Proc. 10th International Phoenix Conference on Computers and Communication (1991), to appear

Resilient Memory-Resident Data Objects

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

Abstract: Data replication has been widely used to build resilient data objects. These objects normally consist of several replicas stored in stable storage and a replication control protocol managing these replicas. Replicated data objects incur a significant penalty resulting from the increased number of disk accesses. We investigate the feasibility of replicated data objects consisting of several memory-resident replicas and one append-only log maintained on disk.

We analyze, under standard Markovian hypotheses, the availability of these data objects when combined with three of the most popular replication control protocols: *available copy* (AC), *majority consensus voting* (MCV) and *dynamic-linear voting* (DLV). We show that replicated objects consisting of *n* memory-resident replicas and a disk-resident log have almost the same availability as replicated objects having *n* disk-resident replicas.

Keywords: file replication, replicated databases, memory-resident databases, majority consensus voting.

1. INTRODUCTION

The last ten years have seen a dramatic reduction of the cost per bit of semi-conductor memory. As a result, it has become possible to contemplate the permanent storage of large files or complete databases in main memory. Memory-resident files and memory-resident databases are an attractive alternative to disk-based files and databases since they do not experience the long delays associated with disk accesses. This speed advantage is extremely important for database management systems as much of the run-time performance bottlenecks within current systems can be attributed to disk access delays.

Access times are not the only difference between memory-resident data objects and their disk-resident counterparts. First, main memory, unlike disk, is a volatile medium. A memory-resident data object will be completely lost in case of a power failure. A backup copy of each file or database must be maintained in stable storage to allow the restoration of the memory-resident copy. Second, memory resident data objects can allow simpler algorithms to be used. The algorithms used to manage the data structures for memory-resident data objects do not need to be concerned with minimizing disk accesses and using disk space efficiently. Studies of the impact of memory-residency on various components of conventional database management systems have concluded that the use of memory-based instead of disk-based systems was an important factor in the performance of memory-resident systems [BHT87, DeWi84, Eich87, Hagm86, LeCa86a, LeCa86b, LeCa87, GMSa88, SaGM89].

Replicated file systems and replicated database systems are excellent candidates for the usage of memory-resident objects. First, these systems tend to be slower than their single-copy counterparts [BMP89, Saty90]. Any technique that reduces their access times would significantly contribute to their wider usage. Second, the performance of these systems could be greatly improved if some of their components were allowed to reside, even temporarily, on diskless sites. Third, a replicated file or a replicated database has copies of its data stored on different sites. Some of these copies—or *replicas*—could be less resilient than others without significantly affecting the overall resiliency of the object. Finally, memory-resident objects may be necessary to handle the very high I/O rates expected from networks and processors 100 to 1,000 faster than these currently in use [OuDo89].

The simplest way to include memory-resident objects in a replicated file system or a replicated database system would be to consider each pair (*main-memory copy*, *back-up copy*) as an atomic entity implementing the abstraction of a non-volatile copy. This approach fails to capitalize on the redundancy already existing within each pair and precludes the use of memory-resident replicas without backup copies. This paper focuses instead on the feasibility of replicated data objects including fully memory-resident replicas and show how extant replication control protocols can be modified to accommodate these objects.

Section 2 of this paper reviews some extant work on replication control protocols. Section 3 introduces replication control protocols tailored to replicated objects consisting of several memory-resident replicas and one append-only log maintained on disk. Section 4 presents a Markov analysis of their availability. Finally section 5 has our conclusions.

2. RELATED RESEARCH

Data are often replicated in distributed systems for protection against site failures or network partitions. When this is done, an access policy must be chosen to insure a consistent view of the data so that it appears as though there were only a single copy of the data. The view presented to the user must remain consistent even in the presence of site failures and network partitions. This task is normally the responsibility of the *consistency control* or *replication control* protocol.

When network partitions are known to be impossible, the *available copy* (AC) protocol [BeGo84] and its variants [PaLo90] provide a simple means for maintaining data consistency. The AC protocol provides the best availability of any non-regenerative replication control protocol. The AC protocol is based on the observation that since partitions are impossible the replicas that have participated in all writes must hold the most recent version of the data. 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, greatly reducing communication costs. Replicas recovering from a failure can repair immediately if there is a current version of the data available. After a failure of all replicas, the recovering sites must wait 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) [Elli77, Thom79] and *weighted voting* [Giff79], 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 [Giff79]. 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. The primary disadvantage of voting is that it requires at least three replicas 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 [PaLo90].

Unlike MCV, the *dynamic voting* (DV) protocol [BGS89, DaBu85] automatically adjusts its access quorum to changes in the state of the network. When 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 reintegrated. To keep track of the status of the replicated object, every replica maintains some state information. This information depends on the implementation, but will include a *version number* identifying the last write recorded by the replica and either a *partition vector* [DaBu85], a count representing the number of sites that participated in the last update [JaMu87], or a *partition set* and an *operation number* [PaLo88] 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 sites is equal to the number of current replicas not in communication. The DV protocol then declares the replicated object to be inaccessible. An extension proposed by Jajodia [Jajo87], known as *dynamic-linear voting* (DLV) resolves these ties by applying a total ordering to the sites. This simple improvement greatly enhances the availability of the replicated data.

Like most extant replica control protocols, these four protocols only apply to replicated objects whose replicas are stored in stable storage. Gifford's *weighted voting* has a provision for replicas stored in volatile storage but these *weak representatives* are always assigned zero votes and are therefore excluded from quorum computations.

3. REPLICATED MEMORY-RESIDENT OBJECTS

Memory-resident objects are particularly vulnerable to site failures since these events normally result in the irrecoverable loss of all memory-resident data. Memory-resident objects that need to survive site failures must include a back-up copy in stable storage that reflects the current value of the object. There have been several recent proposals to organize backups as logs of updates [OuDo89, SaGM89]. One of them, made by Ousterhout and Douglis, concerns memory-resident file systems. Its authors observe that the need to record updates in stable storage is the performance bottleneck of any memory-resident file system. They propose to use redundant arrays of inexpensive disks (RAID) [PGK88] and to represent backups as append-only logs to maximize write overlaps and to eliminate seek times. Another proposal by Salem and Garcia-Molina maintains database backups as a sequence of checkpoints supplemented by a transaction log. Both proposals have interesting implications for replication control since log-structured backups can be regenerated or continued after a partial failure.

Consider for instance the case of a replicated object with three memory-resident replicas and two disk-resident back-ups and assume that the object is managed by some variant of the MCV protocol. Since five replicas of the object are present, read and write quorums could be set to three. This quorum assignment would however allow writes to proceed when both backups are unavailable and the three memory-resident replicas remain accessible. Such a situation should be avoided as it might lead to the irrecoverable loss of these writes.

A first solution to the problem consists of requiring all write quorums to include at least one backup copy. New backups can even be regenerated to replace backups that have become unavailable in the same fashion as failed replicas are replaced by new ones in Pu's *regeneration algorithm* [PNP88]. This approach has the disadvantage of either requiring more than one backup or introducing delays in write operations while a failing backup is regenerated.

Log-structured back-ups offer the advantage of operating in append-only mode. Hence if the site currently holding the log fails, the log can be continued on any site that has disk space available. Should this site fail at a later time, the log could be continued on a third site and so on. This technique allows access to the replicated object as long as an operational disk unit can be found to host the current log. It never requires the transfer to disk of a full copy of the replicated object and does not delay writes for more time than it takes for electing a new *current logging site* for the object. Care must be taken to ensure that a full log can be reassembled during the recovery stage after a failure of all memory resident replicas of the data object. This makes it necessary to start every new log fragment by a special block identifying the log fragment and containing a *log fragment number* as well as a list of all sites that contain previous fragments of the log.

The only availability penalty occurs after the failure of all memory-resident replicas of the data object when the complete log is reassembled from all the log fragments stored at the previous logging sites. This process may involve waiting for the recovery of all sites that could have held a log fragment. It can sometimes be accelerated

(i) by keeping a consistent record in stable storage of the current logging site—using the same techniques as used by AC protocols to record the last site that failed—or

(ii) by taking checkpoints from time to time of the state of the replicated object.

For example, we observe that many ''failures'' are actually orderly shut-downs for maintenance purposes. When these occur, the volatile replicas stored at that site could be written to disk, thus speeding recovery when the site returns to operation.

Replication control protocols for replicated objects consisting of n memory-resident replicas and one disk-based log can be easily derived from extant protocols handling disk-resident replicas by modifying them in the following way:

(i) let the protocol operate exactly as before as long as at least current logging site and at least one memory-resident replica remain available;

(ii) should the current logging site become unavailable, search for a new logging site and disable all write requests until such a site can be found;

(iii) should all memory-resident replicas become unavailable, suspend the operation of the protocol until the current value of the replicated object can be reconstituted and a sufficient number of memory-resident replicas regenerated to leave the object in an available state.

Consider for instance the case of a replicated object with n memory-resident replicas managed by an AC protocol. Assume that m of the n sites holding a replica of the object have a disk unit and are capable of acting as the current logging site. The replicated object will remain available for reads and writes as long as one of these m sites remains operational. Should the last one of these m sites fail, the object will remain available for reads but not for writes as long as one of the $n-m$ diskless sites unit remains operational. After a failure of all n memory-resident replicas, the object will remain unavailable until a new memory-resident replica can be regenerated. Since this task will require the reconstitution of a full log of all writes to the object, it may involve waiting for the recovery of all m sites capable to have acted as a current logging site.

A DV or MCV protocol using memory-resident replicas would allow read or write access to the replicated object as long as a quorum of the replicas are present and there is an available logging site. Read access could continue as long as a quorum of the replicas was present even if the logging site were lost. Write access could continue as soon as a logging site became available. As with the AC protocol, both DV and MCV might have to wait for all m logging sites in order to reconstitute the replicated object in the event of a total failure.

An AC or a DLV protocol modified to handle memory-resident replicas only need to regenerate one memory-resident replica of the object. A modified MCV protocol will have to regenerate a majority of the original number of replicas.

Despite its obvious advantage of requiring only one disk-based log without incurring any regeneration delays, the approach we have sketched has a few limitations which need to be stated. First, it only applies to fully memory-resident replicated objects. Second, it increases the time the object will take to recover from a total failure of all its replicas owing to the reconstruction from the log. This can be mitigated by periodic checkpoints, and in any case we expect total failures to be rare events. Finally, it increases the probability of a non-recoverable error by distributing its single copy of the log among several sites. In this case, it might be prudent to introduce some redundancy in the log or to have it replicated at more than one site. This might in turn involve some further refinements of the DLV protocol to ensure that two disjoint sets of sites could not independently reconstitute the current state of the file and establish competing majority blocks. The problem does not exist with the AC protocol since network partitions are excluded nor with the MCV protocol since it uses a static quorum.

4. STOCHASTIC ANALYSIS

In this section we present an analysis of the availability provided by replicated memory-resident data objects and compare it to that achieved by replicated disk-resident objects. Our study focuses on replicated objects managed by the *available copy*, *majority consensus voting* and *dynamic-linear voting* protocols. Our model consists of a set of sites with independent failure modes connected via a network that does not fail. We assume that each site has enough main memory to store a full replica of the replicated data object and enough disk space to become at any time the current logging site for the object. To simplify our model, we also assume that no attempts are made to store the name of the current logging site or of any of its predecessors in stable storage. As a result, the recovery of a replicated data object after a total failure will always require the recovery of all sites that hold or have held a replica of the object.

When a site fails, a repair process is immediately started 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 λ and that repairs are exponentially distributed with mean μ. Irrecoverable errors are specifically excluded. The system is assumed to exist in statistical equilibrium and to be characterized by a discrete-state Markov process.

The assumptions that we have made are required for a steady-state analysis to be tractable. The assumptions about failure and repair distributions are required to preserve the Markov property and thus keep the number of states finite. If the network is allowed to partition then the number of topologies that must be analyzed is greater than exponential. Thus, this analysis gives an optimistic view of the availability of the replicated objects under consideration.

Definition 4.1. *The availability of a replicated data object consisting of n replicas and managed by a consistency protocol P, denoted $A_P(n)$, is the stationary probability of the system being in a state permitting access.*

4.1. Available Copy Protocols

As seen on figure 1, the state-transition-rate diagram for a replicated object having n disk-resident replicas has $2n$ states [PaLo90]. The first n states labeled from 1 to n represent the states of the file when 1 to n replicas are

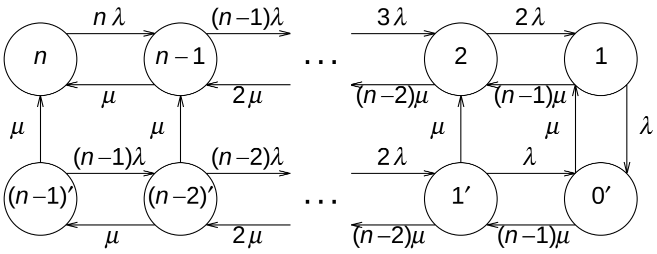

Figure 1: ***n*** **Disk-Resident Replicas Managed by AC**

available; the n other states labeled from $0'$ to $n-1'$ represent the states of the replicated object after all replicas have failed and when 0 to $n-1$ sites holding obsolete replicas of the object have recovered. All transitions from primed states to a non-primed state represent the recovery of the site holding the last available replica of the data object. Primed states correspond to situations where the file remains unavailable.

The availability $A_{AC}(n)$ of the replicated object is given by

$$A_{AC}(n) = 1 - \sum_{0}^{n-1} \bar{p}_i = 1 - \sum_{i=0}^{n-1} \frac{C_{n-i-1}}{C_{n-1}} \frac{\rho^n}{(1+\rho)^n},$$

with

$$C_0 = 1,$$

$$C_1 = (n-1)\rho + 1,$$

and

$$C_k = \frac{(n-k)\rho}{k+1} C_{k-1} - \frac{n-k+1}{k} C_{k-2}$$

for $k > 1$.

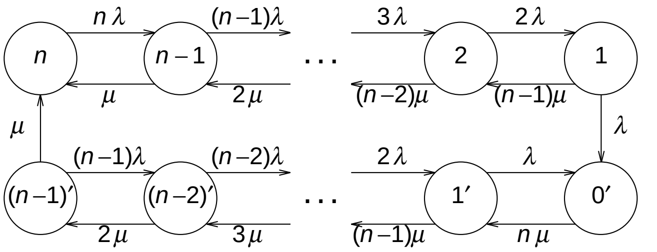

Figure 2: ***n*** **Memory-Resident Replicas Managed by AC or MCV**

When the n disk-resident replicas are replaced by n memory-resident replicas and one disk-resident log, the replicated object behaves exactly as before as long as one replica remains available. When that last available replica has failed, the object will remain unavailable until all sites holding replicas have recovered and a complete log can be reassembled. The AC protocol applied to n memory-resident replicas and one disk-resident log behaves identically to a *naive available copy* protocol (NAC) [PaLo90] applied to n disk-resident replicas. As seen on figure 2, the new state-transition-rate diagram has the same $2n$ states as if the replicas were memory-resident. Transitions between states will be quite similar to those observed before with the exception that the only transition from a primed unavailable state to a non-primed available state is from state $(n-1)'$ to state n.

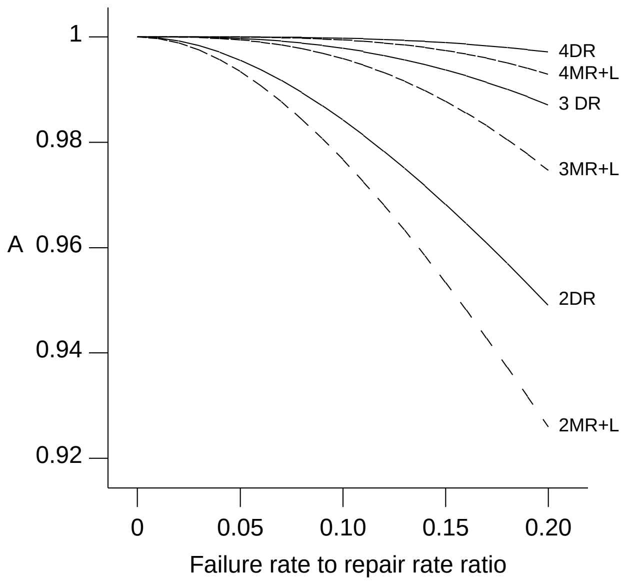

Figure 3: Compared Availabilities for AC

The availability $A_{AC}(n, 1)$ of a replicated object with n memory-resident replicas and one disk-resident log is then given by

$$A_{AC}(n, 1) = A_{NAC}(n) = \sum_{k=1}^{n} p_k = \frac{B(n,\rho)}{B(n,\rho) + \rho B(n, \frac{1}{\rho})}$$

where

$$B(n,\rho) = \sum_{k=1}^{n} \sum_{j=1}^{k} \frac{(n-j)!\,(j-1)!}{(n-k)!\,k!} \rho^{j-k}$$

and $\rho = \lambda/\mu$ is the failure rate to repair rate ratio [PaLo90].

The graph on figure 3 displays the compared availabilities of replicated objects managed by the AC protocol for values of $\rho \le 0.2$. This upper bound corresponds to the failure rate to repair rate ratio of a site that would be unavailable for four hours every day or for thirty-three hours and thirty-six minutes every week. Solid lines are used to represent the availabilities of replicated objects consisting of two, three or four disk-resident replicas while dotted lines are used for objects consisting of two, three or four memory-resident replicas and one disk-resident log. The graph shows that the availabilities afforded by n memory-resident replicas and one disk-based log remain comparable to these achieved with n disk-resident replicas. This was not an unexpected result since the AC protocol applied to n memory-resident replicas and one disk-resident log was expected to behave exactly as a NAC protocol (NAC) applied to n disk-resident replicas.

Most of today's computers are characterized by availabilities well above 0.95 and by values of the failure rate to repair rate ratio ρ well below 0.05, which could lead us to the conclusion that memory-resident replicas supplemented by a single disk-resident log would perform as well as the disk-resident replicas. Besides, observed repair time distributions are characterized by coefficients of variation less than one. Under such conditions, sites will tend to recover in the same order as they failed. The last site to recover after a total failure will often be the last one that failed. When this happens, data objects using disk-resident replicas will be unable to recover faster than data objects using memory-resident replicas.

4.2. Majority Consensus Voting

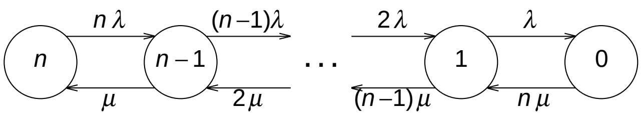

Figure 4: ***n*** **Disk-Resident Replicas Managed by MCV**

As seen on figure 4, the state-transition-rate diagram for a replicated object with n disk-resident replicas managed by the MCV protocol has n states labeled from 0 to n and denoting the current number of replicas available when the object is in that state. If p_j represents the probability that the object is in state j, the availability $A_{MCV}(n)$ of the object is given by

$$A_{MCV}(n) = \sum_{j=\lceil n/2 \rceil}^{n} p_j = \sum_{j=\lceil n/2 \rceil}^{n} \frac{\binom{n}{n-j} \rho^{n-j}}{(1+\rho)^n}$$

when n is odd and by

$$A_{MCV}(n) = \sum_{j=n/2+1}^{n} p_j + \frac{1}{2} p_{\frac{n}{2}}$$

which simplifies into $A_{MCV}(n-1)$ when n is even [Pari86].

A replicated object consisting of n memory-resident replicas and one disk-based log managed by the MCV protocol will remain available so long as a majority of its replicas remain accessible in order to allow access to the replicated object. Recovery after a failure of all memory-resident replicas will be treated in the same fashion as for the AC protocol since no replica can be repaired until all sites holding replicas and possible fragments of the disk-resident log have recovered. Replicated object consisting of memory-resident replicas and a disk-resident log managed by the MCV protocol have therefore the same state-transition-rate diagram as if they were managed by the AC protocol. Their availability figures are however different since the protocol requires a majority of the replicas to be accessible.

The graph on figure 4 displays the compared availabilities of replicated objects managed by the MCV protocol. Solid lines are used to represent the availabilities of replicated objects consisting of three, four or five disk-resident replicas while dotted lines are used for objects consisting of three, four or five memory-resident replicas and one disk-resident log. For all three numbers of replicas investigated, the graph fails to show any significant difference between the availabilities afforded by n memory-resident replicas and one disk-based log and those achieved with n disk-resident replicas. The behavior of configurations including even numbers of memory-resident replicas are also be worth mentioning. Configurations consisting of even numbers of disk-based replicas are known not to perform better than configurations consisting of one less replicas because of the need to resolve ties [Pari86]. While the same ties also occur with configurations consisting of even numbers of memory-resident replicas, they have a slightly higher availability than configurations consisting of one less replica since the presence of one extra replica decreases the probability of experiencing a simultaneous failure of all replicas.

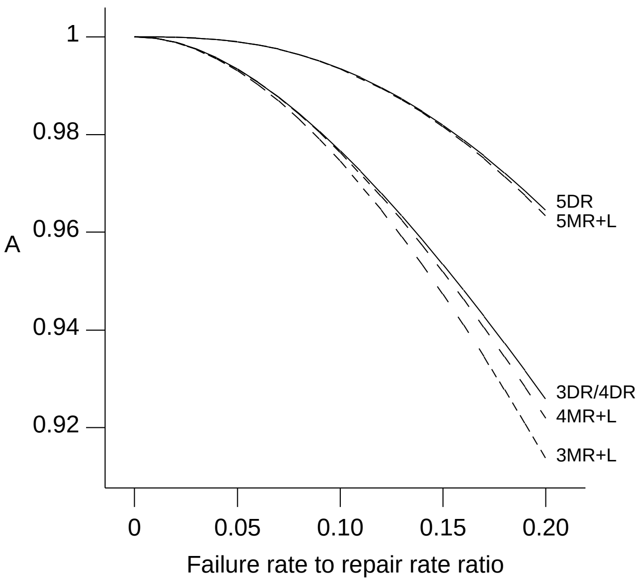

Figure 5: Compared Availabilities for MCV

4.3. Dynamic-Linear Voting

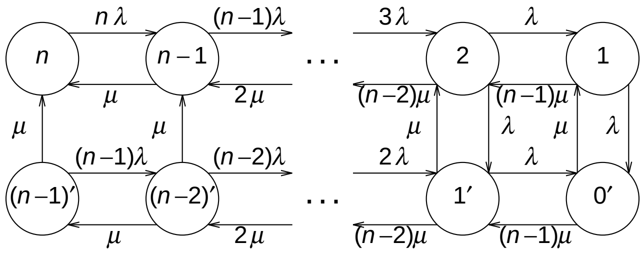

Figure 6: ***n*** **Disk-Resident Replicas Managed by DLV**

As seen on figure 6, the state-transition-rate diagram for a replicated object consisting of n disk-resident replicas managed by the DLV protocol has $2n$ states [PaBu86]. States 1 to n represent the state of the replicated object when 1 to n replicas are accessible and these replicas were the majority of the previous majority block. States 1′ to $(n-1)'$ represent the state of the replicated object when 1 to $n-1$ replicas are accessible but do not constitute a majority of the current majority block. State 0′ indicates that all replicas of the object have failed and no site holding any replica has yet recovered. The transitions from available non-primed states to unavailable primed states correspond to the two situations where either (i) one of the two last accessible replicas of the object fails and the surviving replica follows the replica that failed in the linear ordering of all replicas, or (ii) the last accessible replica of the object fails. The availability $A_{DLV}(n)$ of a replicated object with n disk-resident replicas managed by the DLV protocol is given by

$$A_{LDV}(n) = \sum_{i=1}^{n} p_i$$

where p_i is the probability of being in state i.

As seen on figure 7, $n-1$ new states are to be added to the state transition diagram of figure 6 when the n disk-resident replicas are replaced by n memory-resident

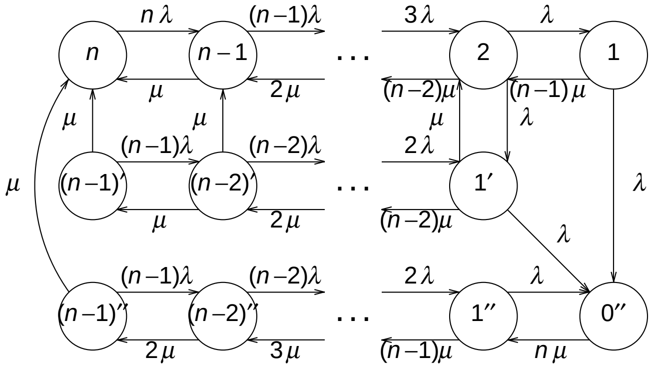

Figure 7: ***n*** **Memory-Resident Replicas Managed by DLV**

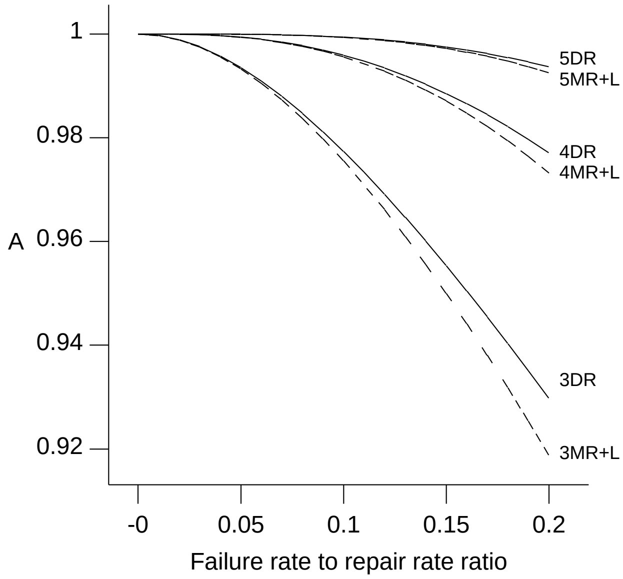

Figure 8: Compared Availabilities for DLV

replicas and one disk-based log. These states labeled from 1″ to $(n-1)''$ represent the states of the replicated object after all replicas have failed and when 1 to $n-1$ sites holding replicas of the object have recovered. State 0′, now relabeled 0″, has no outgoing transition to state 1 since the recovery of a replicated object after a total failure requires the recovery of all sites holding replicas.

The graph on figure 8 displays the compared availabilities of replicated objects managed by the DLV protocol. Solid lines are used to represent the availabilities of replicated objects consisting of three, four or five disk-resident replicas while dotted lines are used for objects consisting of three, four or five memory-resident replicas and one disk-resident log. As for the MCV protocol, the graph fails to show any significant difference between the availabilities afforded by *n* memory-resident replicas and one disk-based log and those achieved with *n* disk-resident replicas. A comparative study of figures 5 and 8 also shows the dramatic improvement of object availability from MCV to DLV for four replicas. A replicated object with four memory-resident replicas and one disk-resident log managed by a DLV protocol has indeed a higher availability than a replicated object with five disk-resident replicas managed by MCV.

5. CONCLUSION

Memory-resident files and memory-resident databases are an attractive alternative to disk-based files and databases since they do not experience the long delays associated with disk accesses. We have investigated the feasibility of replicated data objects consisting of several memory-resident replicas backed-up by a single append-only log maintained on disk.

First, we have shown that such objects can be managed by simple variants of the most popular replication control protocols for disk-resident replicated objects. Second, we have analyzed, under standard Markovian hypotheses, the availability of replicated objects consisting of memory-resident replicas and a single append-only log on disk and shown that they have almost the same availability as replicated objects having all their replicas residing on disk.

Using memory-resident replicas has the advantage of faster access over disk-resident replicas. The cost in an increase in recovery time to reconstruct from the log. We suggested several improvements that could be made to speed the recovery of our protocols. A more complete analysis would take these improvements into account. Even so, our analysis shows that memory-resident replicas provide a level of fault-tolerance comparable to disk-resident replicas.

Further work is still needed to evaluate the availability and reliability of replicated memory-resident data objects under failure conditions including network partitions and to confirm our preliminary conclusion that a single disk-resident log seems to provide an acceptable rate of recovery after a failure of all sites holding replicas.

Acknowledgements

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

BGS89 D. Barbara, H. Garcia-Molina, and A. Spauster, ''Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment,'' *ACM Trans. on Computer Systems,* 7, 4 (1989), pp. 394-426.

BeGo84 P.A. Bernstein and N. Goodman, ''An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases,'' *ACM Trans. on Database Systems,* 9 (4) (1984) 596-615.

BHT87 D. Bitton, M. B. Hanrahan and C. Turbyfill, ''Performance of Complex Queries in Main Memory Database Systems,'' *Proc. 3rd Int. Conf. on Data Engineering* (1987), pp. 72-81.

BMP89 W. A. Burkhard, B. E. Martin and J.-F. Pâris, ''The Gemini Replicated File Test-bed,'' *Information Science,* 48, 2 (1989), pp. 119-134.

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

DeWi84 D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker and D. Wood, ''Implementation Techniques for Main Memory Database Systems,'' *Proc. ACM SIGMOD 1984 Annual Conf.* (1984), pp. 1-8.

Eich87 M. Eich, ''A Classification and Comparison of Main Memory Database Recovery Techniques,'' *Proc. 3rd Int. Conf. on Data Engineering* (1987), pp. 332-339.

Elli77 C. A. Ellis, ''Consistency and Correctness of Duplicate Database Systems,'' *Operating Systems Review,* (11) (1977).

Hagm86 R. B. Hagman, ''A Crash Recovery Scheme for a Memory-Resident Database System,'' *IEEE Trans. on Computers,* C-35 (9) (1986), pp. 839-843.

Giff79 D. K. Gifford, ''Weighted Voting for Replicated Data,'' *Proc. 7th ACM Symposium on Operating System Principles* (1979), pp. 150-161.

GMSa88 H. Garcia-Molina and K. Salem, ''System M: A Transaction Processing System for Memory Resident Data,'' Technical Report CS-TR-195-88, Department of Computer Science, Princeton U. (1988).

Jajo87 S. Jajodia, ''Managing Replicated Files in Partitioned Distributed Database Systems,'' *Proc. 3rd Int. Conf. on Data Engineering,* (1987), pp. 412-418.

JaMu87 S. Jajodia and D. Mutchler. ''Dynamic Voting.'' *Proc. ACM SIGMOD,* (1987), pp. 227-238.

LeCa86a T. J. Lehman and M. J. Carey, ''Query Processing in Main Memory Database Management Systems,'' *Proc. ACM SIGMOD 1986 Annual Conf.* (1986), pp. 239-250.

LeCa86b T. J. Lehman and M. J. Carey, ''A Study of Index Structures for Main Memory Database Management Systems,'' *Proc. 12th Int. Conf. on VLDB* (1986), pp. 294-303.

LeCa87 T. J. Lehman and M. J. Carey, ''A Recovery Algorithm for a High-Performance Memory-Resident Database System,'' *Proc. ACM SIGMOD 1987 Annual Conf.* (1987), pp. 104-116.

OuDo89 J. Ousterhout and F. Douglis, ''Beating the I/O Bottleneck: A Case for Log-Structured File Systems,'' *Operating Systems Review,* 23 (1) (1989), pp. 11-28.

PaBu86 J.-F. Pâris and W. A. Burkhard, ''On the Availability of Dynamic Voting Schemes,'' Technical Report CS86-94, Department of CSE, University of California, San Diego (1986).

PaLo88 J.-F. Pâris and D. D. E. Long, ''Efficient Dynamic Voting Algorithms,'' *Proc. 4th Int. Conf. on Data Engineering,* (1988), pp. 268-276.

PaLo90 J.-F. Pâris and D. D. E. Long ''On the Performance of Available Copy Protocols,'' *Performance Evaluation, 11 (1990), pp. 9-30.*

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

PGK88 D. Patterson, G. Gibson and R. H. Katz, ''A Case for Redundant Arrays of Inexpensive Disks (RAID),'' *Proc. ACM SIGMOD 1988 Annual Conf.* (1988), pp. 109-116.

SaGM89 K. Salem and H. Garcia-Molina, ''Checkpointing Memory-Resident Databases,'' *Proc. 5th Int. Conf. on Data Engineering* (1989), pp. 452-462.

PNP88 C. Pu, J. D. Noe and A. Proudfoot, ''Regeneration of Replicated Objects: A Technique and its Eden Implementation,'' *IEEE Trans. on Software Engineering,* SE-14 (7) (1988), pp. 936-945.

Saty90 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 Distributed Workstation Environment,'' *IEEE Trans. on Computers,* 39, 4 (1990), pp. 447-459.

Thom79 R. H. Thomas, ''A Majority Consensus Approach to Concurrency Control,'' *ACM Trans. on Database Systems* 4 (1979) pp. 180-209.