ENHANCEMENTS TO THE VOTING ALGORITHM

Sushil Jajodia and David Mutchler

Computer Science and Systems Branch
Code 5590
Naval Research Laboratory
Washington, DC 20375-5000

ABSTRACT

There are several consistency control algorithms for managing replicated files in the face of network partitioning due to site or communication link failures. In this paper, we consider the popular ***voting*** scheme along with three enhancements: ***voting with a primary site***, ***dynamic voting***, and ***dynamic voting with linearly ordered copies***. We develop a stochastic model which compares the file availabilities afforded by each of these schemes. We show that in this model dynamic voting with linearly ordered copies provides the greatest availability.

I. INTRODUCTION

There are several consistency control algorithms for managing replicated data in the face of network partitioning due to site or communication link failures [4]. *Voting*[5,12,15] is the best known example of such a scheme. It has several appealing aspects: its availability is reasonable; its simple statement permits a clear correctness proof; and it is simple to implement. *Voting with a primary site* is a simple extension of voting. More recently, researchers have introduced two other enhancements to voting, called *dynamic voting* [6] (see also [3]) and *dynamic voting with linearly ordered copies* [7]. These enhancements share all the advantages of the voting scheme; we show that they provide greater availability as well.

Sections II and III give formal statements of the problem and the four algorithms listed above. Section IV provides a stochastic analysis of the availabilities of these algorithms. The model we use assumes that update requests arrive much more frequently than sites fail or are repaired. In the context of our model, we state theorems that compare the availabilities of the four algorithms. Our main result is that dynamic voting with linearly ordered copies provides the greatest availability.

II. FORMAL SPECIFICATION OF PROBLEM

The distributed database (DDB) system consists of a collection of independent computers, called *nodes* or *sites*, connected via communication links. We assume that site failures are clean, i.e., nodes stop executing without performing any incorrect actions and that node crashes are detectable by other nodes. We do not include Byzantine failures [11] where sites may act in an arbitrary and malicious manner. Site or communication failures may separate the sites into more than one connected component of communicating sites. We call each connected component a *partition*.

There are several logical files in the DDB, and a physical copy of each logical file is stored at one or more sites. Each site keeps a history of all updates which it performed on a file. We assume that each site runs a *concurrency control protocol* which ensures that the execution of all transactions within any partition is serializable [8,1]. While serializability of transactions at each site is certainly desirable, it is not sufficient to guarantee that the transactions running in different sites will combine to yield a serializable result; and therefore, it is necessary to run a *consistency control protocol* which correctly manages the replicated data in the presence of failures. (An excellent survey of several of these strategies is given in [4].) In a *pessimistic* consistency control protocol, mutual consistency of a replicated file is maintained by making sure that all reads are fresh and that files are updated in at most one partition at any given time. We will call such a partition the *majority* partition. Different pessimistic protocols use different definitions of a majority partition. When site or communication link recoveries cause partitions to unite, the nodes form a new partition by comparing their histories and obtain, if necessary, all updates that they have missed. If there does not exist a majority partition, all sites in the system must wait until enough sites and communication links are repaired so that there is once again a majority partition in the system. Since this wait is unavoidable [14], the challenge is to come up with a pessimistic consistency control algorithm which not only preserves mutual consistency of various copies of a file, but at the same time achieves high availability as well.

We can pictorially represent the history of network's failure and recovery by using the notion of a partition graph [10], defined as follows.

Definition 1. A *partition graph* for a file *f* is a directed acyclic graph such that the nodes correspond to the partitions and the edges correspond to either a fragmentation of a partition into two or more subpartitions or a coalescence of two or more partitions into a single partition.

Example 1. An example of a partition graph is shown in Figure 1. The source node is labeled with the names of sites ABCDE that have copies of the file *f*, indicating that these sites are all connected and that copies of *f* are mutually consistent. The initial partition is fragmented into two partitions ABC and DE. Later B becomes isolated from AC, and subsequently A and C also become isolated. Ultimately, A, D, and E resume communication and form a single partition.

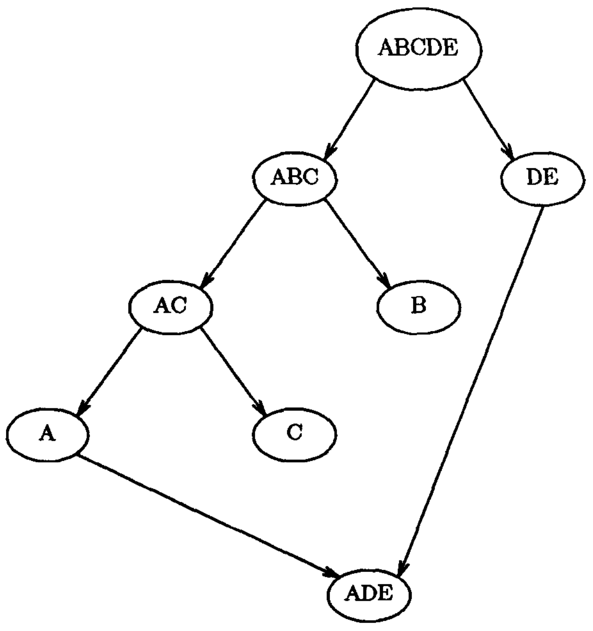

Figure 1. A partition graph for a file stored redundantly at sites A, B, C, D, and E.

The partition graph in Figure 1 illustrates the different availabilities of the algorithms we describe in the next three sections. Voting permits only three partitions—ABCDE, ABC, and ADE—to process new updates to the replicated file. Dynamic voting allows updates in partitions ABCDE, ABC, and AC. Dynamic voting with linearly ordered copies permits updates in one partition at each level of the partition graph, viz., in partitions ABCDE, ABC, AC, A, and ADE.

III. VOTING AND ITS ENHANCEMENTS

A. THE VOTING ALGORITHM

We present a brief overview of the voting algorithm as follows. Each copy of f has associated with it a *version number* which is equal to zero initially and is incremented by one each time the copy is updated. A site can process an update request provided it can communicate with at least $\lfloor n/2 \rfloor$ other sites, where n is the number of sites. If so, it must first ensure that those participating copies which are not current are brought up-to-date. To this end, it computes the maximum, say M, taken over version numbers of all participating copies. The copy which has the version number M is the current copy and is used to propagate missing updates to the other copies. Once this is done, the new update is then performed.

In the voting protocol, more than half of the total number of copies of a replicated file must be available in order for an update to succeed. In the next three sections, we present three generalizations of the voting scheme; each sometimes permits a partition to perform updates even when the partition does not contain more than half of the sites.

B. VOTING WITH A PRIMARY SITE

A partition containing exactly half the sites does not constitute a majority partition under the voting algorithm; if it did, two majority partitions could coexist, possibly destroying the consistency of the replicated file. By choosing one site as the primary site, two such partitions can be distinguished. The voting with a primary site algorithm is the same as voting, except when a partition contains exactly half of the sites. Under voting with a primary site, such a partition constitutes a majority partition if it contains the primary site. Since only one such partition can exist at any particular time, consistency is maintained. Of course, voting with a primary site applies (differs from ordinary voting) only when the number of sites is even.

C. DYNAMIC VOTING

In this section, we provide a brief description of the dynamic voting scheme as given in [6] (see also [3]). There are many details missing, and the reader is referred to [6] for a complete description.

We associate with each copy of the file f a version number and an update sites cardinality, defined as follows.

Definition 1. The *version number* of a copy f_i at a site i is an integer VN_i which counts the number of successful updates to f_i. We set VN_i to zero initially and increment it by one each time an update to f_i occurs.

Definition 2. The *current version number* of a replicated file f is the maximum taken over the version numbers of all copies of f.

Definition 3. A copy is said to be *current* if its version number equals the current version number of the replicated file.

Definition 4. A partition is said to be a *majority partition* if it contains more than half of the current copies of the replicated file f.

Definition 5. Associated with each copy f_i at a site i is another integer called the *update sites cardinality*, denoted by SC_i, which always reflects the number of sites participating in the most recent update to f_i. We let $SC_i = n$ (number of sites) initially, and whenever an update is made to f_i, then SC_i is set to the total number of copies which were updated during this update.

Each site which has a copy of the file f must maintain the copy's version number and update sites cardinality. A site can perform an update iff it is a member of a majority partition. This scheme can be best illustrated by an example.

Example 2. Assume there are five sites A, B, C, D, and E which have copies of the file f. These sites are initially connected and form a single partition. Suppose the file f has been updated nine times, so the initial state can be represented as follows:

	A	B	C	D	E
VN:	9	9	9	9	9
SC:	5	5	5	5	5

At this point suppose site A receives an update, and it finds that it can communicate with sites B and C only. Since A still belongs to a majority partition, it can process the update. The state then changes to:

	A	B	C	D	E
VN:	10	10	10	9	9
SC:	3	3	3	5	5

Suppose now that site A receives yet another update, and it discovers that it can communicate with site C only. The novelty here is that since sites A and C together contain more than half of the current copies of the replicated file, they form a majority partition even though there are only two sites (out of five) in this partition. The database state changes to:

	A	C	B	D	E
VN:	11	11	10	9	9
SC:	2	2	3	5	5

D. DYNAMIC VOTING WITH LINEARLY ORDERED COPIES

Although dynamic voting provides greater availability than the voting algorithm, it is possible to make further improvements. Below we propose a modification to dynamic voting which results in improved availability of replicated files. We should note that this modification was offered originally by Jajodia in [7]. In addition to correcting some minor errors which were present therein, our description below simplifies his algorithm in a significant way making it more practical. That is, the scheme in [7] implicitly assumed the existence of a *connection vector* which instantaneously recorded changes in the system configuration resulting from site failures or network partitions (see also [3], page 89). The cost of maintaining such a vector is significant [2, section 4.2], making schemes which rely on it less desirable. By contrast, our proposal does not require any such complicated message-based coordination mechanism.

Throughout this section, we assume that there is a file f which is stored redundantly at n sites in the distributed system. Initially, these sites are all connected and all copies are mutually consistent. We assign a priori a linear ordering, denoted by $>$, to all sites that have copies of the file f. Our algorithm uses this linear order to "break ties" between the partitions. We have chosen the terminology "linearly ordered copies" instead of "linearly ordered sites" since the linear ordering of sites applies only to the replicated file. The database may contain additional files which are stored redundantly at different sets of sites, and a different linear ordering of sites in each set may be selected for each replicated file. Since our protocol does not depend on the number of files which are replicated or on whether a different ordering of sites is chosen for each file,† we shall continue to assume for ease of exposition that f is the only replicated file.

We will increase availability in our algorithm by having more majority partitions than are possible in dynamic voting. We will do so by altering the definition of a majority partition in Definition 4 as follows:

Definition 4′. A partition P is said to be a *majority partition* if either of the following two conditions holds:

a) The partition P contains more than half of the current copies of the replicated file f.

b) The partition P contains exactly one half of the current copies of the file f and moreover, contains a site S such that i) the physical copy of S is current and ii) $S > S'$ where S' is any other site containing a current copy of f.

The following theorem, whose proof is immediate, provides the basis for the correctness of dynamic voting with linearly ordered copies.

Theorem 1. There can be at most one partition at any given time that satisfies either a) or b) in Definition 4′.

In dynamic voting (without a linear ordering of sites), each site uses a single integer (the update sites cardinality) to count the number of current copies. In order to make use of the linear ordering of sites, dynamic voting with linearly ordered copies requires that each site maintain, in addition to a version number and a update sites cardinality for its copy, a variable called distinguished site, defined as follows.

Definition 6. We associate with each copy f_i at a site S_i a variable called ***distinguished site***, denoted by DS_i, which identifies the site which is greater (in the linear ordering) than all other sites that participated in the last update to f_i. Thus, the distinguished site entry DS_i for the file copy f_i at the site S_i is determined as follows: if $S_1 \ldots S_m$ denote the sites that participated in the most recent update to copy f_i, and if $S_j > S_k$ for all k $(1 \leq k \leq m,\ k \neq j)$ then $DS_i = S_j$.

As before a site can make an update iff it belongs to a majority partition. Suppose a site S wishes to determine if it belongs to a majority partition. It must execute the following steps. Let P below denote the sites in the partition containing S.

Is-Majority:

1) The site S obtains the version numbers VN_i, update sites cardinalities SC_i, and distinguished site values DS_i for all sites S_i in partition P.

2) The site S calculates the value $M = \max\{VN_i : S_i \in P\}$ and the set $I = \{S_j \in P : VN_j = M\}$. Thus, M denotes the largest version number which is in P, and the set I consists of those sites in P which have the version number M. S then takes the update sites cardinality of *any* site in the set I. Denote this by N.

3) If card(I)†† $> N/2$, then S belongs to a majority partition.

4) If card$(I) = N/2$ and if $DS_i \in I$ where DS_i is the distinguished site value of any site in the set I, then also is S a member of a majority partition.

5) Otherwise, S does not belong to a majority partition.

Now, suppose a site S receives an update request. It must first determine if it belongs to a majority partition; if so, it can process the update; otherwise it must reject the update. Specifically, S executes these steps:

† Our work generalizes to the setting where transactions may update two or more files. Any such transaction T will require a majority for *every* file in its read and write set.

†† Notation: For a set X, card(X) denotes its cardinalitv

Do-Update:

I) S first determines if it belongs to a majority partition, by using the **Is-Majority** procedure.

II) If so, it can proceed with the update. Any update must be made to all sites in the set I. The version numbers and the update sites cardinalities at every site in I must be modified as follows: Every site in I has the new version number $M + 1$ and the new update sites cardinality equals card(I). Moreover, if card(I) is even, then the value of DS_i at every site in I is reset also: the new distinguished site value is set to S' where S' in I is such that $S' > S''$ for every S'' in I, $S'' \neq S'$. We should note then whenever an update succeeds at a group of sites, they all must commit the update together with the new version number, update sites cardinality, and distinguished site entry. (Thus, an update operation at a site is atomic in that either all four operations are performed in entirety or they are not performed at all.)

Example 3. To show how this algorithm results in greater availability, we consider once again the scenario of the previous example. We now assume that all sites are linearly ordered as follows: A > B > C > D > E. When the file f has been updated nine times, the database state will be represented as follows where the symbol '-' means that we do not care about the actual value of this variable:

	A	B	C	D	E
VN:	9	9	9	9	9
SC:	5	5	5	5	5
DS:	-	-	-	-	-

At this point, the site A receives an update and finds that it can communicate with sites B and C only. The update is processed leading to:

	A	B	C	D	E
VN:	10	10	10	9	9
SC:	3	3	3	5	5
DS:	-	-	-	-	-

Next, the site A receives yet another update. This time site A finds (in our scenario) that it can communicate only with site C. Since sites A and C together form a majority partition, they perform the update. Moreover, since there are even number of sites in this partition, the value of DS_i is made equal to A. Thus, the database state changes to:

	A	C	B	D	E
VN:	11	11	10	9	9
SC:	2	2	3	5	5
DS:	A	A	-	-	-

Suppose sites A and C perform four additional update operations and subsequently become isolated from each other. The system state changes to the following.

	A	C	B	D	E
VN:	15	15	10	9	9
SC:	2	2	3	5	5
DS:	A	A	-	-	-

The novelty of our approach is that at this point the partition consisting of the single site A is a majority partition. Suppose that A updates f twice; later, D and E unite with A and bring their copies up-to-date. (Exactly how this is done will be described shortly.) If, at this point, A were to become isolated from sites D and E, partition DE will be a majority partition and will be able to perform new updates to f. And then, were C to unite with partition DE and obtain missing updates, the partition CDE will also be able to perform new updates to f. Thus, C will be able to process f in spite of the fact that it was never united with the site A. By contrast, under dynamic voting, once partition AC splits into two partitions A and C, no updates can occur anywhere in the system unless sites A and C once again coalesce into a single partition.

In our scheme, updates are always made to a current copy; those copies which are not current must be made so before they can be updated. For this purpose, sites from time to time determine if their copies are current; if not, they must take steps to obtain missing updates for their copies. Specifically, a site S executes the following steps:

Make-Current:

a) S determines if it belongs to a majority partition. To this end, S computes the value M and the set I as in step 2) of the **Is-Majority** procedure.

b) If the site S belongs to a majority partition and the version number of the copy at S is equal to M, then the copy at S is current.

c) If the site S belongs to a majority partition and its version number is less than M, it may obtain the missing updates from any site in the set I. The values VN_i and SC_i of the copies at site S and at sites in I are changed as follows: each VN_i is set to $M + 1$, and each SC_i equals card(I) + 1. Moreover, if card(I) + 1 is even, then the value of DS_i at every site in $I \cup \{S\}$ is modified as well: DS_i is set to S' where S' in $I \cup \{S\}$ is such that $S' > S''$ for every S'' in $I \cup \{S\}$, $S'' \neq S'$. Note that we again view these operations as atomic: either they are performed in their entirety or not at all.

d) Otherwise, S cannot obtain the missing updates.

Theorem 2. Our consistency control protocol is correct.

Proof. (sketch) We shall show that there exists by our algorithm at most one majority partition at any time. Thus, the replicated file can be updated in at most one partition which is enough to guarantee the mutual consistency of multiple copies.

Now the proof is via induction on the CVN, the current version number of the replicated file f. First suppose that $CVN = 0$. Then version numbers of all copies are equal to zero, and their update sites cardinalities are each equal to n. Thus any majority partition will require at least $\lfloor n/2 \rfloor + 1$ sites. Since there can be at most one such partition, our claim follows.

Next assume that our claim holds for $CVN = k-1$ and that $CVN = k$. Suppose that the kth update was made in a partition called Q. Let $Q_1, \ldots, Q_m$ denote other partitions in the system at the time of this update. By our induction hypothesis, it follows that $Q_1, \ldots, Q_m$ are not majority partitions.

Note that whenever the file is updated or a copy is allowed to catch up, we increment the versions numbers of participating copies by one. Thus, the version numbers of current copies are always monotonically increasing, any partition P formed entirely from the sites in $Q_1, \ldots, Q_m$ will not be a majority partition. Thus, any majority partition P will have to have more than half of the current copies from Q. Since there can be at most one such partition at one time, the claim follows. □

We should note that whenever the **Make-Current** procedure permits an old copy to catch up, we treat this operation like an update in that we increment the version numbers of the participating copies by one. In the earlier version of this algorithm [7], the version numbers were kept the same, only the other values were adjusted to reflect the existence of the additional current copy. We have made this change in the original algorithm based on the availability results of our stochastic model; the current version results in greater availability of the replicated files in most cases.

IV. AVAILABILITIES OF VOTING AND ITS ENHANCEMENTS

A. The stochastic model

This section will compare the availabilities provided by the four algorithms described in this paper: ordinary voting, voting with a primary site, dynamic voting, and dynamic voting with linearly ordered copies. In general, a sequence of failures, repairs, and update requests can occur in such a way that one algorithm can accommodate a request when another cannot, and vice versa. The real question is this: which algorithm is more *likely,* in the long run, to be able to handle any given update request? That is, which algorithm has greater *availability?*

In this section we develop a stochastic model to make precise what is meant by the phrase "more likely" in the preceding paragraph. We state several theorems that compare the availabilities of the algorithms under the assumptions of our model. The main result: **of the four algorithms, dynamic voting with linearly ordered copies provides the greatest availability.**

We now introduce the four assumptions we make to model stochastically the update availabilities of the network under these algorithms. The first four assumptions duplicate assumptions that Pâris uses to analyze the availability of his *voting with witnesses* scheme [9]. The fifth assumption, however, causes our model to deviate from his. Here are the assumptions.

- The communication links between sites are infallible. Only sites go up and down.
- The failures at the various sites form independent Poisson processes with *failure rate* λ. For any given site that is up (functioning), the probability it goes down (fails) at or before the next t time units is $1 - e^{-\lambda t}$.
- Similarly, the repairs at the various sites form independent Poisson processes with *repair rate* μ. As Pâris notes [9, page 608], this assumption is less reasonable than the previous one, but both are necessary if we wish to model the network's behavior by a Markov process.
- Updates are instantaneous. We ignore communication delays in the commit protocol.
- Updates are frequent: after any failure or repair, an update always arrives at a functioning site and is processed before the next failure or repair. An alternative assumption that yields the same model is frequent polling: after any failure or repair, the functioning sites communicate to determine the new status of the system before the next failure or repair. In either form, this assumption permits great reduction in the number of states in the Markov process that describes the network's behavior.

We hasten to remark that the algorithms require none of these assumptions to operate properly. The assumptions are made only to provide a model whose analysis is tractable.

The literature contains two measures of availability. The more standard is the limit as t goes to infinity of the probability that a majority partition exists at time t, where the definition of "majority" depends on the algorithm used. We use this measure in this report, following [13] and [9]. An alternative measure is the limit as t goes to infinity of the probability that an update arriving at an arbitrary site at time t will succeed. This alternative measure, which we used in [6], requires not only that a majority partition exist, but also that the update arrive at a functioning site. Each formula in this paper requires only a trivial modification to use the alternative measure. Most of the algorithm comparisons in this paper hold for either measure of availability.

B. Availabilities of ordinary voting and voting with a primary site

The mean time to failure of a functioning site is $1/\lambda$. The mean time to repair of a failed site is $1/\mu$. It follows that for the Poisson process describing the behavior of the sites, the probability any given site is up at any particular time is

$$\frac{1/\lambda}{1/\lambda + 1/\mu}, \quad \text{that is,} \quad \frac{\mu}{\lambda + \mu}$$

The well-known availability of the voting algorithm is

$$Voting = \sum_{k=\lfloor n/2 \rfloor + 1}^{n} \binom{n}{k} \left[\frac{\mu}{\lambda + \mu}\right]^{k} \left[\frac{\lambda}{\lambda + \mu}\right]^{n-k}$$

The voting algorithm with a primary site retains a majority when exactly half of the sites are functioning, if the functioning sites include the primary site. Thus the availability of the voting algorithm with a primary site is exactly the same as the availability of ordinary voting when there are an odd number of sites, and contains the additional term

$$\frac{1}{2}\binom{n}{n/2}\left[\frac{\mu}{\lambda + \mu}\right]^{n/2}\left[\frac{\lambda}{\lambda + \mu}\right]^{n/2}$$

if n is even.

These same formulas could also be obtained by drawing the state diagram for the birth-death process that describes the number of failed sites and solving the resulting balance equations. We use just such a procedure to analyze the dynamic voting algorithms.

C. Availability of dynamic voting with linearly ordered copies

The system begins with all n sites in the majority partition. Eventually one site fails. Our fourth assumption insures that before another failure occurs or the failed site is repaired, an update arrives at a functioning site. The majority partition finds that it now contains $n-1$ of the n sites with up-to-date copies of the file—still a majority. The update sites cardinality is decremented to $n-1$ at each of the $n-1$ functioning sites, to reflect the number of sites participating in this update. If n is odd (so that $n-1$ is even), then the distinguished site entry at each of the functioning sites is also adjusted. If a second failure then occurs, the majority partition will soon thereafter discover that it contains $n-2$ of the $n-1$ sites with up-to-date copies of the file—still a majority, so the update sites cardinality at the $n-2$ functioning sites will be adjusted to reflect this new majority partition.

The process continues, with the update sites cardinality at the participating sites always increasing or decreasing by one, until there are only two sites in the majority partition. Call these sites A and B. The distinguished site entry associated with both copies of the file will be whichever of A and B is greater (in the given linear order), say, site A. Now suppose a failure occurs. Suppose first that the functioning site is greater (in the given linear order) than the newly-failed site, that is, suppose that site B fails. The single functioning site (site A) forms a new majority partition, even though it would not have done so under dynamic voting without linearly ordered copies. If the single functioning site (site A) now fails, the system is unavailable—subsequent updates are blocked. From this state, one of two events can occur.

- Site A might be repaired. The single-site majority partition is restored and the action of the network continues in the fashion described thus far.
- One or more of the $n-1$ other failed sites might be repaired. If sometime later site A is repaired and an update arrives, site A again can accept updates. In this case, however, the newly-formed majority partition will also include the other sites that have meanwhile been repaired.

We have just described the top two rows of the state diagram shown in Figure 2. (This and all succeeding figures appear at the end of the paper.) State (X,Y,Z) is the state in which:

- The update sites cardinality of each current copy of the file is Y.
- X of these Y sites are up.
- Z of the $n-Y$ other sites are up.

Arcs in the state diagram indicate the rate at which the system moves from state to state.

The system begins in state $(n,n,0)$ and moves back and forth along the second row until it reaches state (2,2,0). We described the situation in which the lesser of the two-site majority partition is next to fail; the system moves to state (1,1,0) and forms a one-site majority partition. The top row of the diagram contains states in which the single member of the one-site majority partition is down, while other sites are repaired and fail again.

Now return to the situation when the majority partition contains exactly two functioning sites, that is, state (2,2,0). This time, suppose that the greater (in the linear ordering) of the two sites is the first to fail. The system moves to state (1,2,0) on the third row of the diagram. This row contains states in which the greater member of the two-site majority partition is down and the lesser member is up, while again other sites are repaired and fail. The action along the fourth row is similar; there both members of the two-site majority partition are down. Note that from the fourth row, the system can move to the second row (if the greater of the two-site majority is repaired) or to the third row (if the lesser of the two-site majority is repaired).

An update request will be accepted if the network is in any of the states on the second row of Figure 2. The probability the system is in one of the second-row states can be found by setting flow-in equal to flow-out for each state and solving the resulting balance equations.

D. Availability of dynamic voting

Dynamic voting can be analyzed by reasoning similar to that in the previous subsection. Reference [6] contains the state diagram for dynamic voting. This diagram is akin to Figure 2, the diagram for dynamic voting with linearly ordered copies. Again, the system can be solved by setting flow-in equal to flow-out and solving the resulting balance equations. Note that if there are no communication delays and updates are frequent, our dynamic voting algorithm is available for updates exactly when the Davcev-Burkhard algorithm [3] is available. Hence the analysis of the availability of dynamic voting applies equally well to the Davcev-Burkhard algorithm.

E. Analytic comparison of the availabilities

Throughout this subsection, we assume that the repair rate μ is larger than the failure rate λ. This assumption is quite natural, and without it, the performance of some of the algorithms would degrade as the number of sites increases. We give results only for three or more sites since the algorithms reduce to trivial or nonsense algorithms when there are fewer sites.

Let *Voting*, *Voting-Primary*, *Dynamic*, and *Dynamic-Linear* denote the availabilites of voting, voting with a primary site, dynamic voting, and dynamic voting with linearly ordered copies, respectively. The following comparisons between the algorithms are immediate.

- *Voting-Primary* = *Voting* when there are an odd number of sites.
- *Voting-Primary* $\geq$ *Voting*.
- *Dynamic-Linear* $\geq$ *Dynamic*.

Figure 2 and the corresponding diagram for dynamic voting yield balance equations. For fixed μ, λ, and n, these balance equations are easily solved by your favorite numerical technique for systems of linear equations. Sample results obtained this way are described in the next subsection. However, it appears difficult to find a closed-form symbolic solution to the $4n-2$ equations obtained from Figure 2 or the $3n-3$ equations from the diagram for dynamic voting. When n is small, MacSyma or the like can solve the systems symbolically; the proofs of Theorems 3 and 4 use such a technique. Theorem 5, which applies to arbitrarily large n, has a more complicated proof. The details are available from the authors.

Theorem 3. When there are exactly 3 sites:
Dynamic-Linear > *Voting* > *Dynamic*.

Theorem 4. When there are exactly 4 sites:
if $\mu/\lambda < 1.8$ (approximately):
Dynamic-Linear > *Voting-Primary* > *Dynamic* > *Voting*;
but otherwise:
Dynamic-Linear > *Dynamic* > *Voting-Primary* > *Voting*.

Theorem 5. When there are 5 or more sites:
Dynamic > *Voting-Primary*†††.

In sum, *dynamic voting with linearly ordered copies has greater availability than any of the other three algorithms.*

F. Numerical comparison of the availabilities

Figure 3 shows availability graphed against the number n of sites, when the repair rate μ is twice the failure rate λ. The behavior shown is typical of other repair/failure ratios as well. Note that dynamic voting with linearly ordered copies is best and that dynamic voting is a close second except when the number of sites is small. All four algorithms converge to a completely available system as the number of sites grows large. However, the two dynamic algorithms converge much more rapidly than the two voting algorithms. Also note that voting with a primary site smooths out the nonmonotonicity of ordinary voting.

Figures 4, 5, and 6 show availability graphed against the repair rate μ, for ten, three, and four sites, respectively. The behavior shown in Figure 4 (ten sites) is typical of the behavior for five or more sites. In all the figures, dynamic voting with linearly ordered copies is best. The curve for dynamic voting lies below the curve for voting in Figure 5 (three sites), but above it in Figure 4 (ten sites). The curve-crossing shown in Figure 6 occurs only when there are exactly four sites. Of special note is that in Figure 4, each dynamic algorithm has high availability even when the repair rate is near the failure rate; the same is not true of voting or voting with a primary site.

V. FUTURE RESEARCH

Hybrids of the four algorithms discussed in this paper are possible. For example, there is a mechanism that permits a switch from dynamic voting to ordinary voting when the size of the majority partition falls below a threshold. We are investigating the availability of such hybrids. We are also developing more elaborate models in which to compute availability, in particular, models without the assumption of frequent updates.

Each of the four algorithms easily generalizes to the setting where different weights are assigned to sites. It would be interesting to know how to assign optimal weights given the particular repair and failure rates for the sites.

††† The claim that dynamic voting has greater availability than voting when there are five sites appears at odds with our results in [6], where we claimed the opposite if μ is near λ. In [6], however, we used a different measure of availability—we required not only that a majority partition exist, but also that the update request arrive at a functioning site. The comparisons for three, four and five sites are sensitive to the measure used. However, dynamic voting with linearly ordered copies is still the best of the four algorithms under the alternative measure, except if there are three sites.

References

1. P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys* **13**(2) pp. 185-221 (1981).
2. W. A. Burkhard, B. E. Martin, and J.-F. Paris, "The Gemini replicated file system test-bed," *Proc. IEEE 3rd Int'l. Conf. on Data Engineering,* pp. 441-448 (1987).
3. D. Davcev and W. Burkhard, "Consistency and recovery control for replicated files," *Proc. 10th ACM Symp. on Operating Systems Principles,* pp. 87-96 (1985).
4. S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned networks," *ACM Computing Surveys* **17**(3) pp. 341-370 (1985).
5. D. K. Gifford, "Weighted voting for replicated data," *Proc. 7th Symp. on Operating System Principles,* pp. 150-162 (1979).
6. S. Jajodia and D. Mutchler, "Dynamic voting," *Proc. ACM SIGMOD Int'l. Conf. on Management of Data,* pp. 227-238 (1987).
7. S. Jajodia, "Managing replicated files in partitioned distributed database systems," *Proc. IEEE 3rd Int'l. Conf. on Data Engineering,* pp. 412-418 (1987).
8. W. H. Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *ACM Computing Surveys* **13**(2) pp. 149-183 (1981).
9. J.-F. Paris, "Voting with witnesses: A consistency scheme for replicated files," *Proc. IEEE Int'l. Conf. on Distributed Computing,* pp. 606-612 (1986).
10. D. S. Parker, Jr., G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline, "Detection of mutual inconsistency in databases," *IEEE Trans. on Software Engineering* **SE-9** (3) pp. 240-247 (1983).
11. M. Pease, R. Shostak, and L. Lamport, "Reaching agreements in the presence of faults," *Journal of ACM* **27**(2) pp. 228-234 (1980).
12. J. Seguin, G. Sergeant, and P. Wilms, "A majority consensus algorithm for the consistency of duplicated and distributed information," *Proc. IEEE Int'l. Conf. on Distributed Computing Systems,* pp. 617-624 (1979).
13. P. G. Selinger, "Replicated data," pp. 223-231 in *Distributed Databases,* ed. I. W. Draffen and F. Poole,Cambridge University Press , Cambridge (1980).
14. D. Skeen and M. Stonebraker, "A formal model of crash recovery in a distributed system," *IEEE Trans. on Software Engineering* **SE-9** (3) pp. 219-228 (1983).
15. R. H. Thomas, "A solution to the concurrency control problem for multiple copy databases," *Proc. IEEE Compcon,* pp. 56-62 (Spring, 1978).

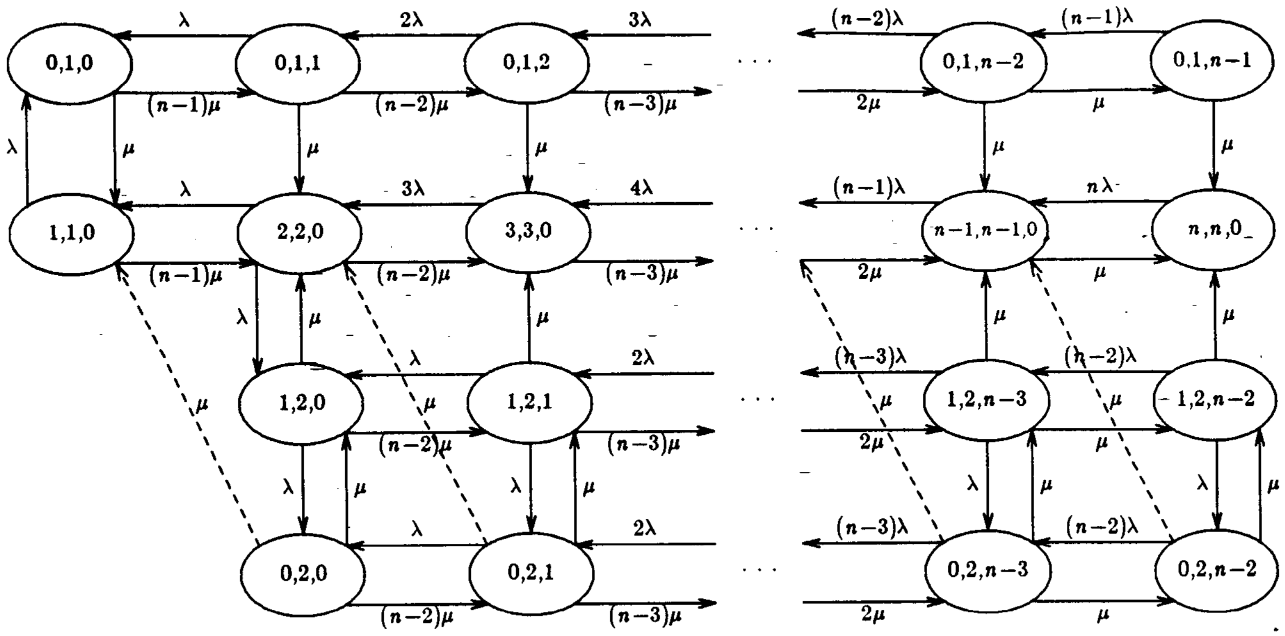

Figure 2. State diagram for dynamic voting with linearly ordered copies.

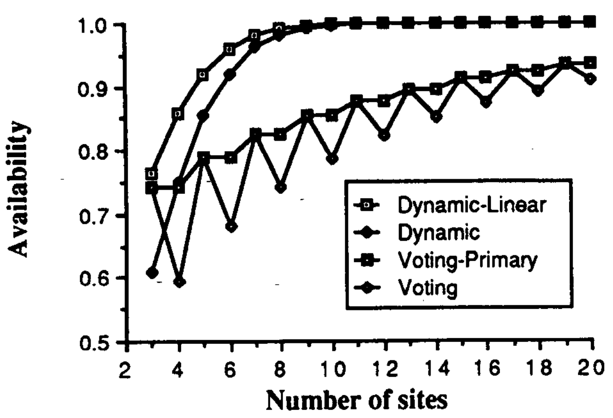

Figure 3. Repair rate is twice failure rate.

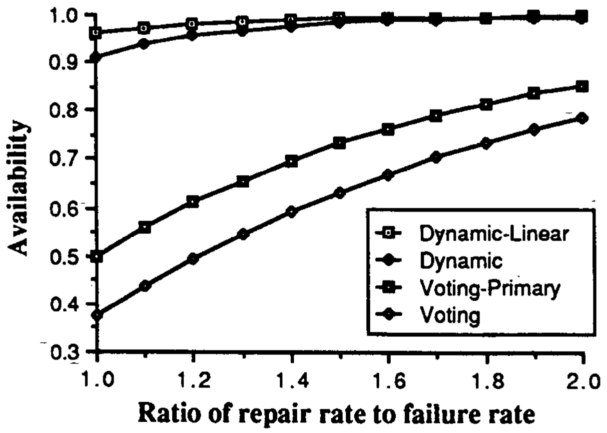

Figure 4. Availability when there are 10 sites.

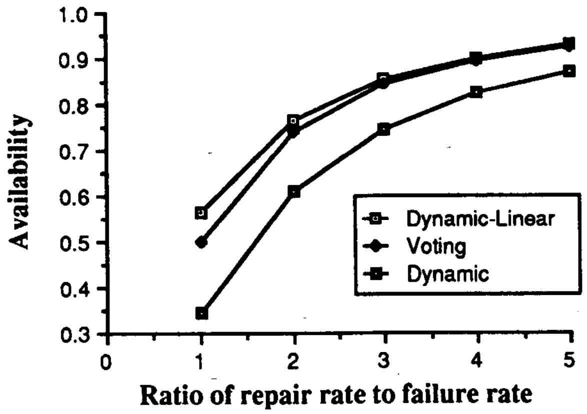

Figure 5. Availability when there are 3 sites.

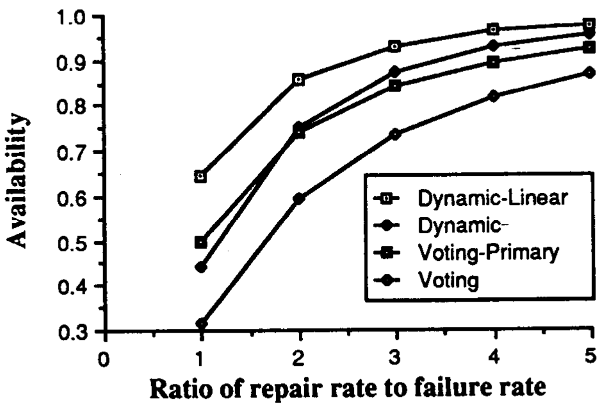

Figure 6. Availability when there are 4 sites.