Efficient Dynamic Voting Algorithms

Jehan-François Pâris *Darrell D. E. Long*

Computer Systems Research Group
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093

ABSTRACT

Voting protocols guarantee consistency of replicated data in the presence of any scenario involving non-Byzantine site failures and network partitions. While *Static Majority Consensus Voting* algorithms use static quorums, *Dynamic Voting* algorithms dynamically adjust quorums to changes in the status of the network of sites holding the copies.

We propose in this paper two novel dynamic voting algorithms. One, called *Optimistic Dynamic Voting,* operates on possibly out-of-date information, which greatly increases the efficiency of the algorithm and simplifies its implementation. The other, called *Topological Dynamic Voting,* explicitly takes into account the topology of the network on which the copies reside to increase the availability of the replicated data.

We also compare availabilities of replicated data managed by both algorithms with those of data managed by existing voting protocols using a simulation model with realistic parameters. Optimistic Dynamic Voting is found to perform as well as the best existing voting algorithms while Topological Dynamic Voting performs much better than all other voting algorithms when two or more copies reside in the same non-partitionable group.

Keywords: file consistency, fault-tolerant systems, replicated files, majority consensus voting.

1. INTRODUCTION

Distributed systems that maintain multiple copies of some data objects need a mechanism to ensure the consistency of the objects in the presence of hardware and software malfunctions. Although many consistency algorithms for replicated files have been introduced in recent years, very few studies have been dedicated to the performance of these algorithms, either in terms of incurred message overhead or in terms of the availability and reliability of the replicated files managed by such algorithms. As a result, investigations in the area of consistency algorithms have often been focused on relatively inefficient algorithms such as Majority-Consensus Voting (MCV) [Elli77, Giff79, Elli83, Garc84]. More efficient algorithms such as Available Copy [BeGo84, LoPa87] and Dynamic Voting [DaBu85, BGS86, Jajo87, PaBu86] have received considerably less attention.

This work was supported in part by a grant from the NCR Corporation and the University of California MICRO program.

Dynamic voting algorithms are especially interesting because they provide high reliability and availability while handling network partitions correctly. They do have some limitations, however. Their implementation is complicated by the requirement of instantaneous state information, which is costly in terms of network traffic. They also require a minimum of three copies to be of any practical interest.

Section 2 of this paper introduces optimistic dynamic voting. Section 3 explains how to improve file availability by explicitly taking into account the topology of the network. Section 4 analyzes the performance of both protocols using a discrete event simulation model with realistic parameters. Finally, section 5 has our conclusions.

2. DYNAMIC VOTING

The weakness of majority consensus voting and of all other static voting protocols is that the quorum is fixed—it can not change once the system has begun operation. Because of this, a few failures can render the data inaccessible. *Dynamic Voting* [DaBu85] is a consistency and recovery control algorithm for replicated objects tailored to environments susceptible to site failures and network partitions. This policy adjusts the necessary quorum of physical copies required for an access operation without manual intervention. A group of physical copies, comprised of a majority of the current physical copies that can communicate among themselves, is referred to as the *majority block.*

Dynamic voting is based on the concept of the *connection vector.* The connection vector instantaneously records the state of the network with respect to all sites. Each physical copy of a replicated data object has an associated ensemble of state information consisting of the version number and the partition vector. The *version number* of a physical copy represents the number of successful write operations to the file that are known to the physical copy. The *partition vector* at a site records the version numbers of all sites as they were most recently received by that site.

In its original form dynamic voting allows accesses to proceed so long as a strict majority of the current physical copies are accessible. In situations where the number of current physical copies within a group of mutually communicating sites is equal to the number of current copies not in communication, dynamic voting cannot proceed and declares the replicated file to be inaccessible.

A simple extension proposed by Jajodia [Jajo87], known as *Lexicographic Dynamic Voting,* resolves these ties by applying a total ordering to the sites.

The sites holding copies of the data are given a static linear ordering. Then, when a tie occurs, if the group of communicating sites contains exactly one-half the current physical copies and that group contains the maximum element among the group of current physical copies, that group is declared to be the majority block.

Our experiments [BMP87] have shown that the connection vector is a difficult object to implement. Attempts to approximate it can consume nearly all of the available machine cycles on a moderate sized site. Since Dynamic Voting provides very high availability, but was not practical in its original form, alternative methods for implementing it were sought.

2.1. Our Method

A replicated file will consist of a set of physical copies residing on distinct sites of a local area network. These sites can fail and the messages they exchange can be lost. It will be assumed that sites not operating correctly will immediately stop operations and that all messages delivered to their destinations will be delivered without alterations in the order they were sent. Byzantine failures are expressly excluded.

A physical copy will be said to be current if it has received all write requests to the replicated object. A group of physical copies, comprised of a majority of the current physical copies that can communicate among themselves, will be referred to as a *majority partition.*

Every physical copy of a replicated file will maintain some state information. This information will include a *operation number,* a *version number* and a *partition set.* The *operation number* o_i is a positive integer that is incremented at every successful operation completed by that copy. The *version number* v_i is a positive integer that identifies the last successful write operation on that copy. The partition set represents the set of physical copies that participated in the most recent operation. It will be used by the dynamic voting algorithm to keep track of changes in the composition of majority partitions, to reinsert recovering copies into a majority partition and to regenerate a majority partition for replicated files that are unavailable.

To illustrate these concepts, consider a replicated file consisting of three physical copies located at sites *A*, *B* and *C*. Assuming that all sites and all links are operational, the initial operation numbers o_i and version numbers v_i are 1 and the partition vector P_i are $\{A, B, C\}$ for all three copies.

A	B	C
$o, v{=}1$	$o, v{=}1$	$o, v{=}1$
$P_A{=}\{A, B, C\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, B, C\}$

The initial majority partition consists of all three copies *A*, *B* and *C*. After seven write operations are successfully completed, the state of the replicated file is represented by:

A	B	C
$o, v{=}8$	$o, v{=}8$	$o, v{=}8$
$P_A{=}\{A, B, C\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, B, C\}$

The majority partition still consists of all three copies *A*, *B* and *C*. Suppose now that site *B* fails. Information is exchanged only at access time, so there is no change in the state information:

A	B	C
$o, v{=}8$	$o, v{=}8$	$o, v{=}8$
$P_A{=}\{A, B, C\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, B, C\}$

The partition consisting of sites *A* and *C* contains a majority of the sites included in the previous majority partition. It will therefore become the new majority *partition.* After three more write operations, we have the following situation:

A	B	C
$o, v{=}11$	$o, v{=}8$	$o, v{=}11$
$P_A{=}\{A, C\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, C\}$

Assume that the link between *A* and *C* fails. Again, no information is exchanged as a result of the network failure. We have now a *partition* of the network into the two disjoint subsets $\{A\}$ and $\{C\}$. The file will then be in the following configuration:

A	B	C
$o, v{=}11$	$o, v{=}8$	$o, v{=}11$
$P_A{=}\{A, C\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, C\}$

We have now exactly one site of the previous majority partition on each side of the partition. Such situations where the number of current physical copies within a group of mutually communicating sites is equal to the number of current copies not in communication, are not infrequent. Our protocol accommodates these situations by introducing a tie breaking rule, as in Lexicographic Dynamic Voting [Jajo87].

Suppose the sites are ordered so that $A > B > C$. Then, after the link between *A* and *C* failed, site *A*, by itself, constitutes the new majority partition. Site *A* can determine this by consulting its partition set and operation number. It knows that it cannot communicate with site *C* and that the previous majority partition consists of the subset $\{A, C\}$. Since *A* ranks higher than *C*, the group containing *A* is the majority partition. By the same reasoning, site *C* determines that it is not the majority partition. Four more write operations would leave the file in the state:

A	B	C
$o, v{=}15$	$o, v{=}8$	$o, v{=}11$
$P_A{=}\{A\}$	$P_B{=}\{A, B, C\}$	$P_C{=}\{A, C\}$

The basis of our protocol is the algorithm for detecting whether the access request is originating within the majority partition. Since there can be only one majority partition, mutual exclusion is guaranteed and consistency is preserved. There are three sets of information that must be maintained: the partition set, P_i, which represents the set of sites which participated in the last successful operation, a operation number, o_i, and a version number, v_i, attached to each site.

Algorithm 1. Algorithm for deciding whether the current partition is the majority partition.

1. Find the set of all sites communicating with the requesting site, call it *R*.

2. Request from each site $i \in R$ its partition set P_i, its operation number o_i, and its version number v_i.
3. Let $Q \subset R$ be the set of all sites with operation numbers that match that of the site with the highest operation number.
4. Let P_m be the partition set of any site in Q.
5. If the cardinality of Q is greater than one half the cardinality of P_m, or is exactly one half and contains the maximum element of P_m then the current partition is the majority partition.

A protocol similar to this was developed independently by Jajodia and Mutchler [JaMu87]. Their protocol used integer values to represent the previous quorum instead of the partition sets that are used here. It requires less storage to implement simple Dynamic Voting, but it cannot accommodate Lexicographic Dynamic Voting as it does not keep track of the identity of the maximum element of the partition set. Our protocol easily includes the lexicographic enhancement, and can be expanded to take topological information into account.

The algorithm for performing a read operation is simple. It first ascertains whether the current partition is the majority partition. This is done in the same way for all of the algorithms presented. A message is broadcast to all sites; those that send replies are considered to be in the current partition. From each of these sites a request is made for their partition set, operation number and version number. The set of current sites is found by computing the maximum operation number of all of the sites. It is this set of sites that will participate in the operation. This set of sites holding up-to-date copies is the *quorum set*. If the quorum set represents a majority of the previous quorum, represented by P_i, then the access request is granted. If there is a tie, that is there are exactly one half of the previous quorum, then a total ordering on the set of sites is used to decide if access will be granted.

```
procedure READ(f : file)
begin
   let U be the set of all sites participating in the replication
   $\langle R, \mathbf{o}, \mathbf{v}, \mathbf{P} \rangle \leftarrow$ START$(U, f)$
   $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
   $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
   choose any $m \in Q$
   if $(|Q| > \frac{|P_m|}{2}) \vee (|Q| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
      perform the read
      COMMIT$(S, o_m+1, v_m, S)$
   else
      ABORT$(R)$
   fi
end READ
```

Figure 1: Read Algorithm

Once it has been ascertained that the current partition is the majority partition, then access can continue. The read operation is performed and the operation number is incremented and sent along with the set of current sites to each of current sites to serve as their new partition sets. This last action serves to modify the quorum required for access requests to be granted in the future.

There are several items in Figure 1 that require explanation. The START operation begins the operation and returns R which is the set of reachable sites and three arrays: **o**,**v** and **P** which are the operation numbers, version numbers and partition sets respectively. The COMMIT operation completes the operation and transmits the new consistency control information to all up-to-date copies.

The operation numbers are introduced to speed the recovery of a site, supplementing the information provided by the version numbers. There is a choice between the extra maintenance of the operation number and increased recovery time. If the version number is incremented on each read operation, then recovery will be forced to occur when it is unnecessary. If version numbers alone were used, and were not incremented for each read operation, then there would be cases where two majority partitions could exist. This is because a read operation does not change the state of the data, and so it should not change the version number of the data. Instead we chose to implement the partition set as a data object with a loose consistency constraint.

The algorithm for writing is similar to the algorithm for reading. Again it is ascertained if the current partition is the majority partition. If this is successful, proceed as in the algorithm for reading. The write operation is performed. The operation number and the version number are incremented and sent along with the set of current sites to all of the current sites to serve as their new partition sets.

```
procedure WRITE(f : file)
begin
   let U be the set of all sites participating in the replication
   $\langle R, \mathbf{o}, \mathbf{v}, \mathbf{P} \rangle \leftarrow$ START$(U, f)$
   $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
   $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
   choose any $m \in Q$
   if $(|Q| > \frac{|P_m|}{2}) \vee (|Q| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
      perform the write
      COMMIT$(S, o_m+1, v_m+1, S)$
   else
      ABORT$(R)$
   fi
end WRITE
```

Figure 2: Write Algorithm

The recovery algorithm begins as do the other algorithms, ascertaining whether the current partition is the majority partition. If the recovering site is able to communicate with the majority partition, then it determines whether the copy at that site is up-to-date. If it is not, then it must be copied from any of the sites in the quorum set. The recovering site then sends the union of the set of current sites and itself to all of the current sites, including itself, to serve as their new partition sets.

```
procedure RECOVER($f$ : file)
begin
  repeat
    let $I$ be the recovering site
    let $U$ be the set of all sites participating in the replication
    $\langle R, \mathbf{o}, \mathbf{v}, \mathbf{P} \rangle \leftarrow$ START($U$, $f$)
    $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
    $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
    choose any $m \in Q$
    if $(|Q| > \frac{|P_m|}{2}) \vee (|Q| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
      if $v_I < v_m$ then
        copy the file from site $m$
      fi
      COMMIT($S \cup \{I\}, o_m+1, v_m, S \cup \{I\}$)
    else
      ABORT($R$)
    fi
  until successful
end RECOVER
```

Figure 3: Recovery Algorithm

The advantage of the algorithms proposed is that they much the same message traffic overhead as majority consensus voting, and that their implementation is simple. There are no assumptions made about the state of the network that which cannot be verified by examining the partition sets and version numbers.

This method is simple and efficient. It provides consistency control, and more generally, mutual exclusion. The availability of data and the reliability of access afforded by this method is superior to majority consensus voting for only a small increase in network traffic.

3. TOPOLOGICAL DYNAMIC VOTING

Existing dynamic voting algorithms require that any new majority block contains a majority of the sites that were in the previous majority partition. As a result, they guarantee file consistency and enforce mutual exclusion in the presence of any combination of non-Byzantine site failures and network partitions. This is not always necessary. Several classes of local area networks, including unsegmented carrier-sense networks and token rings, are immune to partial failures that can create a network partition. Special consistency algorithms have been developed for such environments. These *Available Copy* algorithms [BeGo84, LoPa87, CLP87] are simpler to implement than dynamic voting protocols and led to much higher file availabilities.

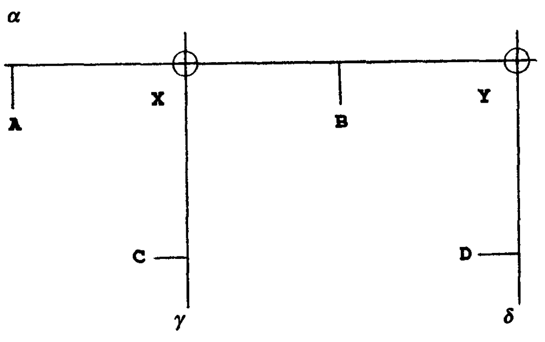

Figure 4

Larger local-area networks often consist of several carrier-sense networks or token rings linked by selective repeaters or gateway hosts. Since repeaters and gateways can fail without causing a total network failure, such networks can be partitioned. The key difference with conventional point-to-point networks is that sites that are on the same carrier-sense network or token ring will never be separated by a partition.

Consider for instance a replicated file with four copies A, B, C and D such that A and B are on the same unsegmented carrier-sense network α while C and D are each on their own segments γ and δ. Let X be the repeater linking α and γ and Y the one linking α and δ. The repeaters X and Y are the only possible partition points and the only possible partitions are $\{\{A, B, C\}, \{D\}\}$, $\{\{A, B, D\}, \{C\}\}$ and $\{\{A, B\}, \{C\}, \{D\}\}$.

Assume now that the file is in the state

A	B	C	D
$o, v=15$	$o, v=15$	$o, v=11$	$o, v=8$
$P_A=\{A, B\}$	$P_B=\{A, B\}$	$P_C=\{A, B, C\}$	$P_D=\{A, B, C, D\}$

where the majority block consists of sites A and B. Assume now that site A fails. Under Lexicographic Dynamic Voting, site B cannot become the majority partition since site A precedes site B in the lexicographic ordering of sites and the file should become unavailable. Doing otherwise was always assumed to be undesirable as it would have lead to situations where sites A and B would be in two disjoint majority blocks. The situation is different here. When site B obtains no answer from site A, it knows that it can result only from a total failure of the network segment α or from a failure of site A. Should α be operational, B knows that A must be unavailable and can safely become the majority block.

More generally, if m copies of a replicated object reside on sites that are on the same carrier-sense segment or token ring, a site attempting to build a new majority block needs to communicate with only one of the m copies to ascertain that the $m-1$ other copies are not likely to be involved in any incompatible attempt to build a majority block. We would like to take advantage of this observation to increase the availability of replicated objects that have more than one copy on the same carrier-sense segment or token ring. We will therefore to modify the dynamic voting algorithm and allow available sites belonging to the previous majority block to carry the votes of unavailable sites belonging to the previous majority block and attached to the same carrier-sense segment or token ring.

Assume that the copies reside on distinct sites of a local area network consisting of p indivisible segments α_1, ..., α_p. If P_k denotes the partition set for site k and if R_k is the set of sites communicating with site k, site k will be able to gather the votes of all sites $j \in P_k$ that are on the same segment as an active site $i \in P_k \cap R_k$.

```
procedure READ(f: file)
begin
  let U be the set of all sites participating in the replication
  $(R, \mathbf{o}, \mathbf{v}, \mathbf{P}) \leftarrow$ START(U, f)
  $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
  $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
  choose any $m \in Q$
  $T \leftarrow \{r \in P_m : \exists \alpha_k, \exists s \in P_m \cup R \; r, s \in \alpha_k\}$
  if $(|T| > \frac{|P_m|}{2}) \vee (|T| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
    perform the read
    COMMIT$(S, o_m+1, v_m, S)$
  else
    ABORT$(R)$
  fi
end READ
```

Figure 5: Read Algorithm

When all the sites are on the same segment, the modified *topological* algorithm degenerates into an available copy protocol as a quorum is guaranteed as long as one copy remains available.

Mutual consistency is guaranteed as long as the same unavailable site belonging to the previous majority block cannot be concurrently claimed by two disjoint attempts to build rival majority blocks. Gateway hosts holding copies cause a special problem as they belong at the same time to more than one segment and could therefore be claimed by rival majority blocks. The simplest solution to this problem is to disallow membership to multiple segments and to assume that every gateway host belongs to only one network segment.

Topological Dynamic Voting, as this algorithm is called, can be easily combined with Optimistic Dynamic Voting to obtain a more efficient consistency algorithm. Figures 5, 6 and 7 detail the read, write and recovery algorithms for *Optimistic Topological Dynamic Voting.* These three algorithms are very similar to the ones presented in the previous section, showing the versatility of this partition set approach.

```
procedure WRITE(f: file)
begin
  let U be the set of all sites participating in the replication
  $(R, \mathbf{o}, \mathbf{v}, \mathbf{P}) \leftarrow$ START(U, f)
  $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
  $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
  choose any $m \in Q$
  $T \leftarrow \{r \in P_m : \exists \alpha_k, \exists s \in P_m \cup R \; r, s \in \alpha_k\}$
  if $(|T| > \frac{|P_m|}{2}) \vee (|T| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
    perform the write
    COMMIT$(S, o_m+1, v_m+1, S)$
  else
    ABORT$(R)$
  fi
end WRITE
```

Figure 6: Write Algorithm

```
procedure RECOVER(f: file)
begin
  repeat
    let l be the recovering site
    let U be the set of all sites participating in the replication
    $(R, \mathbf{o}, \mathbf{v}, \mathbf{P}) \leftarrow$ START(U, f)
    $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{o_s\}\}$
    $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{v_s\}\}$
    choose any $m \in Q$
    $T \leftarrow \{r \in P_m : \exists \alpha_k, \exists s \in P_m \cup R \; r, s \in \alpha_k\}$
    if $(|T| > \frac{|P_m|}{2}) \vee (|T| = \frac{|P_m|}{2} \wedge \max(P_m) \in Q)$ then
      if $v_l < v_m$ then
        copy the file from site m
      fi
      COMMIT$(S \cup \{l\}, o_m+1, v_m, S \cup \{l\})$
    else
      ABORT$(R)$
    fi
  until successful
end RECOVER
```

Figure 7: Recovery Algorithm

Since sites can only claim the votes of unavailable copies residing on the same network segment, the only additional information required to implement Topological Dynamic Voting is a list of sites belonging to the same segment and holding copies of the same object. No global information about the network topology has to be stored anywhere, which makes the algorithm well suited to local area networks with large numbers of hosts.

Table 1: Site Characteristics

Site	Name	Mean Time To Fail (days)	Hardware Failures (%)	Restart Time (min.)	Hardware Repair Time	
					Constant Part (hours)	Exp. Part (hours)
1	csvax	36.5	10	20.0	0.0	2
2	beowulf	10	10	15	4	24
3	grendel	365	90	10	0	2
4	wizard	50	50	15	168	168
5	amos	365	90	10	0	2
6	gremlin	50	50	15	168	168
7	rip	50	50	15	168	168
8	mangle	50	50	15	168	168

Note: Sites 1, 3 and 5 are unavailable for 3 hours every 90 days for preventive maintenance.

4. SIMULATION ANALYSIS

Stochastic process models have been widely used to evaluate consistency protocols. Unfortunately, many difficulties prevent relying solely upon stochastic process modeling: an exponential distribution of repair times is unrealistic, but using other distributions result in intractable problems in most cases; the problem of modeling network partitions and site failures simultaneously is intractable for all but the most basic cases [NKT87]; and algebraic expressions for file reliability are difficult to obtain, even for the simplest algorithms and site configurations. For these and other reasons simulation has been chosen as a method to evaluate the performance of our consistency policies.

An existing network consisting of eight sites and three carrier-sense segments linked by gateways is used as a model. As seen on Figure 8, five of the eight sites are connected on the main carrier-sense segment. One of these sites is the gateway to the second segment, to which the sixth site is also connected; another of the five sites is the gateway to the third segment, to which the seventh and eighth sites are also connected. Each site, including the two gateways, is able to store and maintain copies of a file.

Site failure and repair data are summarized in Table 1. Individual values for mean time to failure, percentage of hardware faults, repair times for hardware and software failures and preventive maintenance schedules were chosen to reflect as accurately as possible the true behavior of the sites modeled. Exponential failure distributions were chosen for all eight sites and repair times were modeled by a constant term plus an exponentially distributed term. Hardware failures normally result in a human intervention and often require a service call. Hardware repair times were modeled by a constant term representing the minimum service time plus an exponentially distributed term representing the actual repair process. Since software failures only require a system restart, constant recovery times are a ssumed. The three carrier-sense segments were assumed not to fail; however, gateways may fail and cause network partitions. Message delivery is guaranteed to all active sites in the current partition when a file access request is made. Local experience justifies these assumptions.

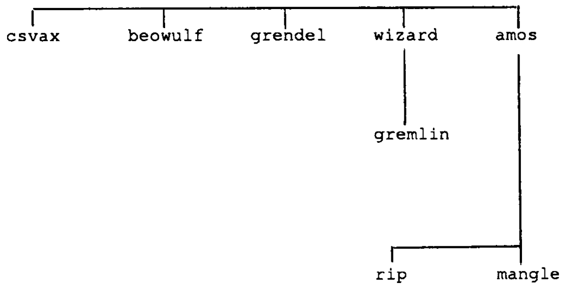

Figure 8: Network Topology

Access to the replicated file is modeled as a single user that can access any of the eight sites. The access requests are granted or refused based solely on the current state of the sites containing copies and the algorithm's capability to guarantee file consistency. Batch-means analysis was used to compute 95% confidence intervals for all performance indices. All sites were operating at the start of the simulation; the time-to-steady-state interval was taken to be 360 days. A more detailed description of our simulation model can be found in [PLG88].

Eight configurations were considered in our study. The first four consist of three copies. Configuration A is comprised of copies on sites 1, 2 and 4, which allows for no partitions. Configuration B consists of copies on sites 1, 2 and 6 with a single partition point at site 4. Configuration C has copies on sites 1, 6 and 8 with one partition point at site 4 and another at site 5.

Table 2: Replicated File Unavailabilities

Sites	Consistency Policy					
	MCV	DV	LDV	ODV	TDV	OTDV
A: 1, 2, 4	0.002130	0.004348	0.000668	0.000849	0.000015	0.000013
B: 1, 2, 6	0.003871	0.008281	0.001214	0.001432	0.000109	0.000066
C: 1, 6, 8	0.031127	0.056428	0.001707	0.003492	0.001707	0.003492
D: 6, 7, 8	0.069342	0.117683	0.053592	0.053357	0.034490	0.031548
E: 1, 2, 3, 4	0.000608	0.000018	0.000012	0.000084	0.000000	0.000000
F: 1, 2, 4, 6	0.002761	0.108034	0.002154	0.000947	0.000018	0.000004
G: 1, 2, 6, 8	0.002027	0.001510	0.000151	0.000339	0.000041	0.000036
H: 1, 2, 7, 8	0.001408	0.004275	0.000171	0.000218	0.000020	0.000043

Configuration D is comprised of copies on sites 6, 7 and 8; either site 4 or 5 can cause a partition. The other four configurations consist of four copies distributed as follows. Configuration E has copies on sites 1, 2, 3 and 4, which allows for no partitions. Configuration F is comprised of copies on sites 1, 2, 4 and 6 with a partition point at site 4. Configuration G has copies on sites 1, 2, 6 and 8 with partition points at sites 4 and 5. Finally, configuration H consists of two pairs of copies at sites 1 and 2 and sites 7 and 8 separated by a single partition point at site 5.

Table 2 summarizes the unavailabilities of replicated files for all eight configurations and all five consistency policies. Unavailabilities are measured and displayed since they indicate more clearly the differences among the policies.

The first finding was that Dynamic Voting (DV) performed worse than Majority Consensus Voting (MCV) for three copies. This is not surprising since the same conclusion had already been reached by Paris and Burkhard using Markov chains [PaBu86]. Dynamic Voting requires at least two copies from the previous majority block to form a new majority block and is more restrictive than Majority Consensus Voting which only requires two copies in this case.

For four copies, it was found that Dynamic Voting performed much better than Majority Consensus Voting in configurations E and G where partitions are either not allowed or not likely to cause ties. The situation was different for configurations F and H where the failure of a single site could result in a tie. For instance, the failure of site 5 in configuration H will normally leave the system with two operational groups of the same size. The unavailability of the configuration is not essentially different from the unavailability of a replicated file consisting of a single copy at site 5. Lexicographic Dynamic Voting (LDV), which resolves ties, outperforms Majority Consensus Voting and Dynamic Voting in all cases.

The performance of Optimistic Dynamic Voting (ODV) was measured assuming one file access per day and expected to obtain unavailabilities intermediary between those of Majority Consensus Voting, which never updates quorums, and those of Lexicographic Dynamic Voting, where the quorums instantaneously reflect any change in the network status. For three of the eight observed configurations, it was found instead that Optimistic Dynamic Voting performed better than Lexicographic Dynamic Voting. This phenomenon is the most apparent for configuration F. This configuration has site 4 as its partition point. As table 1 indicates, sites 1 and 2 recover much faster from hardware failures than site 4. It is therefore better not to update the quorum when site 1 or 2 fails since this does not protect the file against a failure of site 4 but instead delays file recovery until site 4 is repaired. This is exactly what Optimistic Dynamic Voting does when the replicated file is accessed once a day.

As expected, Topological Dynamic Voting (TDV) and Optimistic Topological Dynamic Voting (OTDV) performed much better than all other algorithms when two or more sites were on the same carrier-sense segment. The lowest unavailability figures were obtained for configuration E, which has the four sites with copies on the same Ethernet. The simulation indicated that a replicated object with a similar copy configuration could remain continuously available for more than three hundred years provided no catastrophic failure and no network failure ever occurred during that time interval. Conversely a replicated object with every copy dispersed on a different segment, as it is the case for configuration C, has the same unavailability under Topological and Lexicographic Dynamic Voting.

The last algorithm studied was Optimistic Topological Dynamic Voting (OTDV). As its name indicates, this algorithm is an optimistic voting algorithm taking into account the local topology of the network. Except for configuration C, it was found to perform comparable to or better than Topological Dynamic Voting.

Also measured for every algorithm and every configuration was the mean length of time that a replicated file was unavailable. These figures are summarized in table 3.

Table 3: Mean Duration of Unavailable Periods

Sites	Consistency Policy					
	MCV	DV	LDV	ODV	TDV	OTDV
A: 1, 2, 4	0.101968	0.210651	0.077353	0.084141	0.10764	0.05115
B: 1, 2, 6	0.101059	0.217369	0.078867	0.084387	0.08650	0.05337
C: 1, 6, 8	0.944336	1.868895	0.085960	0.173151	0.085960	0.173151
D: 6, 7, 8	3.000469	5.850864	7.443789	6.293645	7.428305	7.445393
E: 1, 2, 3, 4	0.071134	0.06363	0.08102	0.05417	-	-
F: 1, 2, 4, 6	0.102001	5.962853	0.275006	0.101756	0.05556	0.02252
G: 1, 2, 6, 8	0.084714	0.297879	0.07787	0.073773	0.12407	0.04149
H: 1, 2, 7, 8	0.078933	0.142206	0.135054	0.060009	0.103171	0.051964

5. CONCLUSIONS

In this paper two novel dynamic voting algorithms have been presented. The first one, called *Optimistic Dynamic Voting,* operates on possibly out-of-date information, which greatly increases the efficiency of the algorithm and simplifies its implementation. The other, called *Topological Dynamic Voting,* takes into account the topology of the network on which the copies reside to increase the availability of the replicated data.

A realistic simulation model was built to compare availabilities of replicated data managed by both algorithms with those of data managed by existing voting protocols. It was found that Optimistic Dynamic Voting performed as well as the best existing voting algorithms while Topological Dynamic Voting performed much better than all other voting algorithms when two or more copy reside in the same non-partitionable group.

It is expected that these two new algorithms will greatly contribute to a wider usage of dynamic voting for the management of replicated data. Optimistic Dynamic Voting and Optimistic Topological Dynamic Voting require much less message traffic than their non-optimistic counterparts while achieving comparable, and in some case better, data availabilities. Topological voting greatly improves the availability of replicated objects with two or more copies in the same non-partitionable group without any significant increase in the complexity of the algorithm. More studies are still needed to investigate the inclusion of witness copies [Pari86] and to analyze weight assignments.

Acknowledgements

We wish to thank Walter Burkhard and all the other members of the Gemini group for their help and their encouragement. This work has been done with the aid of MACSYMA, a large symbolic manipulation program developed at the Massachusetts Institute of Technology Laboratory for Computer Science. MACSYMA is a trademark of Symbolics, Inc. We are also grateful to L. F. Cabrera and Sam Toueg for their constructive comments.

References

[BGS86] D. Barbara, H. Garcia-Molina and A. Spauster, "Policies for Dynamic Vote Reassignment," *Proc. 6th Int. Conf. on Distributed Computing Systems,* (May 1986), pp. 37-44.

[BeGo84] P. A. Bernstein and N. Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases." *ACM Trans. on Database Systems,* Vol. 9, No. 4 (Dec. 1984), 596-615.

[BMP87] W. A. Burkhard, B. E. Martin and J.-F. Paris, "The *Gemini* Fault-Tolerant File System: the Management of Replicated Files." *Proc. 3rd Int. Conf. on Data Engineering,* (February 1987), pp. 441-448.

[CLP87] J. L. Carroll, D. Long and J.-F. Paris, "Block-Level Consistency of Replicated Files." *Proc. 7th Int. Conf. on Distributed Computing Systems,* (Sept. 1987), pp. 146-153.

[DaBu85] D. Davcev and W.A. Burkhard, "Consistency and Recovery Control for Replicated Files." *Proc. 10th ACM Symp. on Operating System Principles,* (1985) pp. 87-96.

[Elli77] C. A. Ellis, "Consistency and Correctness of Duplicate Database Systems." *Operating Systems Review,* 11, 1977.

[ElFl83] C. S. Ellis and R. A. Floyd, "The Roe File Systems." *Proc. 3rd Symp. on Reliability in Distributed Software and Database Systems,* (Oct. 1983), pp. 175-181.

[Garc82] H. Garcia-Molina, "Elections in a Distributed Computer Systems." *IEEE Trans. on Computers,* Vol. C-31, No. 1, (Jan. 1982), 48-59.

[Giff79] D. K. Gifford, "Weighted Voting for Replicated Data." *Proc. 7th ACM Symp. on Operating System Principles,* 1979, 150-161.
S. Jajodia, "Managing Replicated Files in Partitioned Distributed Database Systems." *Proc. 3rd Int. Conf. on Data Engineering,* (February 1987), pp. 412-418.

[JaMu87] S. Jajodia and D. Mutchler, "Dynamic Voting." *Proc. ACM SIGMOD 1987 Annual Conf.,* (May 1987), pp. 227-238.

[LoPa87] D. Long and J.-F. Paris, "On Improving the Availability of Replicated Files." *Proc. 6th Symp. on Reliability in Distributed Software and Database Systems,* (March 1987), pp. 77-83.

[NKT87] Nicola, V. F., V. G. Kulkarni and K. S. Trivedi, "Queueing Analysis of Fault-Tolerant Computer Systems," *IEEE Trans. Software Engineering,* Vol. SE-13, No. 3 (March 1987), 363-375.

[PaBu86] J.-F. Paris and W.A. Burkhard, "On the Availability of Dynamic Voting Schemes." Technical Report, Department of CSE, University of California, San Diego.

[Pari86] J.-F. Paris, "Voting with Witnesses: A Consistency Scheme for Replicated Files." *Proc. 6th Int. Conf. on Distributed Computing Systems,* (May 1986), pp. 606-612.

[PLG88] J.-F. Paris, D. Long and A. Glockner, "A Realistic Evaluation of Consistency Algorithms for Replicated Files." *Proc. 21st Annual Simulation Symp.,* (March 1988) to appear.

[ScSc83] R. D. Schlichting, and F. B. Schneider, "Fail Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems." *ACM Trans. on Computer Systems,* 1983, 222-238.