Protecting Replicated Objects Against Media Failures

Jehan-François Pâris

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

Darrell D. E. Long

Computer and Information Sciences
University of California
Santa Cruz, CA 95064

Proceedings of the Workshop on Research Issues on Transaction and Query Processing, Phoenix: IEEE, February 1992, pp. 109–115.

Abstract

We present a replication control protocol that provides excellent data availabilities while guaranteeing that all writes to the object are recorded in at least two replicas. The protocol, *robust dynamic voting* (RDV) accepts reads and writes as long as at least two replicas remain available. The replicated object remains inaccessible until either the two last available replicas recover or one of the two last available replicas can collect the votes of a majority of replicas.

We evaluate the read and write availabilities of replicated data objects managed by the RDV protocol and compare them with those of replicated objects managed by majority consensus voting, dynamic voting and hybrid dynamic voting protocols. We show that RDV can provide extra protection against media failures with no significant loss of availability.

1 Introduction

Distributed transaction processing systems often have to meet strict response time and availability requirements. One possible approach to meet these requirements is to maintain multiple copies or *replicas* of all critical data. Data replication, as this technique is known, offers indeed several major advantages. First, it considerably reduces the likelihood that the data could be destroyed due to a media failure or other catastrophe. Second, it improves the response time of read-only transactions by sharing the load among all sites holding replicas. Finally, it greatly increases the likelihood that the data will remain accessible in the presence of site crashes or communication failures.

Managing replicated data is nevertheless a complex task as the multiple copies of each replicated data object must be kept consistent. Special replication control protocols have been devised to perform that task and to provide the users with a single consistent view of every replicated data object.

The last five years have seen the development of very efficient replication control protocols that require fewer replicas and yet provide higher availability than older protocols such as primary copy or majority consensus voting. Some of these protocols, such as *dynamic-linear voting* and all available copy protocols, allow access to the replicated data object when only one replica remains accessible.

Allowing writes when only one replica is available has the negative side-effect of making the replicated data object singularly vulnerable to media failures. It also requires the storage of all replicas, and their logs, in stable storage [10]. We investigate the feasibility of replication control protocols that guarantee that all writes and all quorum updates are recorded into at least two replicas of the data object. We introduce a dynamic voting protocol, *robust dynamic voting* (RDV) that accepts reads and writes as long as at least two replicas remain available. Following a total failure, the replicated object remains inaccessible until either the two last available replicas recover or one of the two last available replicas can collect the votes of a majority of replicas. As our analysis will show, this added protection against media failures has no significant effect on availability.

Section 2 of this paper reviews the extant replication control protocols that are relevant to our discussion. Section 3 introduces the RDV protocol while section 4 presents a brief analysis of replicated object availability under RDV and section 5 discusses two possible extensions to our basic protocol. Finally, section 6 presents our conclusions.

2 Related Research

When network partitions are known to be impossible, the *available copy* (AC) protocol [1] and its variants [15] provide a simple means for maintaining data consistency. The AC protocol is based on the observation that replicas that have participated in all writes necessarily hold the most recent version of the data. The write rule for the AC protocol is extremely simple: *write to all accessible replicas.* Since network partitions are specifically excluded, all available replicas receive each write. As a result, replicated objects managed by the AC protocols can remain available so long as at least one of their replicas remains accessible. Data can be read from any accessible replica, which greatly reduces communication costs. Replicas recovering from a failure can repair immediately so long as there is another replica holding the most recent version of the data. After a failure of all replicas, the recovering sites cannot ascertain which of them hold the most recent version of the data until the replica that failed last can be found.

Large local-area networks often consist of several carrier-sense segments or token rings linked by repeaters or gateways. Since repeaters and gateways may fail without halting the operation of the entire communication network, these networks are just as susceptible to network partitions as are long-haul point-to-point networks. Replicated data objects having replicas on both sides of a partition could be left with two sets of mutually inconsistent replicas. Although various merging algorithms have been developed to attempt to reconcile these inconsistencies when the partition is repaired, the safest solution to the problem is to adopt a replication control protocol based on quorum consensus.

Quorum consensus protocols, among which *majority consensus voting* (MCV) [5, 17] and *weighted voting* [6], ensure the consistency of replicated data objects by disallowing all read and write requests that cannot collect an appropriate quorum of replicas. Different quorums for read and write operations can be defined and different weights, including none, assigned to every replica [6]. Consistency is guaranteed as long as the write quorum W is high enough to disallow parallel writes on two disjoint subsets of replicas, and the read quorum R is high enough to ensure that read and write quorums always intersect. These conditions are simple to verify, which accounts for the conceptual simplicity and the robustness of voting schemes. Voting has however some disadvantages. It requires a minimum number of three copies to be of any practical use. Even then, quorum requirements tend to disallow a relatively high number of read and update operations. As a result, quorum consensus protocols using static quorums provide reliability and availability figures well below those provided by available copy protocols [15].

Unlike MCV, the *dynamic voting* (DV) protocol [4] automatically adjusts its access quorum to changes in the state of the network. Whenever some replicas of an object become inaccessible either because of a site failure or a network partition, the DV protocol checks if enough replicas remain available to satisfy its current quorum. If this is the case, these replicas constitute a new *majority block,* and a new access quorum is computed. To enforce mutual exclusion, recovered replicas that do not belong to the current majority block are not allowed to participate in elections so long as they have not been formally reintegrated by the protocol. To keep track of the status of the replicated object, every replica maintains some state information. This information will include a *version number* identifying the last write recorded by the replica and either a *partition vector* [4] or both a *partition set* and an *operation number*[14] identifying the replicas belonging to the current majority block.

All quorum-oriented protocols encounter situations where the number of current replicas within a group of mutually communicating replicas is equal to the number of current replicas not in communication. The DV protocol declares then the replicated object to be inaccessible. *Dynamic-linear voting* (DLV) [9] resolves these ties by applying a total ordering to the sites. This simple improvement greatly enhances the availability of the replicated data. *Weighted dynamic voting* (WDV) [3] can achieve the same result by allocating different number of votes —or *weights*— to the replicas. Another variant, *hybrid dynamic voting* (HDV) operates exactly like DLV as long as the replicated object has at least *three* mutually communicating replicas [8]. If one of these three replicas becomes unaccessible, HDV does not update the majority block and starts operating as a static voting protocol. As a result, the replicated object will remain available as long as two of the three replicas in the majority block are in communication. Should this be prevented by additional failures, the replicated object will become unavailable until two replicas from the current majority block can communicate again with each other.

3 The New Protocols

To protect a replicated data object against irrecoverable failures, its replication control should ensure that:

1. all writes to the object are recorded into at least two replicas of the object, and

2. the object can recover from an irrecoverable failure from either of these two replicas

Several protocols satisfy the first but not the second condition. For instance, the DV protocol guarantees that all updates are recorded in at least two replicas of the object but cannot recover from a failure of either of the last two operational replicas of an object until *both* replicas recover.

The only two extant protocols satisfying both conditions are MCV and HDV. The limitations of MCV have been already mentioned. Since the protocol relies on static quorums, it disallows all requests that cannot collect the votes of a majority of the replicas. As a result, it requires a higher level of replication to achieve the same levels of availability and reliability as available copy or dynamic voting protocols. Being a dynamic protocol, HDV does not suffer from these limitations: it can indeed be viewed as an improvement of the DV protocol providing a faster recovery from situations where less than two current replicas remain available. While DV does not allow access to the replicated object until its last two current replicas become available again, HDV allows the object to recover when two of its last three current replicas are available. This condition can be satisfied if the last two current replicas of the object have recovered or one of them plus a specially designated "third replica" have recovered. The role of this third replica is to allow the replicated object to recover when only one of its last two current replicas has recovered without risking the inconsistencies that might result if each of its last two current replicas was allowed to lead a recovery of the replicated object independently of the other.

We propose to improve upon the HDV protocol by replacing this third replica by an entity that will play the same role but be more available. Our solution consists of allowing the object to recover when either its last two current replicas have recovered or only one of them has recovered but the votes of a majority of the $n-2$ other replicas of the objects can be gathered. A protocol based on this recovery rule will allow a faster recovery under most practical circumstances since it will take less time to wait for the recovery of a majority of a given set of replicas than for the recovery of a given replicas. The only two major exceptions to this rule are the cases when (a) the individual availabilities of the replicas of a data object are less than $^1/_2$, or (b) the sites holding the failed replicas cannot be all repaired in parallel and the replication control protocol also controls the scheduling of site repairs. Condition (a) is rather pathological as replicas with availabilities below $^1/_2$ would be unaccessible most of the time. Condition (b) is rarely met in real networks as the time actually spent servicing faulty sites is normally a rather small part of the total site recovery time.

Replicated data objects with only two replicas constitute a special case: Since all writes are recorded into both two replicas of the object, either of these two replicas will always be current. We will therefore allow read access as long as one replica of the object remains available.

We will assume in our formal description of the *robust dynamic voting* protocol (RDV) that each site holding a replica maintains the three sets of information needed to implement an optimistic dynamic voting protocol. These three sets are: the partition set, P_i, which represents the set of sites which participated in the last successful operation, an operation number, o_i incremented at every successful operation and a version number, v_i incremented at every successful write. This information is stored at each site and is modified when an access occurs.

```
procedure READ
begin
    $\langle$R,o,v,P$\rangle$ $\leftarrow$START($U$)
    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$_S$} }
    T$\leftarrow$R-P$_m$
    q=$\frac{|P_m|}{2}$
    t=$\frac{|U-P_m|}{2}$
    choose any $m \in Q$
    if |Q| > q $\vee$ |Q|=q>1$\wedge$max($P_m$)$\in$ Q$\vee$
    |Q|=q=1$\wedge$(|T|>t$\vee$|T|=t$\wedge$max(U-$P_m$)$\in$T) then
        perform the read
        COMMIT(S,o$_m$+1,v$_m$,S)
    else
        ABORT(R)
    fi
end READ
```

Figure 1: Read Algorithm

The algorithm for performing a read operation is simple. It first ascertains whether the current partition is the majority partition. A message is broadcast to all n sites in U requesting their partition set, operation number and version number; those that send replies are considered to be in the current partition. The set of sites holding up-to-date replicas S is found by computing the maximum version number of all of the sites. The set of current accessible sites is called the *quorum set Q*. The access request is granted if Q represents a majority of the previous quorum P_m and Q contains at least two replicas or if Q contains one of the two replicas of P_m and the replicas in R but not in P_m constitute a majority of the replicas in U but not in P_m. We will refer to these replicas as the *excluded* as they are excluded from the current partition set. If there is a tie, that is 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.

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 by the COMMIT operation.

The algorithm for writing is similar to the algorithm for reading. It ascertains first if the current partition is the majority partition and if Q contains at least two elements. If this is successful, 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.

The recovery algorithm starts by ascertaining if the recovering site is able to gather a write quorum or find one of the last two current replicas and gather the votes of a majority of the $n-2$ other replicas. It determines then whether the replica at that site is up-to-date; if it is not, then it must be copied from one 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.

4 Availability Analysis

In this section we present an analysis of the availability provided by the RDV protocol. We will assume here that the availability of a replicated data object is the stationary probability of the object being in a state permitting access at at least one site. This is the traditional definition of the availability of a redundant system [18].

```
procedure WRITE
begin
    $\langle R, \mathbf{o}, \mathbf{v}, \mathbf{P} \rangle \leftarrow \mathrm{START}(U)$
    $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{O_S\}\}$
    $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{V_S\}\}$
    $T \leftarrow R - P_m$
    $q = \frac{|P_m|}{2}$
    $t = \frac{|U - P_m|}{2}$
    choose any $m \in Q$
    if $|Q| > q \vee |Q| = q > 1 \wedge \max(P_m) \in Q$ then
        perform the write
        COMMIT($S, o_m+1, v_m, S$)
    else
        ABORT($R$)
    fi
end WRITE
```

Figure 2: Write Algorithm

Definition 3.1 *A replica is said to be live if it resides on a site that has not experienced any failure since the last time the replica was written to or repaired.*

Definition 3.2 *A replica is said to be dead if it resides on a site that is not operational.*

Definition 3.3 *A replica is said to be comatose if it resides on a site that has failed and the replica has not been repaired yet.*

Our model consists of a set of sites with independent failure modes that are connected via a local-area network. 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 failed sites. We assume that failures are exponentially distributed with mean failure rate λ, and that repairs are exponentially distributed with mean repair rate μ. The system is assumed to exist in statistical equilibrium and to be characterized by a discrete-state Markov process. No attempt is made to model failures of LAN segments, gateways or repeaters. We also assume that file accesses are frequent enough to ensure that the state information stored at the sites always reflects the current state of the system.

The assumptions that we have made are required for a steady-state analysis to be tractable [7]. These assumptions have been made in most recent probabilistic analyses of the availability of replicated data[8, 11, 15, 13]. Purely combinational models that do not require assumptions about failure and repair distributions have been proposed [16, 19] but these models cannot distinguish between live and comatose replicas.

Like all voting protocols that make no assumptions about the topology of the network, the RDV protocol requires a minimum of three replicas to improve upon the write availability of a non-replicated object. When three replicas are present, it performs exactly as majority consensus voting and hybrid dynamic voting.

Figure 4 describes the state transition rate diagram for a replicated object consisting of four replicas managed by the RDV protocol. As in all our next state transition rate diagrams, the numerical value of the label of every state represent the number of replicas that are either live or comatose when the system is in that state. The 3 non-primed states labeled from 2 to 4 represent the 3 available states of the object. Primed states are entered when one of the last two live replicas of the data object fails (transition from state 2 to state $1'$). Transitions from primed states to unprimed states correspond to recoveries of the replicated object. These recoveries occur when either the live replica that failed last recovers or the last live replica can gather the votes of a majority of the two excluded replicas of the object. States marked with two primes represent the four states of the object when the last two live

```
procedure RECOVER
begin
    repeat
        let $l$ be the recovering site
        $\langle R, \mathbf{o}, \mathbf{v}, \mathbf{P} \rangle \leftarrow \text{START}(U)$
        $Q \leftarrow \{r \in R : o_r = \max_{s \in R}\{O_S\}\}$
        $S \leftarrow \{r \in R : v_r = \max_{s \in R}\{V_S\}\}$
        $T \leftarrow R - P_m$
        $q = \frac{|P_m|}{2}$
        $t = \frac{|U - P_m|}{2}$
        choose any $m \in Q$
        if $|Q| > q \vee |Q| = q > 1 \wedge \max(P_m) \in Q \vee$
            $|Q| = q = 1 \wedge T \neq \emptyset \wedge (|T| > t \vee$
            $|T| = t \wedge \max(U - P_m) \in T)$ then
            if $v_l < v_m$ then
                copy the data object from site $m$
            fi
                $\text{COMMIT}(S \cup \{l\}, o_m + 1, v_m, S \cup \{l\})$
        else
            ABORT(R)
        fi
    until successful
end RECOVER
```

Figure 3: Recovery Algorithm

replicas of the object have both failed and zero to two replicas are comatose. The transition from state $2''$ to state 3 corresponds to the recovery of the replicated object when one of the last two live replicas recovers while the two excluded replicas of the object were comatose. Since ties will occur when one of the last two live replicas of the object while one of the other two replicas of the object is comatose, we needed to introduce two states $1''$. State $1''_+$ corresponds to the state of the replicated object when the excluded comatose replica precedes the other excluded replica in the lexicographic ordering of replicas; recovery is possible as soon as one of the last two live replicas of the object recovers. State $1''_-$ corresponds to the state of the replicated object when the excluded comatose replica follows the other excluded replica in the lexicographic ordering of replicas; recovery cannot complete until two additional replicas recover.

As one can see on figure 5, the state transition diagram for five replicas managed by the RDV protocol offers the same three-layer organization as the diagram for four replicas. Note that the replicated object will always have three excluded replicas when it becomes unavailable and that ties cannot occur.

Exact algebraic expressions for the availability of replicated data objects with four and five replicas managed by the RDV protocols were derived from these state transition diagrams using standard algebraic techniques and the symbolic manipulation program MACSYMA. These two expressions are quotients of very large polynomials in λ and μ and were not included in the paper because of their sizes.

Figures 6, 7, and 8 display the availabilities achieved by the RDV protocol with three, four and five replicas respectively. These availabilities are compared with those provided by majority consensus voting (MCV), dynamic voting (DV), dynamic-linear voting (DLV) and hybrid dynamic voting (HDV) for the same numbers of replicas. In all three graphs the failure rate to repair rate $\rho = \lambda/\mu$ varies between 0 and 0.2. Zero corresponds to perfectly reliable replicas and 0.2 to replicas that are repaired five times faster than they fail and have an individual availability of 5/6.

Figure 6 shows that for MCV, HDV and RDV have exactly the same availability for three replicas while DV per-

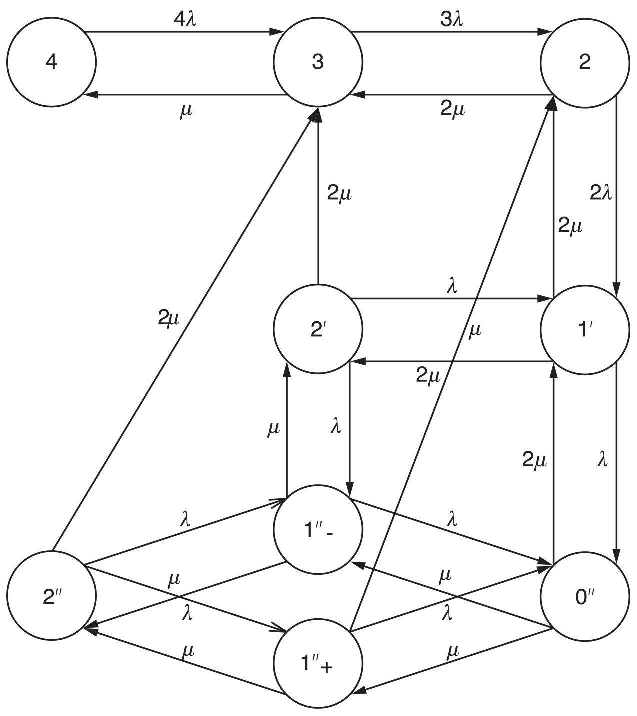

Figure 4: RDV State Transition Diagram for Four Replicas

forms somewhat worse and DLV somewhat better. This result is not surprising as MCV, HDV and RDV operate exactly in the same fashion when three replicas are present. Figures 7 and 8 show almost undistinguishable availabilities for DLV, HDV and RDV while DV performs significantly worse and MCV much worse.

Two important conclusions can be reached from this study. First, protecting replicated objects against irrecoverable media failures can be done at almost no cost in all environments where network partitions are possible as the two protocols achieving this objective perform almost as well as DLV, which is the best extant dynamic voting protocol. Second, the conjecture that HDV provides the greatest data availability among all dynamic voting protocols [8] was found not to hold when the availability of a data object was assumed to be the probability that it is accessible by at least one site.

These conclusions need to be qualified as they rely on the hypotheses introduced by our Markovian analysis. It was assumed that network partitions and other partial communication failures were possible but extremely unlikely. Failures and repairs were assumed to be independent processes with exponential distributions. Simultaneous failures of all sites holding replicas were not considered. The most controversial of these three assumptions is the exclusion of communication failures. Protocols that require two replicas to allow writes are inherently at a disadvantage against protocols that need only one replica. Hence RDV and HDV are likely to perform less well that DLV in environments where network partitions are likely. Extant simulation studies of the performance of replication control protocols have concluded that all other factors do not significantly alter the rankings of the protocols under study [2, 11, 12].

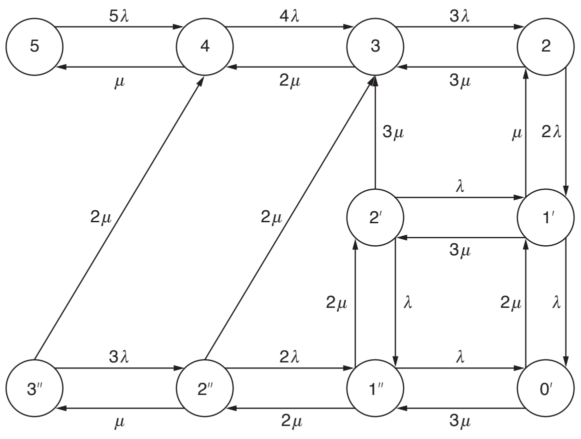

Figure 5: RDV State Transition Diagram for Five Replicas

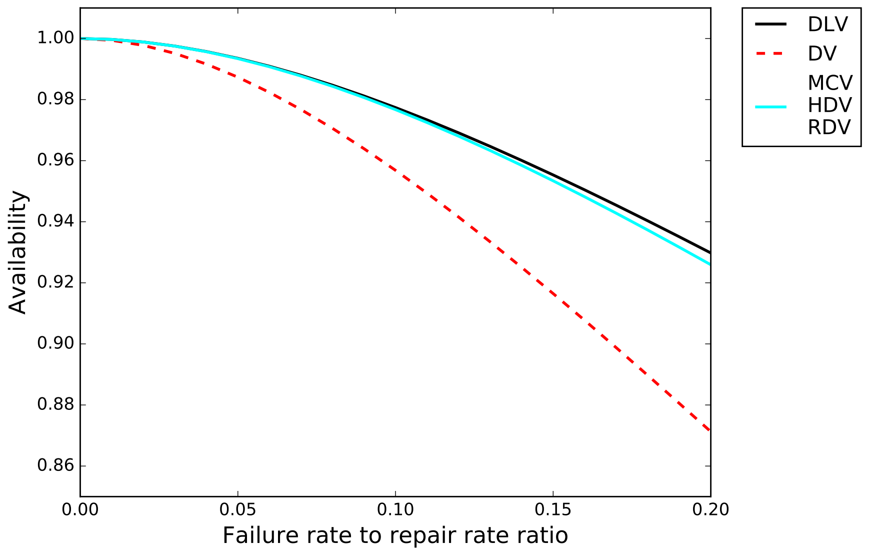

Figure 6: Compared Availabilities for Three Replicas

5 Possible Extensions

Several extensions of the RDV protocol are possible and we will briefly mention two here. First the RDV protocol could greatly benefit from combining it with regeneration as new replicas could be generated to replace the replicas that failed and increase the likelyhood that at least two current replicas of the object will always remain accessible [11, 13]. If the object is large and writes only affect small portions of its contents, temporary write logs could be generated instead of full replicas to reduce regeneration costs.

Second the RDV protocol would be unduly restrictive in environments where network partitions are known

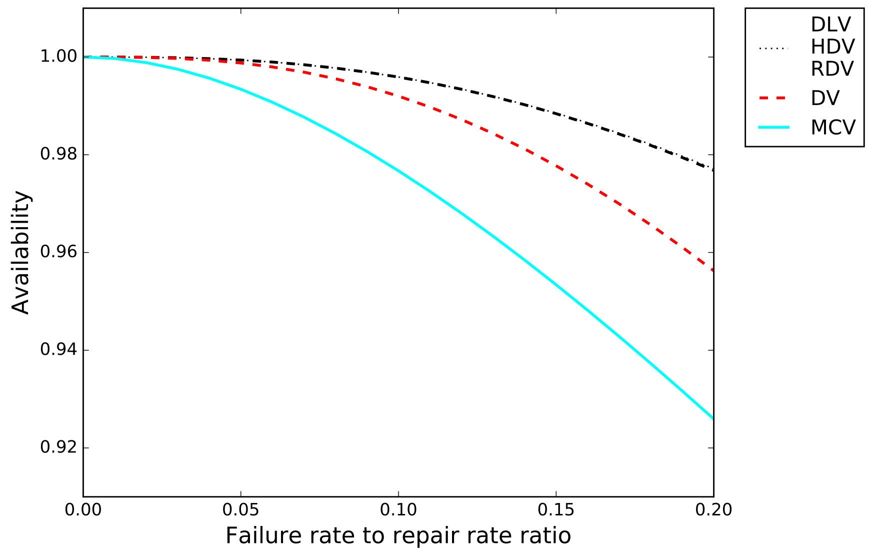

Figure 7: Compared Availabilities for Four Replicas

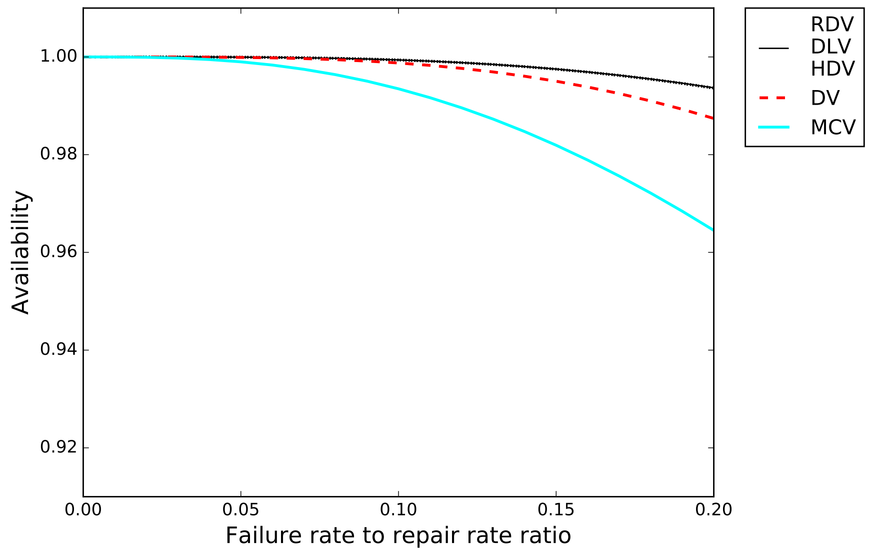

Figure 8: Compared Availabilities for Five Replicas

to be impossible. One could envision a *robust available copy* protocol (RAC) tailored to these environments and guaranteeing like RDV that all writes to the object are recorded in at least two replicas. Such protocol would differ from RDV in two important features:

1. the current partition set P would be required to contain a majority of the replicas in the previous partition set; partition sets would instead be called *available sets* and a new available set only required to contain one live replica from the previous available sets; and

2. reads would be allowed as long as one live replica of the replicated object remains accessible although writes would continue to require two live replicas.

6 Conclusion

The last five years have seen the development of very efficient replication control protocols requiring much less replicas to provide the same level of data availability as older protocols such as majority consensus voting. In this paper, we have described a replication control protocol protecting replicated data objects against irrecoverable failures by requiring all writes to be performed on at least two replicas.

A Markov model was used to evaluate the availability of replicated data objects managed by the RDV protocol and compare it with those of replicated objects managed by several voting protocols. Our study indicated that protecting replicated objects against irrecoverable media failures can be done at almost no cost in all environments where network partitions are possible as the protocols achieving this objective perform almost as well as the best extant dynamic voting protocol.

Acknowledgements

Jehan-François Pâris was supported in part by a grant from the NCR Corporation and the University of California MICRO program. Darrell Long was supported in part by a University of California Regents Junior Faculty Fellowship. We are grateful to Elizabeth Pâris and Mary Long for their editorial comments.

The Markov analysis of the availability of the protocols under study 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

References

[1] P. A. Bernstein and N. Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Database Systems (TODS)*, 9(4):596–615, 1984.

[2] J. L. Carroll and D. D. E. Long. The Effect of Failure and Repair Distributions on Consistency Protocols for Replicated Data Objects. In *Proceedings of the 22nd annual symposium on Simulation*, pp. 47–60. IEEE Computer Society Press, 1989.

[3] D. Davcev. A Dynamic Voting Scheme in Distributed Systems. *IEEE Transactions on Software Engineering*, 15(1):93–97, 1989.

[4] D. Davcev and W. A. Burkhard. Consistency and Recovery Control for Replicated Files. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles (SOSP '85)*, pp. 87–96, 1985.

[5] C. A. Ellis. Consistency and Correctness of Duplicate Database Systems. *ACM SIGOPS Operating Systems Review*, 11(5):67–84, 1977.

[6] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles (SOSP '79)*, pp. 150–162. ACM, 1979.

[7] B. V. Gnedenko. *Mathematical Methods of Reliability Theory*. Moscow, English Translation, New York , Academic Press, 1968.

[8] S. Jajodia and D. Mutchler. Integrating Static and Dynamic Protocols to Enhance File Availability. In *Proceedings of the Fourth International Conference on Data Engineering*, pp. 144–154. IEEE, 1988.

[9] S. Jajodia and D. Mutchler. Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. *ACM Transactions on Database Systems (TODS)*, 15(2):230–280, 1990.

[10] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. Replication in the Harp File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, pp. 226–238. ACM, 1991.

[11] D. D. E. Long and J.-F. Pâris. A Realistic Evaluation of Optimistic Dynamic Voting. In *Proceedings of the Seventh Symposium on Reliable Distributed Systems (SRDS '88)*, pp. 129–137, Columbus, Oct. 1988. IEEE.

[12] J. Noe and A. Andreassian. Effectiveness of Replication in Distributed Computing Networks. In *Proceedings of the Seventh International Conference on Distributed Computing Systems*, pp. 508–513, 1987.

[13] J.-F. Pâris. Voting with Witnesses: A Constistency Scheme for Replicated Files. In *Proceedings of the Sixth International Conference on Distributed Computing Systems (ICDCS '86)*, pp. 606–612, 1986.

[14] J.-F. Pâris and D. D. E. Long. Efficient Dynamic Voting Algorithms. In *Proceedings of the Fourth International Conference on Data Engineering*, pp. 268–275. IEEE, 1988.

[15] J. F. Pâris and D. D. E. Long. On the performance of available copy protocols. *Performance Evaluation*, 11, 1990.

[16] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of Replicated Objects: A Technique and Its Eden Implementation. In *Proceedings of the Second International Conference on Data Engineering*, pp. 175–187. IEEE Computer Society, 1986.

[17] R. H. Thomas. A Majority Consensus Approach to Concurrency Control. *ACM Transactions on Database Systems (TODS)*, 4(2):180–209, 1979.

[18] K. S. Trivedi. *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[19] R. Van Renesse and A. S. Tanenbaum. Voting with Ghosts. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, pp. 456–462. IEEE, 1988.