A Leaner, More Efficient, Available Copy Protocol

Darrell D. E. Long[†]
Department of Computer Science
University of California
Santa Cruz, CA 95064

darrell@cs.ucsc.edu

Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204-3475

paris@cs.uh.edu

Abstract

Available copy protocols provide the highest data availability and data reliability of all replication protocols that do not regenerate failed replicas. Unfortunately, all existing implementations of available copy protocols either rely on complex procedures for ascertaining which replicas are up to date after a total failure or have to wait for the recovery of all failed sites.

We present a simple technique for efficiently implementing the available copy protocol. Our protocol does not require version numbers *and maintains only $n + \log(n)$ bits of state per replica. We also show under standard Markovian assumptions that our new protocol provides the same data availability as the best feasible implementations of the available copy protocol.*

Keywords: *distributed file systems, data replication, replication control protocols*

1. Introduction

Critical data are often replicated either to reduce read access times or to provide constant data availability in the presence of failures. This technique is known as *data replication*. As can be expected, data replication introduces its own problems, the most important of which is maintaining all replicas in a consistent state. This is a complex task because host failures and network partitions may occasion incomplete updates that leave some replicas inconsistent. Special *replication control protocols* have been devised to perform this task in a transparent fashion. These protocols differ in their message overhead, their handling of network partitions and the data availabilities they provide.

[†]The work of this author was done while a Visiting Scientist at IBM Almaden Research Center.

A first class of protocols makes the assumption that network partitions are either unlikely or unlikely to occasion conflicting updates. The best known of them are the *available copy protocol* (AC) [2, 7], the *regeneration algorithm* [17] and the Coda replication control protocol [18].

The second class of protocols take the approach that data consistency is much more important than data availability. These protocols rely on quorums to provide mutual exclusion and prevent conflicting updates. As a result, they provide lower data availabilities than the other protocols. The best known quorum-oriented protocols include *majority consensus voting* (MCV), *weighted voting* (WV) [5], *dynamic voting* (DV) [4], *dynamic-linear voting* (DLV) [9] and *voting with witnesses* (VWW) [15].

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*, that is an integer that is incremented each time the replicated data are modified. Protocols such as *optimistic available copy* [10] and 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 *was-available set*, a *connection 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. As we will see, 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 a new implementation of Bernstein and Goodman's *available copy protocol* [2]. Our new protocol maintains 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 protocol requires 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. The recovery procedure is also greatly simplified as it suffices now to gather all the replicas in any mutually agreed cohort set to find the current version of the replicated object.

The remainder of this paper is organized as follows: Section 2 contains a review of existing replication control protocols and Section 3 introduces our new protocol; Section 4 includes a study of the dependability of our protocol. Possible extensions are discussed in Section 5 while Section 6 has our conclusions.

2. Available Copy Protocols

Available copy protocols are based on the observation that if any one site has received all updates to a given data object it holds the current version of the data object. Since they discount the possibility of network partitions, they can allow access to a replicated data object as long as a single replica of the data object remains available. As a consequence of this, they provide the highest data availability and data reliability of all replication protocols that do not regenerate failed replicas [16].

There are three parts to an available copy protocol: write, read and recovery. The rule for writing is extremely simple: *write to all accessible replicas*. Since all accessible replicas receive each write, they are kept in a consistent state: the replicated data can then be read from *any* accessible replica. When a site holding a replica recovers from a failure, this replica needs to be compared, in some manner, with another replica that contains the current version of the data object. If all sites holding replicas of the data object have failed, no replica can recover until the last site(s) to fail can be found. This is the most complex part of any available protocol and the only one to differ significantly from one implementation to another.

2.1. The original available copy protocol

The original available copy protocol [2, 7] relies on a complex mechanism to locate that site. Several sets of failure information are to be maintained in real time, including the set of sites participating in the replication of the data object and the sets of sites that had been specifically *included* or *excluded*. An *included* site s is one that is known to hold a current replica of the data object while an *excluded* site t is one that has failed and whose failure has been recorded by an operational site executing an **exclude(**s**)** transaction. When a failed site t repairs following a failure, it attempts to locate another site s that is operational. If such a site can be found, then t will repair from s and request s to execute the transaction **include(**t**)**. In the presence of a total failure, the sets of included and excluded sets are used to determine the site—or set of sites—that failed last and holds a current replica of the data object.

2.2. The naïve available copy protocol

The *naïve available copy* (NAC) protocol [16] avoids the problem of failure detection by not maintaining any site failure information. It behaves like the original available copy protocol except in the event of a total failure, in which case it must wait for *all* sites participating in the replication to recover. The only metadata maintained by the NAC protocol are the *version numbers* of the replicas.

The price for the simplicity of the NAC protocol is a slower recovery after a total failure and a lower overall data availability. In most cases, total failures will be rather exceptional events that are much more likely to result from a catastrophic event affecting all sites holding replicas than from successive site failures. When this is the case, *all* available copy protocols will have to wait for the recovery of all sites holding replicas.

2.3. The optimistic available copy protocol

Like the original available copy protocol, the *optimistic available copy* (OAC) protocol [10] maintains availability information about each and every replica but it only updates this information when the replicated data object is modified or when a recovery occurs. The protocol maintains two pieces of information at each site holding a replica: a *version number* and a *was-available set*. The *was-available set* for an active replica s, denoted W_s, lists those replicas that s knows to be up to date. This includes all replicas that received the most recent write and all replicas that have repaired from s since the last write.

Was-available sets can be maintained inexpensively by ascertaining which replicas are operational when the replicated data object is first accessed and by sending this information along with the first write; the second write will contain the set of replicas which received the first write and so forth.

Similarly, when a replica t recovers from a replica s, s sends to t its new was-available set $W_s \cup \{t\}$. Recovering from a total failure requires finding the last site(s) that failed. These sites are known to belong to the closure of the was-available set with respect to the recovering site s, that is

$$C^*(W_s) = \bigcup_{k=0}^{n} C^k(W_s)$$

where $C^k(W_s) = \bigcup_{t \in W_s} C^{k-1}(W_t)$ and $C^0(W_s) = W_s$.

3. A More Efficient Available Copy Protocol

One of the major objectives of the OAC protocol was to reduce the costs of updating the was-available sets of operational sites [11]. So it was decided that:

1. was-available set updates should always be piggy backed on existing read, write and site recovery operations, and
2. was-available set updates should never involve sites that were not involved in each read, write or site recovery operation.

Hence, site recovery operations only update the was-available set of the two sites actually participating in the actual recovery, namely the recovering site and that of the site from where the recovering site obtained the correct state of the replicated data object. As a result, the was-available sets of the operational sites cease to be identical after a site recovery because only two operational sites will have included the site that recovered in their was-available sets. Updating the was-available sets of *all* operational sites would have had the two advantages of (a) making all available sets current and (b) removing the need to compute the closure of these sets every time the system has to recover from a total failure.

As it happened, the OAC protocol was formalized [10] well before its data availability was fully analyzed [11, 16]. So the benefits of updating the was-available sets at recovery time were only understood after the protocol had been fully specified and this update was done independently of the site recovery process itself [11, 16].

An even more important simplification could be achieved if the was-available sets could be always be correctly updated every time the replicated object is modified. We would know then that all the sites in the most recent was-available sets would all have the most recent version of the replicated object and would not need *version numbers* to distinguish them.

Table 1. Example of failure and recovery

C_A	C_B	C_C
A,B,C	A,B,C	A,B,C
A,B	A,B	-
A	-	-
A	A,B	A,B,C

We propose to record *exact* membership information in new metadata, which we will call *cohort sets* to distinguish them from was-available sets. A cohort set for any replica represents the set of replicas that participated in the last write that involved that replica. For example, if there are three replicas, A, B and C, and C_A, C_B and C_C are the corresponding cohort sets, Table 1 can be used to illustrate what happens when replicas fail and recover. Suppose that the system starts with a full complement of replicas. At some time in the future, replica C fails, and a write operation occurs. The state of the system is reflected in the second row of the table, where $C_A = C_B = \{A, B\}$, which indicates that replicas A and B were the only participants in the last write operation. At some point further on, suppose that replica B also fails, followed by a write operation. This is reflected in the third row, where only replica A is current. Suppose now that the other two replicas recover, then the state of the system is reflected in the fourth row. The only replica to be current is replica A because $C_A = \{A\}$.

Thus the rule is that, when k of the original n replicas have identical cohort sets all containing exactly these same k replicas and no others, we can assert that these k replicas are all current and all other replicas are stale. Hence version numbers are redundant.

In general, we can also expect site failures to be much less frequent events than write operations. In this situation, cohort sets will almost always have been updated between consecutive site failures. Thus, after a failure of all sites holding replicas, the k current replicas, will also be the k last replicas that failed last. We can even expect k to be equal to one unless the last sites that failed did it so closely together that no write access took place during that time. The worst case is of course a simultaneous failure of *all* sites holding replicas. All replicas will be current but to establish this fact beyond any doubt we will need all sites holding to recover first. In this case, the new protocol will perform no better (and no worse) than the NAC protocol, which requires version numbers.

In the following section, we consider cohort sets a little more formally.

3.1. Cohort sets

A cohort set is the set of all replicas which have participated in the last write operation. The cohort sets are similar to the was-available sets used for the available copy protocol, except by managing them differently, we avoid the need to compute the closure. This results in a substantial simplification in the recovery procedure.

Definition 1 *A cohort set for a replica represents the set of replicas that were current during the last write that included this replica.*

It should be clear that the last replica or replicas to fail must be represented by this set, since if any of the replicas had participated in subsequent write operation, then its cohort set would not contain the replicas that did not participate in this operation.

It can be shown that the current set of replicas must have equal and complete cohort sets. By *equal*, we mean that the cohort sets for the replicas in question must have the same membership. By *complete*, we mean that every replica in question must be represented in the cohort sets, and every replica in the cohort sets must be present and under consideration.

Theorem 1 *The necessary and sufficient condition for recovery is that a subset of replicas can be found such that their cohort sets are equal and complete.*

Instead of presenting a formal proof, which would be tedious and not very illustrative, we will demonstrate the result in an informal manner.

Perhaps the easiest way to understand this result, is to view it as a directed graph. Let each copy be represented by a node, and the cohort set of each copy represent the out-going edges from that node. That is, if copy A has $C_A = \{A, B\}$ then node A would have directed edges $\vec{AA}$ and $\vec{AB}$.

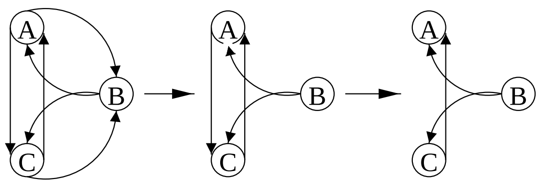

Figure 1. A sequence of failure events.

If we consider Figure 1, we see that in the initial state, the system is represented by a completely connected graph, where every node has an edge to every other node. In this case, it is clear that having complete and equal cohort sets is equivalent to having a completely connected graph.

If we now consider the write operation, it will write identical cohort sets to all live copies. These cohort sets list all of the live copies, and when viewed as a graph they form a completely connected subgraph. In our example, suppose that copy B fails, and this failure is followed by a write operation. Since B has failed, the write cannot change its cohort set, and so its out-going edges remain unchanged, but the edges from A and C to B are deleted, and nodes A and C now form a completely connected subgraph. Node B is now a disconnected component. Since no edges outside the completely connected subgraph are added, the subgraph remains unique. If we take the example one step further, suppose now that node C fails. Its edge to node A will remain, but there will be no edge back from node A and so node C is also in a disconnected component. Since node A has only the edge $\vec{AA}$ it is the unique completely connected subgraph.

To understand the repair operation, consider Figure 2. The recovery operation is the inverse of the write operation. When a node which was disconnected is brought up to date, it is given the same cohort set as those nodes in the (unique) completely connected subgraph (which now includes the recovered node). Since these cohort sets do not include any nodes outside of the completely connected subgraph, the subgraph remains unique. Suppose now that node B is repaired. In this case, $C_A = C_B = \{A, B\}$, which, when views as a graph means that node A has edges $\vec{AA}$ and $\vec{AB}$ and node B has edges $\vec{BB}$ and $\vec{BA}$. When node C recovers, it is handled in the exact same manner.

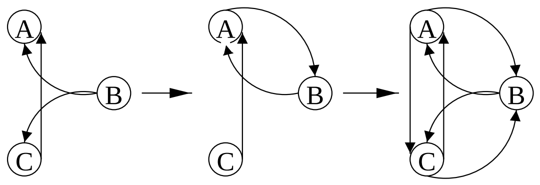

Figure 2. A sequence of repair events.

3.2. Reading and writing

The cohort sets are the sole metadata required for accessing the replicated data. These sets must be consistent for the recovery algorithm to operate correctly. Hence they must be completely written to stable storage after the detection of a every failure. While extremely rare, it is possible that a second failure could occur while the cohort sets are being written.

There are several methods for insuring that the cohort sets are written to stable storage in a consistent fashion. The first is to leverage an existing commit mechanism [8]. The second is to employ a reliable multicast protocol [3]. We will describe a third method that uses a simple two phase write protocol to insure that the cohort sets are written in a consistent manner.

In order to mitigate the effects of this unlikely failure scenario, two phases are used to write the cohort sets to stable storage. In the first phase, so-called *tentative cohort sets* are written. These sets are exactly like the regular cohort sets, but exist only briefly until the *committed cohort sets* have been safely written to disk. If this fails, the system can fall back to the original committed cohort sets. In the second phase, the tentative cohort sets are cleared and the committed cohort sets are written. Should this fail, then the tentative cohort sets that remain can be used in conjunction with the newly committed cohort sets to determine set of consistent replicas.

The cohort sets are modified when a write operation occurs following a failure. It is assumed that write operations are frequent enough to provide sufficiently fine grained failure detection. If this is not the case, then cohort sets can be modified when read operations occur. If an asynchronous failure notification mechanism is available, then this can be used to modify the cohort sets.

3.3. Recovering individual replicas

In the absence of total failure, a recovering replica will find replicas that are available. It is then a simple matter to integrate this recovering replica into the set of current replicas. First, the data from one of the current replicas is copied to the recovering replica. Second, the cohort set of one of the current replicas (recall that they are identical) is taken and the identity of the recovered replica is added to it. This new cohort set is then written to all current replicas, including the one which has just recovered.

3.4. Recovering from a total failure

The recovery from total failure is the most intricate operation in the system. As discussed in the previous section, the cohort sets must be carefully maintained. If we employ the two phase algorithm described in Section 3.2, then the following algorithm can be applied.

In order to declare a set of replicas current, the cohort sets are checked for equality and completeness in the following order:

1. All cohort sets are tentative. This will succeed if there was a failure after the tentative cohort sets were written, but before the committed cohort sets were written.
2. A mix of tentative and committed cohort sets. This will succeed if there was a failure while the committed cohort sets were being written.
3. Only the committed cohort sets. This is the most common case, and will succeed if the two phase write completed successfully.

There is one further case, that is, when a failure occurs during the initial writing of the cohort sets. In this case, the cohort sets can be safely ignored since the committed cohort sets represent a consistent view of the system.

The system considers each recovering replica in turn as it becomes active. This replica will compare its cohort set to the cohort sets of all other reovering replicas. When it is able to contact all replicas in its cohort set, and the cohort sets of each of these replicas agree with it (the equal and complete property), then this replica and all replicas in its cohort set can be declared to be current.

Replicas which are unable to complete this procedure are out-of-date and must repair from one of the current replicas, as discussed in Section 3.3.

4. 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.

The analysis of our new available copy protocol is identical to the analysis of the OAC protocol presented in [11] and [16].

The 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 mean μ. All access requests are assumed to be characterized by a Poisson process with mean κ. The system is assumed to exist in statistical equilibrium. Although the assumption of

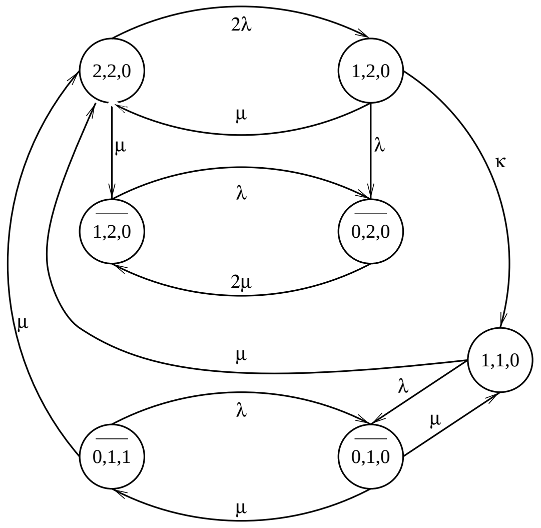

Figure 3. State transition diagram

an independent failure rate λ is reasonable if the sites have independent power sources, the assumptions of exponential repair times and exponential inter-access times are harder to defend on general grounds. However, all three assumptions are necessary to represent each system by a Markov process with a finite number of states [6].

Definition 2 *The* availability *of a replicated data object consisting of n replicas and managed by a replica control protocol S, denoted $\mathcal{A}_S(n)$, is the stationary probability of the system being in a state where the replica control protocol will grant access to the data object.*

The states of the Markov model are labeled by the ordered triple $\langle i, j, k\rangle$ where i represents the number of current (or up-to-date) replicas, j represents the cardinality of the current cohort set, and k represents the number of replicas that are out-of-date. When a triple is marked with a bar, for example $\langle\overline{1, 2, 0}\rangle$, this indicates that the system is unavailable.

Figure 3 has the state transition diagram for two replicas managed by the new AC protocol. A system of equations can be derived from this state transition diagram, and solved either algebraically or using numerical methods. If we let $\rho = \frac{\lambda}{\mu}$ and $\phi = \frac{\kappa}{\mu}$, then the equations are significantly simplified.

The availability, $\mathcal{A}_{AC}(2)$, of the system is the sum of probabilities of being in a state where access is permitted, and is given by the expression:

$$\mathcal{A}_{AC}(2) = \frac{\phi\rho^2 + 3\rho^2 + 3\phi\rho + 4\rho + \phi + 1}{(\rho+1)^3(\rho+\phi+1)}$$

If the writes occur with sufficient frequency that the cohort sets can be assumed to be up-to-date, then the availability is given by the expression:

$$\mathcal{A}_{AC}(2) = \frac{\rho^2 + 3\rho + 1}{(\rho+1)^3}$$

The availability of a system with three replicas can be derived in a similar manner. In this case, the state diagram has sixteen states. The resulting expression is very large, and has been omitted for the sake of brevity. If we again make the assumption of frequent writes, the then availability of system with three disks is given by the expression:

$$\mathcal{A}_{AC}(3) = \frac{2\rho^4 + 11\rho^3 + 17\rho^2 + 9\rho + 2}{(\rho+1)^3(2\rho^2 + 3\rho + 2)}$$

This analysis can be done for any number of replicas, though the equations quickly become unmanageable. If the frequent write assumption is made, then a closed form solution has been derived [11].

Figure 4 and 5 respectively represent the availability of two and three replicas managed by the new AC protocol for values of ρ varying between 0 and 0.25. We selected these values because a recent study [12] has shown that

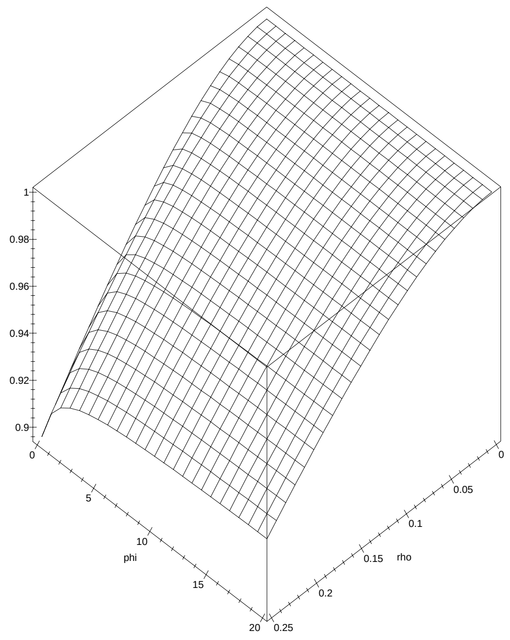

Figure 4. Availability of two replicas managed by the new AC protocol

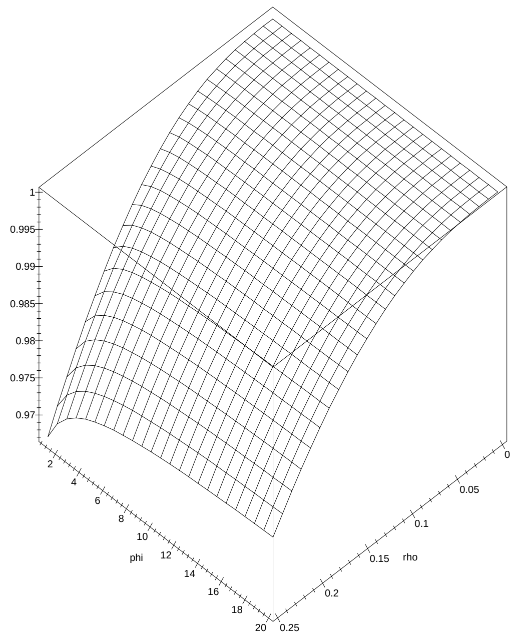

Figure 5. Availability of three replicas managed by the new AC protocol

the mean time to failure (MTTF) for modern systems is approximately 29 days plus or minus 2. The mean time to repair (MTTR) is approximately 4 days plus or minus one. This results in reasonable values for ρ falling in the interval $0.06 < \rho < 0.22$ for the average host connected to the Internet. Dedicated servers are likely to have service contracts which will result in a MTTR of one day or less, which will significantly lower the reasonable values of ρ into the range of $0.03 < \rho < 0.04$. Professional maintenance and conditioned power will also significantly increase the MTTF, but these influences are more difficult to quantify. 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, $\kappa > 4\mu$.

5. Possible Extensions

There are at least two possible extensions to our new protocol that are worth mentioning. The first extension concerns protocols that do not guarantee data consistency in presence of network partitions but promise to detect *ex post facto* data inconsistencies that may have resulted from these partitions. A second applies to quorum-based protocols.

5.1. Protocols detecting data inconsistencies

Available copy protocols were designed for network topologies where network partitions were known to be impossible or extremely unlikely. Other protocols, such as the Coda replication control protocol [18, 14], follow the same *write to all/read any* philosophy as the available copy protocol but promise to detect *ex post facto* data inconsistencies that may have resulted from these partitions.

Detecting data inconsistencies in our protocol will involve comparing the cohort sets of the replicas searching for disjoint subsets of replicas such that every replica in each subset has a cohort set describing exactly that subset. If we find two or more of these subsets, there are two or more different versions of the file pretending to be the current version of the file. Otherwise we know that the data object has one single current state.

5.2. Extension to quorum-based protocols

Unlike available copy protocols, quorum-based protocols guarantee the consistency of the replicated data in the presence of network partitions. In their simplest form, quorum-based 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 the votes of a *quorum* of the replicas. Should this be prevented by a sufficient number of site failures, the replicated object is considered to be inaccessible. Protocols that adjusts quorums, such as dynamic voting and its variants [4, 9], or modify the number of votes assigned to each replica [1], are known to provide higher data availability than protocol using static quorums.

Cohort sets represent the set of replicas that participated in the last write operation. Whenever writes are significantly more frequent than site failures, they also provide a good approximation of the set of currently available replicas and can thus be used to implement dynamic voting protocols [13].

6. Conclusions

Available copy protocols provide the highest data availability and data reliability of all replication protocols that do not regenerate failed replicas. Unfortunately, all existing implementations of available copy protocols either rely on complex procedures for ascertaining which replicas are up to date after a total failure or have to wait for the recovery of all failed sites.

We have presented a simple technique for efficiently implementing the available copy protocol. Our protocol does not require *version numbers* and maintains only $n + \log(n)$ bits of state per replica, that is n bits for storing the current set of active replicas (the so-called *cohort set* and $\log(n)$ bits for storing the identity of the replica. The recovery procedure is also greatly simplified as it suffices now to gather all the replicas in any mutually agreed cohort set to find the current version of the replicated object.

We have also shown that our new protocol provides the same data availability as the best feasible implementations of the available copy protocol.

More work still needs to be done to extend the applicability of our technique and 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.

Acknowledgements

We are grateful for the input of our colleagues at the IBM Almaden Research Center, in particular L. F. Cabrera, N. Pass, N. Hanami, S. Edelman and A. Lam.

References

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

[2] P. A. Bernstein and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases," *ACM Transactions on Database Systems*, Vol. 9, No. 4 (1984), pp. 596–615.

[3] K. Birman and T. Joseph, "Reliable Communication in the Presence of Failures," *ACM Transactions on Computer Systems*, Vol. 5, No.1 (1987), pp. 47–76.

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

[5] D. K. Gifford, "Weighted Voting for Replicated Data," *Proc. 7th ACM Symposium on Operating System Principles*, (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 Symposium on Principles of Database Systems*, (1983), pp. 8–15.

[8] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* Morgan Kaufman Publishers, San Mateo, Calif. (1993).

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

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

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

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

[13] D. D. E. Long and J.-F. Pâris, "Voting without Version Numbers," submitted for publication.

[14] L. B. Mummert, M. R. Ebling and M. Satyanarayanan, "Exploiting Weak Connectivity for Mobile File Access," *Proc. 15th ACM Symposium on Operating Systems Principles*, (1995), pp. 33–45.

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

[16] J.-F. Pâris and D. D. E. Long, "On the Performance of Available Copy Protocols," *Performance Evaluation*, Vol. 11, (1990) pp. 9–30.

[17] C. Pu, J. D. Noe and A. Proudfoot, "Regeneration of Replicated Objects: A Technique and its Eden Implementation," *IEEE Transactions on Software Engineering*, Vol. SE-14, No. 7 (1988), pp. 936–945.

[18] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A Highly Available File System for a Workstation Environment," *IEEE Transactions on Computers*, Vol. C-39, No. 4 (1990), pp. 447–459.