Title

Voting without version numbers

Authors

Long, DDE

Paris, J-F

Publication Date

1997

DOI

10.1109/pccc.1997.581496

VOTING WITHOUT VERSION NUMBERS

Darrell D. E. Long

Department of Computer Science
IBM Almaden Research Center
San Jose, CA 95120

darrell@almaden.ibm.com

Jehan-François Pâris

Department of Computer Science
University of Houston
Houston, TX 77204-3475

paris@cs.uh.edu

ABSTRACT

Voting protocols are widely used to provide mutual exclusion in distributed systems and to guarantee the consistency of replicated data in the presence of network partitions. Unfortunately, the most efficient voting protocols require fairly complex metadata to assert which replicas are up-to-date and to denote the replicas that belong to that set.

We present a much simpler technique that does not require *version numbers* and maintains only $n+\log(n)$ bits of state per replica. We show, under standard Markovian assumptions, that a static voting protocol using our method provides nearly the same data availability as a static voting protocol using version numbers. We also describe a dynamic voting protocol using our method that provides the same data availability as a dynamic voting protocol using much more complex metadata.

I. INTRODUCTION

Voting protocols have been extensively used to provide mutual exclusion in distributed systems and to guarantee the consistency of replicated data in the presence of network partitions. They owe this distinction to their simplicity and their robustness. In their simplest form, voting protocols assume that the correct state of a replicated object is the state of the majority of its replicas. Ascertaining the state of a replicated object requires collecting a *quorum* of the replicas. Should this be prevented by too many site failures, the replicated object is considered to be inaccessible.

Majority Consensus Voting (MCV) and Weighted Voting (WV) [5] are both called *static* protocols because the required quorums of replicas and the number of votes assigned to each replica are fixed. *Dynamic* protocols that adjusts quorums, such as dynamic voting and its variants [4, 8, 10], or modify the number of votes assigned to each replica [2], can minimize the impact of site failures and increase availability.

The first author is Associate Professor of Computer Science at the University of California, Santa Cruz.

The *Dynamic voting* (DV) protocol [4] instantly adjusts quorums to reflect changes in the state of the network of sites holding the replicas. The DV protocol requires each site to maintain in real time a *connection vector* recording the state of the network. Since the dynamic voting protocol does not assign weights to replicas and does not include a tie-breaking rule, a majority block must always contain at least two replicas. A simple extension, known as *Dynamic-linear voting* (DLV) [8], resolves these ties by applying a total ordering to the sites.

A common feature of all replication control protocols is the use of *metadata* to record the states of the replicas. These metadata nearly always include a *version number*, which is an integer that is incremented each time the replicated data are modified. All dynamic voting protocols also require each replica to keep track of the identities of the replicas it believes to be operational. This information is kept in a metadata structure, variously called a *connection vector*, a *partition vector* or a *majority block*.

Despite the important role played by these metadata, the problem of finding the most efficient metadata organization for a given replication control policy has not received the attention that it deserves. The results of this neglect have been replication control protocols with bloated metadata and complex procedures for ascertaining which replicas are up-to-date.

We present new implementations of the majority consensus voting and the dynamic-linear dynamic protocols that do not require version numbers. Instead our protocols maintain for each replica a *cohort set* that is updated any time a failure is detected or a replica residing on a site that failed is repaired. By requiring that all changes in the cohort set involve all sites in the new cohort set, we guarantee that all replicas sharing the same cohort set are identical and remove the need for maintaining version numbers. As a result, our protocols require only $n + \log(n)$ bits of metadata per replica, that is n bits for storing the cohort set and $\log(n)$ bits for storing the identity of the replica.

II. THE CASE FOR COHORT SETS

All extant replication control protocols incorporate some form of *version numbers*, that is, integers that are incremented every time the replicated data are modified. In most cases, these version numbers are used to distinguish outdated replicas from replicas that are up-to-date. Version numbers offer several disadvantages.

First, they take more space than other metadata. While metadata do not constitute a significant storage cost when the unit of replication is a file system or a data base, they pose a significant cost when the unit of replication is a single file, a data base table, or a group of disk sectors. As a result, it is desirable to allocate as few bits as necessary to version numbers.

Second, any scheme that increments version numbers monotonically will eventually overflow any fixed size integer. We can delay this occurrence by resetting version numbers to zero every time all the replicas return to a consistent state but, even so, there is a danger of overflow. To see this, consider the classical *Gambler's ruin* problem [6], which shows that any compulsive gambler will eventually have a run of bad luck that will ruin him. The same holds for version numbers: the replicated data will eventually encounter a sequence of failures and repairs that will overflow any finite integer. This can be made arbitrarily unlikely by increasing the number of bits allocated to the version numbers but cannot be totally eliminated.

Third, version numbers must be incremented every time the replicated data are updated even though the states of the replicas have not changed. A better solution would only require to update the metadata whenever a replica fails or recovers.

Finally, version numbers do not suffice to implement dynamic voting protocols, among which the dynamic voting protocol and its variants [4, 8], which all require additional metadata.

We propose to replace version numbers by *cohort sets*, that is records of all replicas that participated in the last write operation.

A. Static voting with cohort sets

We will consider replicated data objects consisting of n replicas located at different sites of a network and managed by a static voting protocol. The protocol may, or may not have, different quorums for read and write operations as well as different weights, including none, for each replica. We will only assume that:

1. all write quorums Q_W are high enough to disallow simultaneous writes on disjoint subsets of replicas, and
2. all read quorums are high enough to force each read quorum Q_R to have a non-empty intersection with any write quorum Q_W.

Mutual consistency among the replicas will be guaranteed through site failures and network partitions as long as (a) messages between sites are delivered intact in the order they were placed on the network or not at all, and (b) sites that fail immediately stop operation. Byzantine behaviors are specifically excluded.

We will first give a formal definition of cohort sets then explain the procedure to be used to find the current replica(s) in a read quorum and show when and why this procedure fails.

Definition 1 *The cohort set for a replica represents the set of replicas that were current after the last write or replica recovery in which the replica participated.*

Axiom 1 *The cohort set C_r of a replica r always includes that replica.*

This requires cohort sets to be updated (a) whenever the replica participates in a write and (b) whenever a replica that became unavailable recovers. After any write operation, the cohort sets of all replicas that participated in the write operation must be identical and contain exactly these replicas. Similarly, after any replica recovery, the cohort sets of all replicas that participated in the recovery must be identical.

By requiring that all changes in the cohort set involve all sites in the new cohort set, we guarantee that all replicas sharing the same cohort set are identical and remove the need for maintaining version numbers. If n denotes the number of replicas, our protocol will thus require only $n + \log(n)$ bits of metadata per replica, that is n bits for storing the cohort set and $\log(n)$ bits for storing the identity of the replica.

Lemma 1 *Consider a replicated data object with n replicas, m of which are accessible. If the m replicas have identical cohort sets and constitute a read quorum Q_R, they are all up-to-date.*

Proof:

1. Since the m replicas constitute a read quorum Q_R, each and every write quorum Q_W must contain at least one replica of Q_R. Hence at least one of them participated in the last write to the object.
2. Since the m replicas have identical cohort sets, they all participated in the same last write. By induction on each and every write or replica recovery operation, they are identical. Hence, if one of them is up-to-date, they are all.

■

The situation is somewhat more complicated if the m replicas do not have identical cohort sets and is better explained with an example. Consider, for instance,

a replicated data object consisting of three replicas, a, b and c and assume that the read and write quorums are two replicas. Let C_a, C_b and C_c be the respective cohort sets of a, b and c. If the three replicas are up-to-date, their cohort sets will be:

$$C_a = \{a, b, c\} \quad C_b = \{a, b, c\} \quad C_c = \{a, b, c\}$$

Assume now that c becomes unavailable because the site holding c fails or becomes unreachable as the result of a network partition. The cohort sets of a and b will not be updated until a write occurs. After that write the three cohort sets will be:

$$C_a = \{a, b\} \quad C_b = \{a, b\} \quad (C_c = \{a, b, c\})$$

Assume now that a becomes unavailable. Since b is now the only accessible replica, the replicated object will remain unavailable until either a or c recovers. If c recovers, then the two operational replicas will have different cohort sets:

$$(C_a = \{a, b\}) \quad C_b = \{a, b\} \quad C_c = \{a, b, c\}$$

The protocol can assert that replica b is up-to-date because its cohort set C_b is a subset of the cohort set of replica c.

Theorem 1 *Consider a replicated data object with n, m of which are operational. If the m replicas constitute a read quorum Q_R, then all replicas $r \in Q_R$ such that $C_r \subseteq C_s$ for all $s \in Q_R$ are up-to-date.*

Proof:

1. Since the m replicas constitute a read quorum Q_R, each and every write quorum Q_W must contain at least one replica of Q_R. Hence Q_R contains at least one replica that participated in the last write.

2. If two replicas r and s have the same cohort sets C_r and C_s, they are in the same state because otherwise the cohort set of one of them, say s, would contain a replica that was not current after the last write or replica recovery in which s participated.

3. Consider now a replica $r \in Q_R$ such that $C_r \subset C_s$ for all $s \in Q_R$. Should r not be current, there should be in Q_R at least one current replica x such that $C_r \subset C_x$. But then C_x would contain r even though r was supposed not to be current.

4. Hence all replicas $r \in Q_R$ such that $C_r \subseteq C_s$ for all $s \in Q_R$ are up-to-date.

■

After replica c is brought up-to-date, the new cohort sets are:

$$(C_a = \{a, b\}) \quad C_b = \{b, c\} \quad C_c = \{b, c\}$$

There are a few cases where the protocol fails to find the current replica(s) in Q_R because Q_R does not contain any replica s such that its cohort set C_s is a subset of all the cohort sets of the other replicas of Q_R. Consider the following scenario:

1. After replica c is brought up-to-date, replicas b and c receive a new update and the cohort sets become:

$$(C_a = \{a, b\}) \quad C_b = \{b, c\} \quad C_c = \{b, c\}$$

2. Replica b becomes unavailable and replica c recovers; the cohort sets of a and c are such that $C_a \not\subseteq C_c$ and $C_c \not\subseteq C_a$:

$$C_a = \{a, b\} \quad (C_b = \{b, c\}) \quad C_c = \{b, c\}$$

   Therefore, the protocol will fail to recognize that c is the current replica. As a result the replicated data object will remain unavailable although two out of three replicas can be accessed.

This is clearly a failure of our protocol: any protocol using version numbers would have compared the version numbers of replicas a and c and found that replica c had the higher version number.

To recover, our protocol will need instead to wait for the recovery of replica b. We will then have two replicas, namely b and c with identical cohort sets. Since these two replicas form a valid quorum, the protocol can use Lemma 1 to assert that the two replicas are both current and use them to bring replica a up-to-date.

The result is a slightly lower available availability when using cohort sets to implement majority consensus voting. But, as we will see, this is easily remedied by using dynamic voting.

B. Dynamic-linear voting with cohort sets

As we said before, dynamic voting protocols improve upon static voting protocols by either modifying the weights allocated to the surviving replicas [2] or temporarily excluding from quorum computations replicas that are unaccessible [4, 8]. These latter protocols, among which dynamic voting [4] and dynamic-linear voting [8], are particularly suited to implementations using cohort sets because we can use these cohort sets to store the set of replicas currently included in quorum computations. We will say that these replicas are part of the current *majority block*. To do that, we will have to modify very slightly the definition of cohort sets to exclude from the cohort sets replicas that are not in the current majority block.

Definition 2 *The* dynamic *cohort set for a replica represents the set of replicas that (a) were current*

after the last write or replica recovery in which the replica participated and (b) are in the current majority block.

Axiom 1 and Lemma 1, as well as Theorem 1, still apply although the read quorum mentioned in Lemma 1 and in Theorem 1 must now be replaced by a majority of the replicas belonging to the current majority block.

The simplicity of having to maintain only one cohort set can be contrasted with the complexity of the *optimistic dynamic voting* protocol [10] where each replica must maintain:

1. a partition set P_i representing the set of sites which participated in the last successful operation on the replicated data,
2. an operation number, o_i that is incremented at every access, and
3. a version number, v_i that is incremented at every write.

The price to pay for this simplification is minimal. First, we cannot reintegrate outdated replicas into the current majority block without bringing them first up-to-date. Second, excluding a replica from the majority block has now the side effect of marking the replica as being outdated even when the excluded replica is identical to the replicas in the majority block.

Majority blocks present the interesting property that any new majority block must contain a majority of the replicas in the current majority block. Because of this property, we will not encounter with dynamic voting protocols situations similar to that described in the previous subsection where a quorum of replicas was present but the protocol could not assert which replica(s) should be considered current. To understand this, let us return to the replicated data object of our previous example and assume now it is managed by a dynamic-linear voting protocol updating the cohort sets every time a write operation or a replica recovery occurs. Let us further assume that the dynamic-linear protocol ranks the three sites in the order $a > b > c$.

If the three replicas are up-to-date, their cohort sets will be equal to:

$$C_a = \{a, b, c\} \quad C_b = \{a, b, c\} \quad C_c = \{a, b, c\}$$

If a write occurs after c has become unavailable, the cohort sets of a and b will become equal to $\{a, b\}$ and c will be excluded from the majority block:

$$C_a = \{a, b\} \quad C_b = \{a, b\} \quad (C_c = \{a, b, c\})$$

Should a now become unavailable, replica b will remain the only accessible replica and the replicated object will remain unavailable until a recovers. Unlike what happened before, the recovery of c will not make the replicated object available again because replica c cannot participate in quorum computations before being formally reintegrated to the majority block. Hence the cohort sets of a, b and c will remain unchanged:

$$(C_a = \{a, b\}) \quad C_b = \{a, b\} \quad C_c = \{a, b, c\}$$

Thus, we have the following theorem:

Theorem 2 *An implementation of the dynamic-linear voting using cohort sets and no version numbers will never fail to detect a valid quorum of replicas within the current majority block.*

Proof:

1. At any given time, all replicas in the current majority block will always have the same cohort sets and these cohort sets will reflect the membership of the current cohort set. Since every new majority block must contain a majority of the replicas in the previous majority block, any subset of replicas having identical cohort sets and constituting a majority of the replicas in the majority block represented by these cohort sets will constitute a valid quorum of replicas within the current majority block.
2. Conversely, any set of replicas that does not include such a subset will not be a valid quorum within the current majority block because it will not contain a majority of the replicas in the current majority block.

■

In other words, an implementation of the dynamic-linear voting using cohort sets and no version numbers will always provide the same data availability as any other implementation of the dynamic-linear protocol that updates its metadata at the same frequency.

III. AVAILABILITY ANALYSIS

Availability is the most common measure of fault tolerance for repairable systems that are expected to remain operational over a long period of time. It is traditionally defined as the fraction of time a system is operational. In the case of replicated data objects, the availability of a replicated object represents the fraction of time that the consistency control protocol will allow access to the object.

Our system model consists of a set of sites with independent failure modes connected via a network which does not fail. When a site fails, a repair process is immediately initiated at that site. Should several sites fail, the repair process will be performed in parallel on those sites. Site failures are assumed to be exponentially distributed with mean λ, and repairs are assumed to be exponentially distributed with

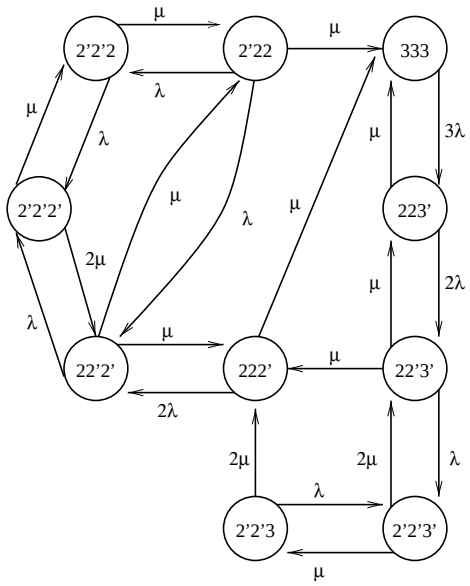

Figure 1: State transition diagram of MCV with cohort sets for three replicas

mean μ. The system is assumed to exist in statistical equilibrium. Although the assumption of an independent failure rate λ is reasonable if the sites have independent power sources, the assumption of exponential repair times is harder to defend on general grounds. However, both hypotheses are necessary to represent each system by a Markov process with a finite number of states [6].

A. Availability of static voting with cohort sets

Figure 1 has the state transition diagram for three replicas managed by a majority consensus voting protocol using cohort sets assuming that the data are *continuously accessed*. Each state is identified by a triple $\langle xyz \rangle$ where x, y and z represent the sizes of the cohort sets of the three replicas. The cohort sets of failed replicas are identified by a prime mark. States that have identical numbers of replicas in each cohort sets but differ in these cohort sets are identified by different orderings of the three digits.

State $\langle 333 \rangle$ represents the initial state of the three replicas when they are all operational and have identical cohort sets. The only two other available states are state $\langle 223' \rangle$ and state $\langle 222' \rangle$. State $\langle 223' \rangle$ represents the state of the system after one of the three replicas has failed and the cohort sets of the two remaining replicas have been updated. To reach state $\langle 222' \rangle$, the system should experience first a failure of one of its three replicas and move to state $\rangle 223' \langle$. Then a second replica should fail bringing the system to state $\langle 22'3' \rangle$. If the replica that failed first recovers first, its cohort set will be updated and the system will be in state $\langle 222' \rangle$.

Note also that state $\langle 2'22 \rangle$ is not available although two of the three replicas are operational. It corresponds to the case where the two operational replicas have intersecting cohort sets and the protocol cannot recognize which one of them is current.

The availability of the replicated data object is then given by:

$$A_{MCV\text{-}CS}(3) p_{333} + p_{223'} + p_{222'} = \\ = \frac{4\,\rho^5 + 31\,\rho^4 + 83\,\rho^3 + 91\,\rho^2 + 39\,\rho + 6}{(\,\rho + 1\,)^5\ (\,4\,\rho^2 + 9\,\rho + 6)}$$

where p_{ijk} denotes the probability that the system is in state $\langle ijk \rangle$ and $\rho = \frac{\lambda}{\mu}$.

This availability is slightly lower than if version numbers had been used to manage the replicas. In that case, the replicated object would have remained available whenever at least two of the three replicas were available and the overall data availability would have been:

$$A_{MCV\text{-}VN}(3) = A_{MCV\text{-}CS}(3) + p_{2'22} = \frac{3\rho + 1}{(\rho + 1)^3}$$

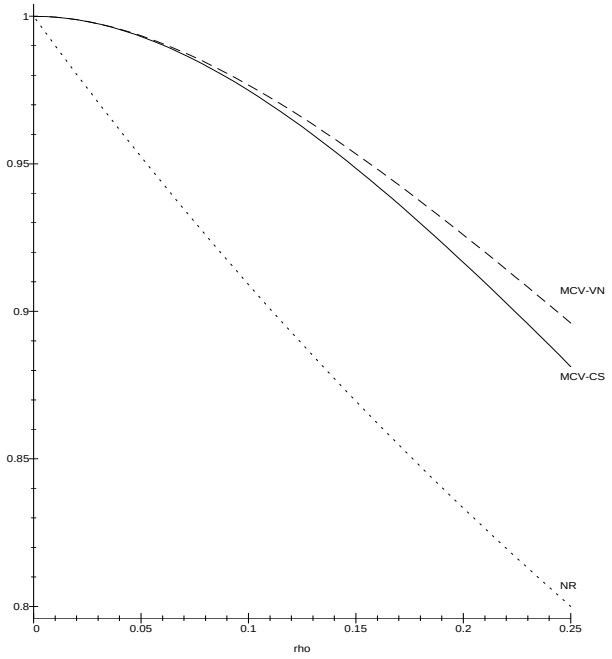

Figure 2: Compared availabilities of MCV for three replicas with version numbers (**top**), MCV for three replicas with cohort sets (**middle**) and a single replica (**bottom**).

Figure 2 displays the compared availabilities of MCV for three replicas with version numbers (*top graph*) and MCV for three replicas with cohort sets (*middle graph*). The bottom graph represents the availability of a single unreplicated data object and was included to provide a baseline.

We selected a range of values for ρ between 0 and 0.25 because a recent study [11] has shown that the

mean time to failure (MTTF) for modern systems is approximately 16 days plus or minus one. The mean time to repair (MTTR) is approximately 29 hours plus or minus two.

As one can see, the difference of data availability between the two implementations of MCV never exceeds 0.014 over the range of values of ρ being considered. This might be considered a small price to pay for not having to update the version numbers at every write access.

B. Availability of Dynamic-Linear Voting with Cohort Sets

The availability analysis of our new implementation of dynamic-linear voting is essentially similar to that of the *optimistic dynamic voting* protocol, as it was presented [10]. The only difference between the two implementations is that optimistic dynamic voting updates the metadata at every access while dynamic-linear voting with cohort sets will update the cohort sets when write operation discovers a failure.

The critical issue is not so much the data availability afforded by dynamic-linear voting with cohort sets as the minimum write access rate required to guarantee a timely detection of site failures and network partitions. Should write operations be not frequent enough to provide sufficiently fine grained detection of site failures and network partitions, then cohort sets can be modified when read operations occur. The penalty to pay then will be the possibility of unnecessary copy repairs. These costs can however be kept to a minimum if the repair process uses a smart algorithm to detect which pages of the replica need to be updated [1].

The availability of three replicas managed by the optimistic dynamic voting protocol was found to be equal to:

$$\frac{2\rho^4 + \phi\rho^3 + 6\rho^3 + 3\phi\rho^2 + 11\rho^2 + 4\phi\rho + 6\rho + \phi + 1}{(\rho+1)^4(2\rho+\phi+1)}$$

where $\rho = \frac{\lambda}{\mu}$, $\phi = \frac{\kappa}{\mu}$ and κ is the rate at which the replicated object is accessed [10]. This expression can also be used to represent the availability of three replicas managed by our new implementation of the dynamic-linear algorithm if we redefine κ to represent now the rate at which the data are *updated*.

Figure 3 represents the availability of three replicas managed by our implementation for values of ρ varying between 0 and 0.25 and ϕ varying between 0 and 20. As one can see, the impact of the update rate to repair rate ratio ϕ on the availability becomes insignificant as soon as $\phi > 4$ or, in other words, $\kappa > 4\mu$. It appears that there is no point in updating the cohort sets during read operations as long as the frequency of the write operations exceeds four times the failure rate. Assuming a mean time to fail of 16 days, this would mean two write access every week.

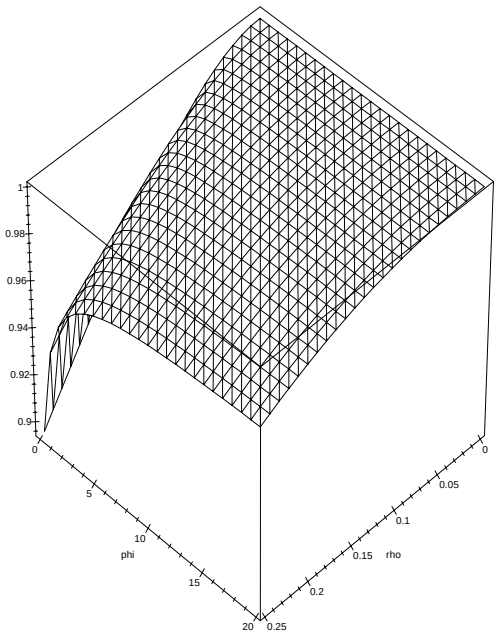

Figure 3: Availability of dynamic-linear voting with cohort sets as a function of ρ and ϕ.

Thus, two write accesses per week are enough to guarantee that dynamic-linear voting with cohort sets will never be outperformed by any other implementation of the dynamic-linear voting protocol despite the fact that all of these implementations use more complex metadata and update these metadata much more frequently.

IV. POSSIBLE EXTENSIONS

One of the most vexing limitations of voting protocol is the fact these protocols require at least three voting entities to improve upon the availability afforded by unreplicated data. Fortunately, one of these three entities can be a *witness*, that is an entity containing the same metadata as a regular replica but no data [12]. Cohort sets make witnesses especially attractive because they reduce to an absolute minimum the storage costs and the access costs.

To evaluate the storage cost of a witness we need to distinguish between static voting protocols where witnesses do not need to be included in cohort sets and dynamic protocols where unaccessible witnesses can be excluded from quorum computations and the status of each witness recorded in the cohort set of each voting entity. In the first case, adding one witness to a replicated object consisting of n voting entities requires $n + O(\log(n))$ additional bits. In the case of dynamic voting protocols, we must add to these $n + O(\log(n))$ bits the extra bit required at each of the $n + 1$ voting entities to store the status of the

witness for a total cost of $2n + O(\log(n))$.

The update costs of witnesses are also extremely low because cohort sets, unlike version numbers are only updated when a replica becomes unaccessible and when it can be reached again. Hence, total cost of updating a witness over its lifetime will depend only on the frequency of site failures and network partitions and not on the frequency of read and write accesses.

Because of these very low costs, it becomes feasible to multiply the number of witnesses to a point where the replicated data object would include more witnesses than replicas. We can view these witnesses as the distributed equivalent of a single very reliable witness that would know at any time which replicas are current and would enforce mutual exclusion. We would thereby achieve a very close approximation of the *optimistic available copy* protocol [3], which does not guarantee the consistency of the replicated data in the presence of network partitions but provides much higher data availabilities than voting protocols.

V. CONCLUSIONS

Voting protocols have been extremely popular during the last ten years due to their robustness and their conceptual simplicity. Unfortunately, voting protocols also suffer from some major drawbacks. First, they require at least three voting entities to improve upon the availability afforded by a single replica. Second, static voting protocols provide poor data availabilities compared to other replication control protocols. Finally, dynamic voting protocols were thought to be complicated to implement and require complex metadata.

We have presented a novel implementation of static and dynamic voting protocols using cohort sets and requiring only $n + \log(n)$ bits of state per voting entity. Unlike version numbers that need to be incremented every time the replicated data are updated, cohort sets are only updated whenever a change in the availability of the replicas is detected. We have shown under standard Markovian hypothesis that a static voting protocol using cohort sets provides almost the same availability as a static voting protocol using version numbers. We have described a dynamic voting protocol using cohort sets that provides the same data availability as a dynamic voting protocol using much more complex metadata. We have also found that cohort sets dramatically reduce the update costs of witnesses since witnesses would only need to be updated whenever a change in the accessibility of a replica is detected.

More work still needs to be done to investigate alternative implementations of the cohort set update process. One promising avenue would be to allow the cohort sets of some replicas to continue to include some replicas that failed before the last write but after the penultimate operation that recomputed the cohort set.

VI. REFERENCES

[1] D. Barbará, H. Garcia-Molina and B. Feijoo, "Exploiting Symmetries for Low-Cost Comparison of File Copies," *Proc. 8th Int. Conf. on Distributed Computing Systems*, (1988), pp. 471–479.

[2] D. Barbara, H. Garcia-Molina and A. Spauster, "Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment," *ACM TOCS*, Vol. 7, No. 4 (1989), pp. 394–426.

[3] P. A. Bernstein and N. Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases," *ACM TODS*, Vol. 9, No. 4 (1984), pp. 596–615.

[4] D. Davčev and W. A. Burkhard, "Consistency and Recovery Control for Replicated Files," *Proc. 10th ACM SOSP*, (1985) pp. 87–96.

[5] D. K. Gifford, "Weighted Voting for Replicated Data," *Proc. 7th ACM SOSP*, (1979), pp. 150–161.

[6] B. V. Gnedenko, *Mathematical Methods in Reliability Theory,* Moscow, English Translation, New York, Academic Press, (1968).

[7] N. Goodman, D. Skeen, A. Chan, U. Dayal, R. Fox and D. Ries, "A Recovery Algorithm for a Distributed Database System," *Proc. 2nd ACM PODS Symposium*, (1983), pp. 8–15.

[8] S. Jajodia and D. Mutchler, "Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database," *ACM TODS*, Vol. 15, No. 2 (1990), pp. 230–405.

[9] D. D. E. Long and J.-F. Pâris, "On Improving the Availability of Replicated Files," *Proc. 6th Symp. on Reliable Distributed Systems*, (1987), pp. 77–83.

[10] D. D. E. Long, "The Management of Replication in a Distributed System," Ph.D. dissertation, University of California, San Diego, 1988.

[11] D. D. E. Long, A. Muir, and R. Golding. "A Longitudinal Study of Internet Host Reliability," *Proc. 14th Symp. on Reliable Distributed Systems*, (1995), pp. 2–9.

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