment

$$v_a = 6,\ v_b = 17,\ v_c = v_d = 51.$$

Now, node a returns and wishes to catch up. It takes 2 times its own votes automatically. Then, it checks to see who is in the active group and notes that node b is not. Since node b was in the last active group a was in, b must have become excluded in the interim. So, node a takes $2 \times v_b$ votes as well, which it has learned from node c (or d) is 17. So, node a gains 46 votes for a total of 52. The final assignment is

$$v_a = 52,\ v_b = 17,\ v_c = v_d = 51.$$

and node a has caught up.

Of course, we can not let votes increase forever. Eventually the nodes must decide to decrease their votes. After some node hits a predefined threshold vote value it can attempt to initiate some consensus technique to bring all the votes back down to their original values. This may require that all nodes be in the active group and that other processing wait until the votes are reset. This is probably not too severe, though, since this reconfiguration should not need to be done too often.

2.4.2 Decrease Strategies

Decreasing votes avoids the problem of votes getting too big. To implement this, each node must keep track of how many votes it took on when some other

node became excluded. When the node returns to the active group, the nodes can decrease their votes by the appropriate amount. This requires that each site maintain a table with an entry for each other node indicating the vote change.

In the next section we present performance results for the techniques described along with an analysis of the advantages and disadvantages of each method.

3. PERFORMANCE RESULTS

As discovered in the last section we have many policies to choose from for an implementation of dynamic vote reassignment. The next step is to determine their usefulness. Several questions require our attention: Is dynamic vote reassignment much better than retaining a static assignment? Is group consensus much more resilient than the autonomous techniques? Is one autonomous technique better than the rest? Is the topology of the network relevant? In an attempt to answer these questions we simulated the policies of the previous section.

3.1 Methods

Each experiment uses an event driven simulator where the events consist of node and link failures, and repairs. Times between failures (and repairs) are exponentially distributed. The failure rates are assumed to be very high in order to focus our attention on how well the system adapts. In other words, if we choose typical failure rates (e.g., each component is down 1% of the time), the system will halt rarely and the vote reassignment policy will have little effect on average reliability measures. However, our goal is to minimize disruptions *during*

failures, so instead we zero in on a failure period when the system is unstable, by selecting high failure rates.

To simplify the simulation, we assume that the vote reassignment is done instantaneously. In other words, if an event causes a new vote assignment, the vote changes occur before the next event does. This approach will yield worst case results for autonomous reconfiguration techniques as compared to group consensus since we presume that group consensus vote increases take much longer in practice than autonomous techniques.

The topology and the connectivity of the communication network play an important role in dynamic vote reassignment. Of course, we must limit the number of networks studied, and Figure 2 presents the six 5-node systems for which we give results in this paper. The networks represent a spectrum of connectivity. We chose 5-node networks because we think they are the most representative. Larger networks are not expected: remember, we are only looking at nodes containing database copies. Finally, the results for smaller systems appear to be subsumed by the results for these networks.

All results obtained a 95% confidence interval with a width of 10 percentage points. Each policy was simulated for 200 time units for the five network topologies shown in Figure 2, with nodes and links failing and repairing at a rate of 20. Nine strategies were simulated:

(1). Alliance: $2v_x$ votes with vote catchup

(2). Alliance: v_x votes with vote catchup

(3). Alliance: $\left\lceil \frac{2v_x}{N} \right\rceil$ votes with vote decreasing

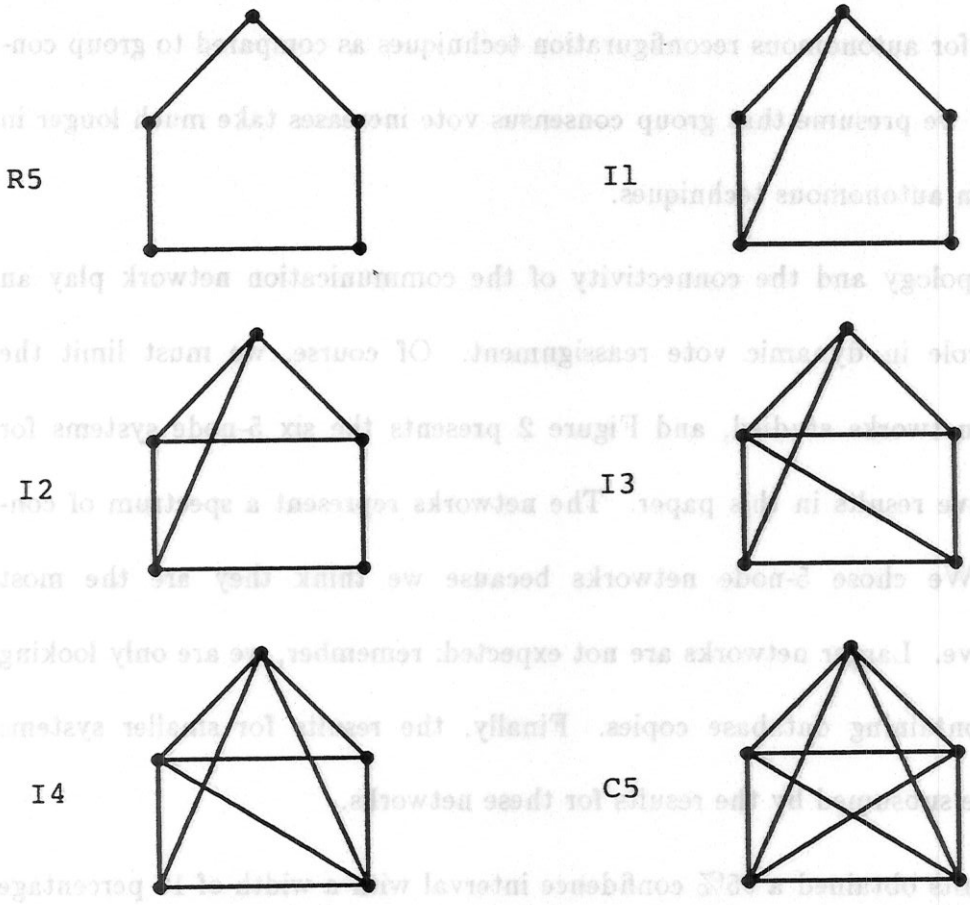


Figure 2

(4). Alliance: $2v_x$ votes with vote decreasing
(5). Alliance: v_x votes with vote decreasing
(6). Overthrow: ranking with vote decreasing
(7). Overthrow: token passing with vote decreasing
(8). Group consensus
(9). Static Vote Assignment

The overthrow technique combined with vote catch up was not implemented. Since in overthrow votes are not evenly distributed when a node is excluded, a node catching up would only catch up to one other node and do little to rebalance the voting power. Vote catch up was not implemented for the alliance technique with $\left\lceil \frac{2v_x}{N} \right\rceil$ votes either, since it would be difficult in practice for the returning node to know how many nodes were in the active group when it was excluded.

For each strategy, the nodes start off with an initial assignment determined by the topology of the network. Votes are distributed according to the number of links incident to a node. If need be, an extra vote is added to the node with the largest number of incident edges to make the total number of votes odd. See [BARB85a] for the details and rationale of this technique. The static vote assignment strategy retains this same assignment throughout a run. The group consensus technique readjusts the votes after each failure according to this algorithm.

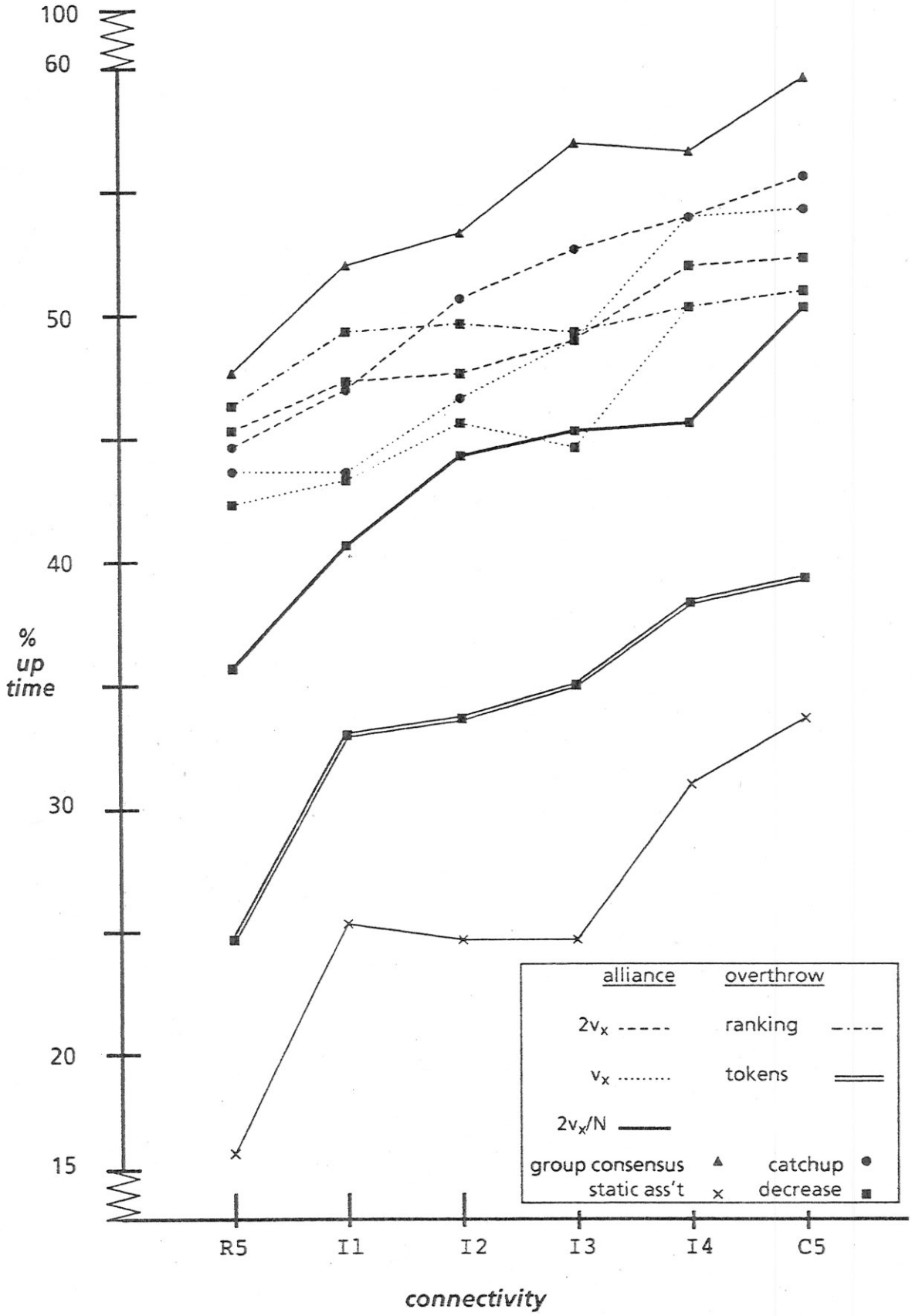
The strategies are compared on the basis of mean percent uptime, the average over all runs of the percentage of time that some group in the system could perform operations. For purposes of illustration, this is displayed graphically in

Graph 1, even though the connectivity axis is not continuous. (Note that up times are relatively low, but remember that we are looking at a failure period only.) Incidentally, we also performed a number of other tests. Due to space limitations, they are not presented here, but the results of Graph 1 are representative.

3.2. Analysis

Looking at Graph 1, we see that any of the reconfiguration strategies proved more effective at keeping the system operative than retaining a static assignment. Group consensus showed a slight gain over the autonomous techniques in all cases. Among the alliance techniques, assigning $2v_x$ votes with catching up yielded the best results, especially in the intermediate cases, although $2v_x$ with vote decreasing worked as well for graphs R5 and I1. In general, though, catching up yielded better results than its decreasing counterpart for higher connectivity graphs. Alliance $2v_x$ performed better than v_x under the same balancing strategy. Use of the alliance policy with $\frac{2v_x}{N}$ votes showed relative improvement as the connectivity increased.

Overthrow using token passing did not work as well as the other reconfiguration strategies. This is not surprising since when the token ends up with some node not in the majority, no node gets extra votes. Overthrow using ranking, however, performed well. It was in fact slightly better than the alliance techniques for graphs R5 and I1, but worse for the I2, I3, I4 and C5 graphs. This is not surprising in light of work done in [BARB85a] on static assignment, which asserts that vote distribution is not as advisable for rings and other low


100
60
50
40
30
20
15
% up time
R5
I1
I2
I3
I4
C5
connectivity
alliance
overthrow
2v_x
ranking
v_x
tokens
2v_x/N
group consensus
catchup
static ass't
decrease

Graph 1

connectivity networks; a singleton assignment is preferable. But higher connectivity graphs perform better under vote distribution. Otherwise, $2v_x$ with decreasing appears to be a good choice.

4. CONCLUSIONS

Judging from the results, dynamic vote reassignment is a valuable technique for any system with frequent enough failures to render it inoperative. Specifically, note that dynamic vote reassignment can improve up time over a static assignment by a *factor* of 2 or 3. The results at first seem to point to group consensus as the best choice overall. Keep in mind, though, that the nature of the simulation favors group consensus over autonomous techniques. In general, group consensus will take longer and is harder to implement. In light of this and how closely these techniques did perform in the simulation, autonomous techniques may be a viable alternative.

Of the autonomous techniques, overthrow, in consideration of its simplicity and low message traffic is advisable, especially in low connectivity networks. For higher connectivity networks, especially the completely connected case, the win in performance of the alliance technique may be enough to deal with the message traffic. Catching up is better, but reconfiguration must be easy to handle to make it worthwhile. Otherwise, $2v_x$ with decreasing appears to be a good choice.

In summary, vote assignments play a central role in making data available during partitions. Giving a system the capability to dynamically adjust these votes is relatively simple with autonomous techniques and can significantly improve reliability. The results we have presented suggest good policies for

implementing such reconfigurations.

References

[BARB85a] D. Barbara, "Mutual Exclusion in Distributed Systems", Ph.D Thesis, Department of Electrical Engineering and Computer Science, Princeton University, October 1985.

[BARB85b] D. Barbara, H. Garcia-Molina, and A. Spauster,"Dynamic Vote Assignment: Protocols and Policies", Technical Report, Department of Computer Science, Princeton University, September 1985.

[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.

[GARC82] H. Garcia-Molina, "Reliability Issues for Fully Replicated Distributed Databases", *IEEE Computer*, Vol. 15, No. 9, September 1982, pp. 34-42.

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

[THOM79] R.H. Thomas, "A Majority Consensus Approach to Concurrency Control", *ACM Transactions on Database Systems*, Vol. 4, No. 2, June 1979, pp. 180-209.