DYNAMIC VOTING

Sushil Jajodia
David Mutchler

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

ABSTRACT

In a voting-based algorithm, a replicated file can be updated in a partition if it contains a majority of copies. In this paper, we propose an extension of this scheme which permits a file to be updated in a partition provided it contains a majority of up-to-date copies Our scheme not only preserves mutual consistency of the replicated file, but provides improvement in its availability as well We develop a stochastic model which gives insight into the improvements afforded by our scheme over the voting scheme

I INTRODUCTION

A *partitioning* of a distributed database (DDB) occurs when the sites in the network split into groups of communicating sites due to node or communication failures The sites in each group can communicate with each other, but no site in one group is able to communicate with sites in other groups We refer to each such group as a *partition* The algorithms which allow a partitioned DDB to continue functioning generally fall into one of two classes † Those in the first class take a *pessimistic* approach in that they share the philosophy that mutual consistency is of considerably greater importance than availability [1,2,3,8,11,13,18,25,29] Consistency is enforced by permitting files to be accessed only in one partition As a consequence, any updates which are permitted in a partition do not conflict with updates in other partitions, assuring mutual consistency of data when partitions are reunited

The algorithms in the second class take the approach that the database must be available even in the face of node or communication link failures and permit every group of sites in a partitioned DDB to perform new updates [9,20,22,30,31] Since now the databases of the partitions will clearly diverge, they require a strategy for conflict detection and resolution Usually, rollbacks are used as a means for preserving consistency, conflicting transactions are rolled back when partitions are reunited Since coordinating the undoing of transactions is a very difficult task, these methods are called *optimistic* since they are useful primarily in a situation where the number of items in a particular database is large and the probability of conflicts among transactions is small

In this paper, we consider only the pessimistic strategies Although these methods guarantee that there will not be more than one partition performing the update at the same time, they cannot guarantee that there will be a partition which can perform updates at any given time All known methods share the drawback that failures can occur in such a way that no updates can be performed anywhere in the system until these failures are repaired While this is undesirable, it is a necessary property exhibited by any protocol which tolerates network partitioning [27], thus, the challenge is to devise consistency control algorithms which preserve mutual consistency of replicated files and which, at the same time, provide improvement in the availability of files over that of existing schemes

Voting [13,25,29] is the best known example of a pessimistic scheme In its simplest form, an item can be updated in a partition if it contains more than half of the sites where the file is replicated Thus, if a file is replicated at 100 sites, a majority partition will have to contain at least 51 of these sites In the event that

† There are algorithms (see [12,15,23,28] for example) which do not belong to either of these two classes, however, they require a priori knowledge of the kind of updates to be made to the file We make no such assumption in this paper

there does not exist such a partition, no updates can occur anywhere in the system

In this paper we propose a generalization to the voting scheme, called the *dynamic voting scheme* It has several advantages beyond those of the voting scheme

- Unlike the voting scheme, the number of sites necessary for an update is a function of the number of *current* copies in existence at the time of the update
- This required quorum changes dynamically without any manual intervention
- A file can be updated in a partition if it contains more than half of the *current* copies As a consequence, a file may be replicated at 100 different sites, yet it is possible for this scheme to allow updates with as few as 2 copies accessible
- Like the voting scheme, our protocol is simple to state as well as implement It requires only a slight modification to the voting scheme
- It does not require a complicated message-based coordination mechanism

We have called our algorithm dynamic voting since it has the same availability as the scheme proposed by Davcev and Burkhard in [8], whenever one scheme permits an update to occur in the system, the other does also Their scheme, however, suffers from severe drawbacks (see section III) While our scheme has all the advantages enjoyed by their scheme, ours does not suffer from any of the drawbacks mentioned in section III

In section VI, we present a stochastic model and compute the availability of the simple voting and dynamic voting algorithms in the context of that model We prove that under the assumptions of the model, the availability of the dynamic voting algorithm is greater than or equal to the availability of the voting algorithm if there are four or more sites, except for a single special situation (five sites and slow repairs)

II. FORMAL SPECIFICATION OF THE PROBLEM

The distributed database 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 [24] We do not include Byzantine failures [21] 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 sites runs a *concurrency control protocol* which ensures that the execution of all transactions at every site is serializable [6, 17] 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 In a *pessimistic* consistency control protocol, a replicated file can be 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 [27], 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

III. PREVIOUS PESSIMISTIC ALGORITHMS

In this section, we outline several existing pessimistic schemes and point out how these methods restrict availability (An excellent survey of several of these strategies is given in [10]) In a subsequent section, we shall give our consistency control algorithm which provides greater file availability

Primary Site [3] Here each file has a primary physical copy, and the permission to update any copy resides with the node containing the primary copy Therefore, the partition containing the node with the primary copy is defined to be a majority partition No updates can occur anywhere in the system if the site containing the primary copy crashes

True-Copy Token [18] In this scheme, there is a token associated with each file, and a partition is a majority partition if it contains a site which has the file token There are two drawbacks to this scheme First, it is possible that tokens can get lost, and it is a nontrivial problem to recreate lost tokens Second, if the token happens to reside in a partition containing only rarely used sites, the file will, for all practical purposes, become unavailable

Voting [13, 29] In a voting-based system, a partition is a majority partition if it contains more than half of the sites where the file is replicated By definition, there can be at most one such partition In the event that there does not exist a partition containing a majority of sites, no updates can occur anywhere in the system A brief description of the basic voting algorithm is given in next section

Missing Writes Algorithm [11] This algorithm runs in two modes, a normal mode in which it reads any copy and writes all copies and a failure mode in which it uses voting Several files must be maintained to keep track of updates which were made while the network is partitioned—the so-called "missing write" information—so they may be propagated to other sites later These files need not be maintained during normal operation, but can grow rapidly during failures [10, page 355]

Accessible Copies Algorithm [1, 2] This algorithm is a generalization of the voting scheme since it allows different partitions to have different quorums for a majority The drawback is that the required quorum for each partition is fixed a priori Moreover, this algorithm "relies on coordinating a consistent view among the nodes" [4, page 197] By contrast, the required quorum in our scheme changes dynamically as well as automatically No elaborate mechanism is required for this purpose

Davcev-Burkhard Algorithm [8] As we mentioned in the introduction, the Davcev-Burkhard algorithm has the same availability as that of our scheme Their scheme, however, has many severe drawbacks

- The major difficulty is that they assume that each site maintains an n-bit binary *connection vector* that it uses for checking the availability of remote sites The jth entry of the connection vector at site i is equal to 1 if site i can communicate with site j, it is equal to 0 otherwise It is assumed that the "changes in the system configuration resulting from site failure or network partitions are instantaneously recorded within the proper connection vectors" (see [8, page 89]) Perhaps it is possible to come up with an efficient way to implement the connection vector scheme, but it is sure to entail substantial communication overhead
- Their scheme requires each copy to have an associated *version vector*†, which is an array of n integers where n is the number of copies, and these vectors have the problem that it is difficult to keep them consistent among communicating sites while the system is being reconfigured
- The use of version vectors requires more storage as well as processing time for their manipulation, especially if n is large
- If the number of copies varies with time, the length of the version vectors may need to change dynamically, which is not easy to program

Our scheme manages to avoid all these flaws while retaining all the advantages of the Davcev-Burkhard algorithm

Finally, two recent papers [4, 5] also deserve mentioning They contain protocols for dynamically reassigning votes upon site or communication link failures in an attempt to make the system more resilient

IV. THE VOTING ALGORITHM

We present a brief overview of the original voting algorithm to show that our scheme requires only a slight modification to it

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 Thus, a version number represents up-to-dateness of a copy A site can process an update provided it belongs to a partition containing at least $\lfloor n/2 \rfloor$ other sites If so, it must first ensure that those participating copies which are not up-to-date 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

Thus, in the voting algorithm, we see that 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 section, we present a generalization of the voting scheme that sometimes permits a partition to perform updates even when it does not contain more than half of the sites Like the voting algorithm, it preserves mutual consistency of replicated files, it also provides improvement in their availability

V THE DYNAMIC VOTING ALGORITHM

A. Description of the algorithm

In this section, we provide a description of our algorithm in its simplest form where each file is assigned an equal weight (of one) It is possible to permit copies having different weights This generalization is straightforward, so the details are omitted

We associate with each copy of the file f a version number and an update sites cardinality which are

† Called *partition vector* in [8]

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 let VN_i be equal 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 a majority 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 set to f_i, then SC_i is set equal to the total number of copies which were updated during this update

Each site which has a copy of the file f must maintain its version number and its update sites cardinality. We can now describe our consistency control algorithm. Suppose a site receives an update request. It must first determine if it belongs to a majority partition. If so, it can perform the update, otherwise, it must reject the update. A site can easily decide if it belongs to such a partition as follows

Suppose a site A_1 wishes to determine if it belongs to a majority partition. It asks all sites with which it can communicate and which have copies of the file f for their version numbers and update sites cardinalities. Let $A_2, \ldots, A_m$ denote these sites. Let VN_i and SC_i denote, respectively, the version number and update sites cardinality of the copy f_i of the file f at site A_i. Now,

1) The site A_1 calculates

$$\text{the value } M = \max\{VN_i \mid 1 \le i \le m\},$$
$$\text{the set } I = \{A_j \mid VN_j = M, 1 \le j \le m\},$$
$$\text{and the value } N = \max\{SC_k \mid A_k \in I\}$$

2) If $\text{card}(I) > N/2$, then A_1 lies in a majority partition and can proceed with the update, otherwise, it does not belong to such a partition and must reject the update.† Note that we can permit A_1 to read the file if its partition contains exactly one-half of the current copies (i.e., if the values $\text{card}(I)$ and $N/2$ coincide)

† Notation. For a set X, $\text{card}(X)$ denotes its cardinality

3) Whenever an update is made, it must be made to all sites in the set I. Moreover, the values VN_i and SC_i at every site in I must be modified as well by letting

$$VN_i = M + 1$$
$$SC_i = \text{card}(I)$$

Each time an update succeeds at a site, it must commit the update together with the version number as well as the update sites cardinality. Thus an update operation at a site is atomic in the sense that either all three operations—updates to the file, version number, and update sites cardinality—are performed in entirety or are not performed at all

Our scheme can be best illustrated by an example

Example 1. 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 a majority 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. Thus site A can process the update, after which the database state will be:

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

In our scheme, updates are always made to the up-to-date copy of the file, those copies which are not must be brought up-to-date before the new update can proceed. For this purpose, sites from time to time determine if their copies are current. If not, they must take steps to ensure that their copies are brought up-to-date. We describe how this is done next

A site A_1 can easily determine whether or not its copy is current as follows.

i) Let $A_2, \ldots, A_m$ denote other sites in A_1's partition A_1 needs to examine the version numbers and update sites cardinalities of all these sites

ii) A_1's copy is not current if its version number is less than the version number of some other site in its partition A_1's copy is current if it has a copy with the highest version number and if, moreover, it belongs to a majority partition

Suppose the site A_1 realizes that its copy is not current and wishes to catch up It may do so if and only if it belongs to a majority partition Specifically, site A_1 must execute these steps

a) The site A_1 must examine the version numbers and the update sites cardinalities of all sites in its partition Let $A_2, \ldots, A_m$ denote the other sites of the partition

b) A_1 computes

$$\text{the value } M = \max\{VN_i \quad 1 \leq i \leq m\},$$
$$\text{the set } I = \{A_j \quad VN_j = M, \ 1 \leq j \leq m\},$$
$$\text{and the value } N = \max\{SC_k \quad A_k \in I\}$$

c) If $\text{card}(I) > N/2$, then the sites in A_1's partition together form a majority partition, so A_1 can bring its copy to up-to-date status by requesting the missing updates from any site in I The version number VN_1 of the copy A_1 is set to equal M, and the update sites cardinality SC_1, along with the update sites cardinalities of each site in I, is set equal to $N + 1$

d) Otherwise, A_1 cannot request missing updates from anyone

There is a good reason why we do not allow sites to merge arbitrarily The example below illustrates why some form of restriction is necessary

Example 2. Continuing with the example given above, suppose that site B resumes communication with sites D and E Our algorithm will not permit these sites to merge for if we were to allow this to happen, the database state will become

	A	C	B	D	E
VN	11	11	10	10	10
SC	2	2	5	5	5

The partition BDE will conclude that it too is a majority partition Thus, we will have two "majority" partitions which can make updates independently, possibly leading to inconsistencies between the partition copies

On the other hand, in the previous example, suppose sites A, C, D, and E were to unite as a single partition Then the copies at D and E can be brought to the current status The database state will then become

	A	C	D	E	B
VN	11	11	11	11	10
SC	4	4	4	4	3

B. Addition and deletion of sites

In this section, we show how we may add or delete copies of the replicated file f dynamically from the system Suppose a site A_1 that does not have a copy of file f wishes to have one The site A_1 sets the version number as well as the update sites cardinality of its copy to be $-\infty$ where $-\infty$ is some value much smaller than any actual values for the version numbers and update sites cardinalities Then, it may initiate the steps which permit a site's copy to catch up

It is equally easy for copies to be deleted from the system Suppose a site A_1 decides that it no longer wants to have its own copy of the file f If A_1's copy is not current, it can delete its copy any time it desires It is desirable, however, that A_1 notify other sites having the copies that it no longer maintains a copy and should be excluded from their future attempts of forming a majority If A_1's copy is current, it will be permitted to delete its copy in one of the following two ways

A The site A_1 may do so if it belongs to a partition having *all* current copies of the replicated file If this is so, it requests all sites having the current copies to decrement their update sites cardinalities by one

B There is another option if A_1 is a member of a partition containing only a *majority* of current copies In this case, all sites in the partition that have current copies increment their version numbers by one and decrement their update sites cardinalities by one

C. Proof of correctness

In this section, we briefly argue that our consistency control protocol is correct It suffices to show that there exists by our algorithm at most one majority partition at any time for then the replicated file can be updated in at most one partition which is enough to guarantee the mutual consistency of multiple copies

It is not difficult to see that our algorithm permits at most one majority partition to process updates at any time since it has the following two key properties

- The version numbers of current copies are monotonically increasing and unique to a given majority See steps 2) and 3) of our algorithm
- For a given version number, any two consecutive majority partitions have at least one site in

common, and the update sites cardinality of this common site gives the total number of current copies in existence See step c) of our algorithm

VI. AVAILABILITY OF VOTING AND DYNAMIC VOTING

A The stochastic model

When there are just three sites with copies of the replicated file, if an update request can be accommodated under the dynamic voting algorithm, it can also be accommodated under the voting algorithm When there are four or more sites, however, a sequence of failures, repairs, and update requests can occur in such a way that the dynamic voting algorithm can accommodate a request when the voting algorithm cannot, and vice versa For example, suppose a five-site network splits into two partitions,

$$A \quad B \quad C \mid D \quad E$$

and an update appears at site A Both voting and dynamic voting permit the update to be processed If site C thereafter forms a third partition by itself,

$$A \quad B \mid C \mid D \quad E$$

dynamic voting permits updates arriving at sites A or B to be processed, while voting rejects all update requests On the other hand, if the two-site A-B partition now splinters into two single-site partitions while site C joins sites D and E,

$$A \mid B \mid C \quad D \quad E$$

ordinary voting would permit updates arriving at sites C, D, or E to be processed, while dynamic voting rejects all update requests in this state

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 Key aspects of the model are no attempt is made to model changes to the network topology (only sites go up and down), updates are assumed to be frequent relative to failures and repairs, and communication delays in the commit protocol are ignored We show that under the assumptions of the model, the dynamic voting algorithm provides an improvement over the voting algorithm in the availability of replicated files if there are four or more sites, except for a single special situation (five sites and slow repairs)

We now introduce the five assumptions we make to model stochastically the update availability of the network under these two algorithms The first four assumptions duplicate assumptions that Pâris uses to analyze the availability of his *voting with witnesses* scheme [19] The fifth assumption, however, causes our model to deviate from his Here are the assumptions, their justification comes next

- The communication links between sites are infallible Only sites go up and down Any site that is up can send a message to any other site that is up
- 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
- 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

The first assumption seems rather odd the model prohibits precisely the phenomenon—partitioning—that voting and dynamic voting are designed to tolerate! We make this assumption in order to sidestep the countless network topologies that might occur as links fail Our choice has been to use rather strong assumptions to obtain a proof of a strong theorem, rather than to use weaker assumptions and present only simulation results Needless to say, weakening the assumptions while maintaining the results is one focus of future research

As Pâris notes [19, page 608], the third assumption is less reasonable than the second, but both are necessary if we wish to model the network's behavior by a Markov process The fourth assumption (instantaneous updates) is another simplification necessary to maintain the Markovian model Because we are interested in a comparison of voting and dynamic voting, and because both algorithms face similar communication delays in the commit protocol, the fourth assumption does not seem unreasonable

On the other hand, the fifth assumption is not necessary to force a Markovian model In either of its forms, the fifth assumption permits great reduction in the number of states in the Markov process that describes the network's behavior This simplifies the analysis and led to discovery of the theorem that appears at the end of this section

Our dynamic voting algorithm is available for updates exactly when the Davcev-Burkhard algorithm is available, the two algorithms have the same

availability Hence this section applies equally well to the Davcev-Burkhard algorithm

B Two measures of availability

The standard measure of availability (see [26] and [19], for example) 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 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 requires not only that a majority partition exist, but also that the update arrive at a functioning site In this report, we will use the alternative measure, deeming it more appropriate Each formula in this paper requires only a trivial modification to use the standard measure Furthermore, use of the standard measure only *increases* the superiority of dynamic voting over voting That is, the measure of availability that we use in this report shows dynamic voting in its least favorable light

C Availability under the voting algorithm

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 availability of the voting algorithm is

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

The $\frac{k}{n}$ term in the summation reflects the fact that an update request can be processed only if the site at which the request arrives is one of the k sites in the majority partition

This same formula 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 will use just such a procedure to analyze the dynamic voting algorithm

D Availability under dynamic voting

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 cardinalities are adjusted to $n-1$ at the $n-1$ functioning sites 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 cardinalities will be adjusted to $n-2$ at the $n-2$ functioning sites The process continues, with update site cardinalities always increasing or decreasing by one, until there are only two sites in the majority partition and a failure then occurs The subsequent update is blocked—one out of two sites is not a majority Of these two sites in the most recent majority partition, call the one still up site U and the one now down site D From this state, one of three events can occur

- Site D might be repaired The two-site majority partition is restored and the action of the network continues in the fashion described thus far
- One or more of the other $n-2$ failed sites might be repaired If sometime later site D is repaired and an update arrives, the functioning sites include both (hence a majority) of the sites with up-to-date copies of the file In this case, however, the newly-formed majority partition will also include the other sites that have meanwhile been repaired
- Site U might fail Now both site U and site D must be repaired before a new majority partition will be formed Again any such newly-formed majority partition will also include any other sites that have meanwhile been repaired

The state diagram we have just described is shown in Figure 1 on the next page State (X,Y,Z) is the state in which

- The update sites cardinality of each up-to-date copy of the file is Y
- X of the Y sites with update sites cardinality Y 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

An update request will be accepted if it arrives at a functioning site and the network is in any of the states on the top row of Figure 1 Label the top-row states $A_0, A_1, \ldots, A_{n-2}$, from left to right Label the middle-row and bottom-row states B_0 through B_{n-2} and C_0 through C_{n-2}, respectively, again left to right In an abuse of notation, we will also let A_k denote the steady-state *probability* of state A_k, and similarly for the B_ks and C_ks The availability of the network is

$$\sum_{k=0}^{n-2} \frac{k+2}{n} A_k$$

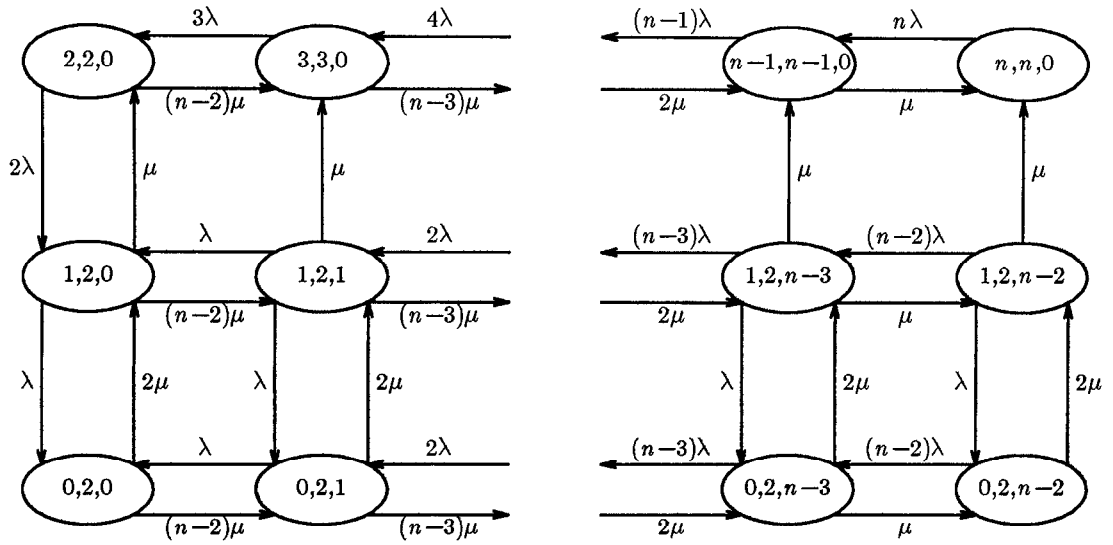

Figure 1. The state diagram for dynamic voting.

The $\frac{k+2}{n}$ term reflects the fact that the site at which the update request arrives must be one of the $k+2$ functioning sites in state A_k

To find the steady-state probabilities of the states in Figure 1, set flow-out equal to flow-in to obtain the following *balance equations,* one equation per state The three leftmost states have special equations

$$\Big[2\lambda + (n-2)\mu\Big] A_0 = 3\lambda A_1 + \mu B_0$$

$$\Big[\lambda + (n-1)\mu\Big] B_0 = \lambda B_1 + 2\mu C_0 + 2\lambda A_0$$

$$n\mu C_0 = \lambda C_1 + \lambda B_0$$

For the remaining top-row states ($k = 1, 2, \quad n-2$)

$$\Big[(k+2)\lambda + (n-k-2)\mu\Big] A_k$$

$$= (k+3)\lambda A_{k+1} + (n-k-1)\mu A_{k-1} + \mu B_k$$

For the remaining middle-row states ($k = 1, 2, \quad n-2$)

$$\Big[(k+1)\lambda + (n-k-1)\mu\Big] B_k$$

$$= (k+1)\lambda B_{k+1} + (n-k-1)\mu B_{k-1} + 2\mu C_k$$

For the remaining bottom-row states ($k = 1, 2, \quad n-2$)

$$\Big[k\lambda + (n-k)\mu\Big] C_k$$

$$= (k+1)\lambda C_{k+1} + (n-k-1)\mu C_{k-1} + \lambda B_k$$

By defining $A_{n-1} = B_{n-1} = C_{n-1} = 0$, the above equations are seen correct for the right-boundary states

One of these $3n-3$ equations is redundant Replace it by the equation that says the probabilities sum to one

$$\sum_{k=0}^{n-2} (A_k + B_k + C_k) = 1$$

It appears difficult to find a closed-form solution to the above system of equations However, for fixed μ, λ and n, the system is easily solved by your favorite numerical technique for systems of linear equations Furthermore, for fixed small values of n, the symbolic manipulator MACSYMA[TM†] can solve the system in terms of μ and λ The final word, however, is given by the theorem in the next section

† MACSYMA is a trademark of Symbolics, Inc It was originally developed by the Mathlab Group of the MIT Laboratory for Computer Science

E Results

Theorem Suppose the repair rate μ is at least as large as the failure rate λ Then the availability of dynamic voting is greater than the availability of voting if and only if

- there are 4 sites in the network, *OR*
- there are 5 sites in the network and the ratio of repair rate to failure rate is at least 1 3070, *OR*
- there are 6 or more sites in the network

Proof. The proof for 4 or 5 sites is by explicit, symbolic calculations performed by MACSYMA The more challenging proof for 6 or more sites is intricate but mathematically unsophisticated Details are available from the authors The superiority of voting when there are only 3 sites is obvious voting requires that *any* two sites (or more) be up, while dynamic voting requires that either all three sites or the *specific* two sites that formed the most recent majority partition be up □

The assumption that $\mu \geq \lambda$ will hold in any reasonable application The superiority of voting in the aberrant case of 5 sites and repair rate approximately equal to failure rate is best explained as a residual effect from the 3-site subsystem When there are 4 sites, this effect is overwhelmed by the poor performance of voting when the number of sites is even

Recall that we have used a non-standard measure of availability, namely, the limiting probability that an update arriving at some arbitrary site will be processed If we were to have used the more traditional measure—the long-term probability that a majority partition exists—the aberrant case disappears That is, under the traditional measure of availability, dynamic voting is better than voting if there are 4 or more sites

The content of our theorem is seen more easily in Figures 2 through 5 (These figures appear at the end of the paper, after the references) Each figure shows the availability of both algorithms In every figure except Figure 5, which is for 5 sites, the top curve is for dynamic voting, the bottom for voting Take heed that the scaling of the axes varies from figure to figure

Figures 2 and 3 display availability graphed against the number n of sites, for repair/failure ratios of 2 0 and 10 0, respectively As n grows large, the availability of each algorithm converges to $\frac{\mu}{\mu+\lambda}$, the probability that the site at which the update arrives is a functioning site Figure 2 shows with particular clarity the differences between the two convergences, namely, that the dynamic voting algorithm

- approaches its limiting value much more quickly,
- shows a *monotonic* increase in its availability

Figure 3 again shows availability versus number of sites but this time using a more reasonable value for the repair/failure ratio Here both voting and dynamic voting are not too far from the maximum possible availability even with only 4 sites Both are within 0001 of this maximum by the time the number of sites has reached 14

Figures 4 and 5 display availability versus the repair/failure ratio The number of sites is 4 and 5, respectively Observe that in Figure 4, there is a discernible difference between the two algorithms even for repair/failure ratios close to 20 However, the difference between voting and dynamic voting is most striking when the repair/failure ratio is close to 1 Thus the difference is most striking in situations that are not particularly reasonable Figure 5 displays the aberrant crossing behavior, such behavior occurs only when there are exactly 5 sites

VII OPEN PROBLEMS

In this paper, we have proposed a pessimistic consistency control algorithm for managing replicated files in the presence of network partitioning due to node and communication link failures Our algorithm

- requires only a slight modification to the voting scheme and
- provides improvement in the availability of replicated files over the voting scheme

We have developed a stochastic model which provides an estimate of this improvement We are continuing our efforts to analyze the availability of these algorithms Here are some open problems we face

- Our model insisted that updates be far more frequent than site failures and repairs Does the superior availability of the dynamic voting algorithm hold up when this assumption is relaxed?
- By not allowing link failures, we chose to ignore the network topology in our analysis We expect dynamic voting to be superior to voting in most (if not all) topologies, but we have not proved such a theorem in this paper The two algorithms should be compared under link failures from several particular initial topologies, perhaps a ring or a complete graph, for instance Or one might concentrate on (say) five sites and investigate the same six topologies that Barbara et al use in [5] Best would be to prove a theorem that shows the superiority of dynamic voting over some wide *class* of initial network topologies
- How does the delay in processing an update affect the comparison of voting and dynamic voting?

- We have compared dynamic voting to ordinary voting How does dynamic voting compare against other algorithms for maintaining the consistency of a replicated file in the presence of network partitioning? For instance, one might compare dynamic voting against the algorithms that Coan et al analyze in [7]
- Both the algorithms analyzed permit various enhancements [14,16] How do these enhancements affect the availability of the algorithms?

References

1 A El Abbadi, D Skeen, and F Christian, "An efficient, fault-tolerant protocol for replicated data management," *Proc 4th ACM Symp on Principles of Database Systems*, pp 215-228, 1985

2 A El Abbadi and S Toueg, "Availability in partitioned replicated databases," *Proc 5th ACM Symp on Principles of Database Systems*, pp 240-251, 1986

3 P A Alsberg and J D Day, "A principle for resilient sharing of distributed resources," *Proc 2nd Int'l Conf on Software Engineering*, pp 562-570, 1976

4 D Barbara, H Garcia-Molina, and A Spauster, "Protocols for dynamic vote reassignment," *Proc 5th ACM Symp on Principles of Distributed Computing*, pp 195-205, 1986

5 D Barbara, H Garcia-Molina, and A Spauster, "Policies for dynamic vote reassignment," *Proc IEEE Conf on Distributed Computing*, pp 37-44, 1986

6 P A Bernstein and N Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys*, vol 13, no 2, pp 185-221, 1981

7 B A Coan, B M Oki, and E K Kolodner, "Limitations on database availability when networks partition," *Proc 5th ACM Symp on Principles of Distrib Computing*, pp 187-194, 1986

8 D Davcev and W Burkhard, "Consistency and recovery control for replicated files," *Proc 10th ACM Symp on Operating Systems Principles*, pp 87-96, 1985

9 S B Davidson, "Optimism and consistency in partitioned distributed database systems," *ACM Trans on Database Systems*, vol 9, no 3, pp 456-481, 1984

10 S B Davidson, H Garcia-Molina, and D Skeen, "Consistency in partitioned networks," *ACM Computing Surveys*, vol 17, no 3, pp 341-370, 1985

11 D L Eager and K C Sevcik, "Achieving robustness in distributed database systems," *ACM Trans on Database Systems*, vol 8, no 3, pp 354-381, 1983

12 M J Fischer and A Michael, "Sacrificing serializability to attain high availability of data in an unreliable network," *Proc ACM Symp on Principles of Database Systems*, pp 70-75, 1982

13 D K Gifford, "Weighted voting for replicated data," *Proc 7th Symp on Operating System Principles*, pp 150-162, 1979

14 S Jajodia and C A Meadows, "Mutual consistency in decentralized distributed systems," *Proc IEEE 3rd Int'l Conf on Data Engineering*, pp 396-404, 1987

15 S Jajodia, "Managing replicated files in partitioned distributed database systems," *Proc IEEE 3rd Int'l Conf on Data Engineering*, pp 412-418, 1987

16 S Jajodia and D Mutchler, "Enhancements to the voting algorithm," *Proc 13th Int'l Conf on Very Large Data Bases*, 1987 Submitted

17 W H Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *ACM Computing Surveys*, vol 13, no 2, pp 149-183, 1981

18 T Minoura and G Wiederhold, "Resilient extended true-copy token scheme for a distributed database system," *IEEE Trans on Software Engineering*, vol SE-8, no 3, pp 173-189, 1982

19 J-F Paris, "Voting with witnesses A consistency scheme for replicated files," *Proc IEEE Int'l Conf on Distributed Computing*, pp 606-612, 1986

20 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*, vol SE-9 , no 3, pp 240-247, 1983

21 M Pease, R Shostak, and L Lamport, "Reaching agreements in the presence of faults," *Journal of ACM*, vol 27, no 2, pp 228-234, 1980

22 K V S Ramarao, "Detection of mutual inconsistency in distributed databases," *Proc 3rd IEEE Int'l Conf on Data Engineering*, pp 405-411, 1987

23 S K Sarin, B T Blaustein, and C W Kaufman, "System architecture for partition-tolerant distributed databases," *IEEE Trans on Computers*, vol C-34, no 12, pp 1158-1163, 1985

24 R Schlicting and F Schneider, "Fail-stop processors An approach to designing fault-tolerant distributed computing systems," *ACM Trans on*

Computer Systems, vol 1, no 3, pp 222-238, 1983

25 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

26 P G Selinger, "Replicated data," in *Distributed Databases*, ed I W Draffen and F Poole, pp 223-231 , Cambridge University Press , Cambridge , 1980

27 D Skeen and M Stonebraker, "A formal model of crash recovery in a distributed system," *IEEE Trans on Software Engineering*, vol SE-9 , no 3, pp 219-228, 1983

28 D Skeen and D Wright, "Increasing availability in partitioned database systems," *Proc 3rd ACM Symp on Principles of Database Systems*, pp 290-299, 1984

29 R H Thomas, "A solution to the concurrency control problem for multiple copy databases," *Proc IEEE Compcon*, pp 56-62, Spring, 1978

30 D D Wright, "On merging partitioned databases," *ACM SIGMOD Record*, vol 13, no 4, pp 6-14, 1983

31 D D Wright, "Managing distributed databases in partitioned networks," Technical Report No 83-572, Dept of Computer Science, Cornell University, 1983

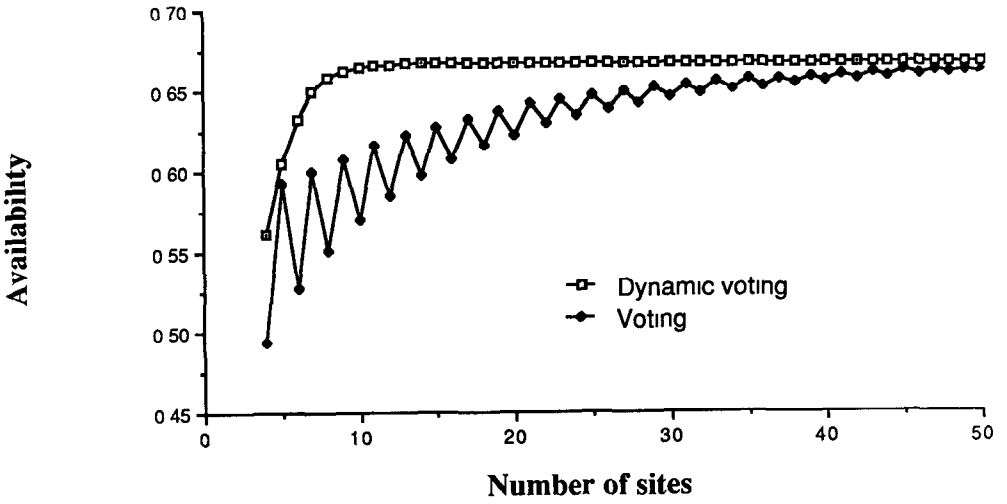

Figure 2. Availability when the repair rate is twice the failure rate.

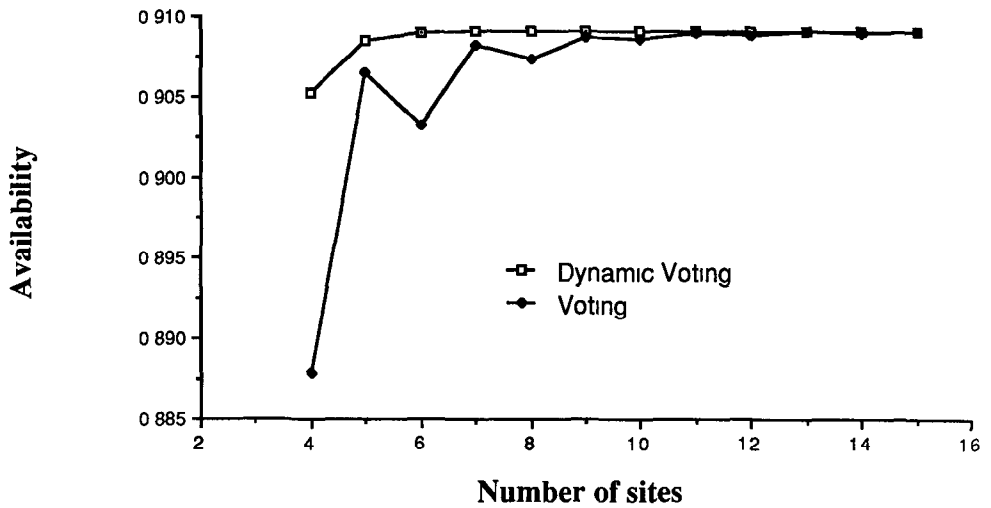

Figure 3. Availability when the repair rate is ten times the failure rate.

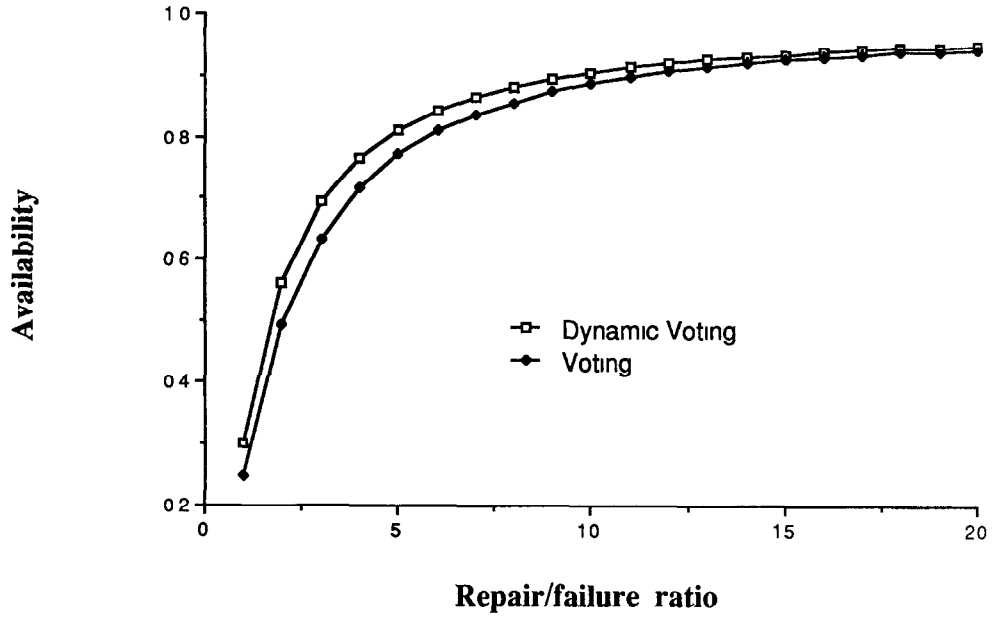

Figure 4. Availability when the network has four sites. The behavior for six or more sites is similar.

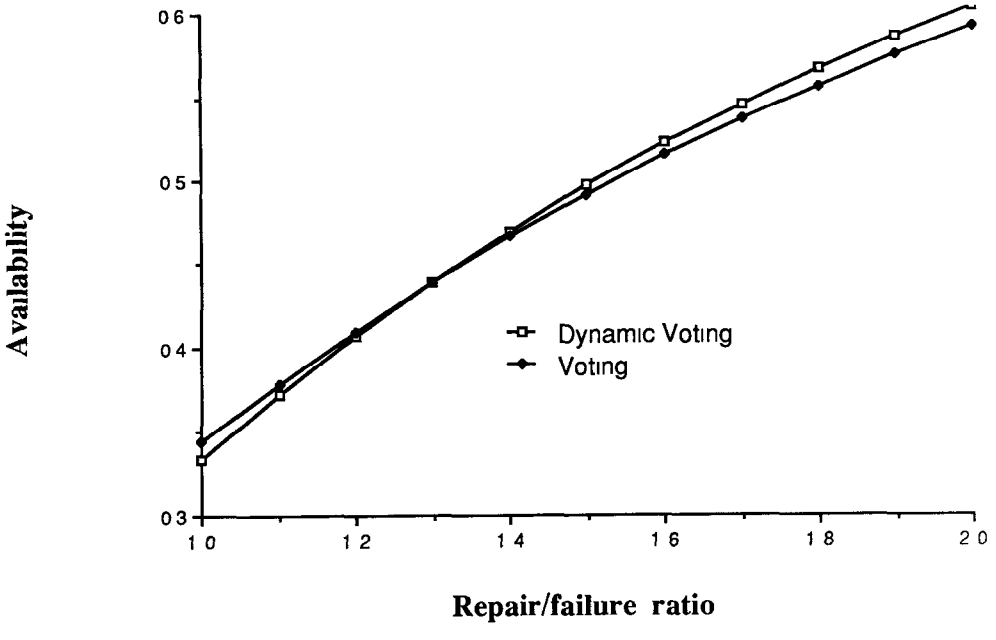

Figure 5. Availability when the network has five sites. This crossing behavior occurs only under these circumstances.