Regeneration Protocols for Replicated Objects

Darrell D. E. Long[1] *Jehan-François Pâris*[2]

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

ABSTRACT

The reliability and availability of replicated data can often be increased by generating new replicas when some become inaccessible due to system malfunctions. This technique has been used in the *Regeneration Algorithm,* a replica control protocol based on file regeneration.

The read and write availabilities of replicated data managed by the Regeneration Algorithm are evaluated and two new regeneration protocols are presented that overcome some of its limitations. The first protocol combines regeneration and the *Available Copy* approach to improve availability of replicated data. The second combines regeneration and the *Dynamic Voting* approach to guarantee data consistency in the presence of network partitions while maintaining a high availability. Expressions for the availabilities of replicated data managed by both protocols are derived and found to improve significantly on the availability achieved using extant consistency protocols.

Keywords: file consistency, fault-tolerant systems, replicated files, mutual exclusion, performance evaluation.

1. INTRODUCTION

Recent advances in computer network technology and reductions in the cost of storage media have made the replication of important data on several sites of a local area network a cost-effective proposition. The management of replicated data is a cumbersome task, complicated by site failures and network partitions that may introduce inconsistencies between replicas of the same object. Special *consistency control protocols* have been devised to insure that the users of the replicated data are always provided with a single consistent view of all replicated objects.

A number of these protocols have been described in the literature [1-7,12-18]. Most of these protocols operate by disallowing accesses to the replicated data when some conditions on the states of the replicas are not met.

Pu [15-17] has recently proposed a consistency control protocol taking a more active approach. The *Regeneration Algorithm* regenerates new replicas when it detects that one or more of the replicas have become inaccessible due to system malfunctions. While reads are allowed under their protocol so long as one current replica of the object remains accessible, writes are disabled if fewer than the initial number of replicas are accessible and there are not enough spares for the missing replicas to be regenerated. No provisions are made by the algorithm for enforcing mutual exclusion or for recovering from a total failure.

The concept of regeneration is appealing because replicas can usually be generated at a much faster rate than sites can be repaired. Previously, a similar idea had been used in a modified majority consistency voting protocol proposed by one of the authors [13]. The protocol applies to replicated data consisting of n full replicas and m witnesses. Witnesses have most of the attributes of full replicas, contain all of the file state information and participate like full replicas in the collection of quorums but hold no data [12]. They can be easily converted into full replicas without requiring any adjustment of the read and write quorums. Unlike the Regeneration Algorithm, this protocol provides mutual exclusion and guarantees file consistency in the presence of network partitions. The decision to use static quorums in the protocols did not allow changes in the total number of replicas and witnesses.

Although the Regeneration Algorithm does not suffer from this lack of flexibility, it has its own limitations. As Noe and Andreassian suggested in their study of the effectiveness of replication [10], write availabilities could be significantly improved by applying the philosophy of the *Available Copy* protocol [2] to the Regeneration Algorithm. Under their proposal, missing replicas would be replaced by new replicas whenever feasible but writes would continue to be allowed even if some or all of the missing replicas could not be regenerated.

Another limitation of the Regeneration Algorithm is its inability to guarantee data consistency in the presence of network partitions. Many local area networks consist of several carrier-sense segments or token rings linked by selective repeaters or gateway sites. Since repeaters and gateways can fail without causing a total network failure, special merging protocols are needed to allow replicas to be spread over more than one segment or token ring.

The read and write availabilities of replicated data managed by the Regeneration Algorithm are evaluated and two new protocols that overcome some of its limitations are presented. The first proposal combines regeneration and

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

[1] Computer and Information Sciences, University of California, Santa Cruz, CA 95064.

[2] Department of Computer Science, University of Houston, Houston, TX 77004.

and improve the availability of replicated objects. The second proposal combines regeneration and the *Dynamic Voting* approach. Unlike other regeneration protocols, it applies to environments where network failures are likely to occur as it guarantees data consistency in the presence of network partitions.

Section 2 contains an analysis of the performance of the Regeneration Algorithm. Section 3 discusses the benefits of combining regeneration and the *Available Copy* approach. Section 4 explains how mutual exclusion can be enforced by combining regeneration with *Dynamic Voting.* Section 5 discusses the operation costs of regeneration protocols and discusses strategies to reduce them. Finally, Section 6 has our conclusions.

2. AVAILABILITY ANALYSIS OF REGENERATION

A replicated object managed by the Regeneration Algorithm consists of m sites holding replicas of the data and n spare sites on which new replicas can be installed when the algorithm detects that one or more of the current replicas have become inaccessible. Spare sites and sites holding replicas may fail; we assume that failed sites are eventually repaired.

There are three cases for the Regeneration Algorithm: read, write and recovery. A read operation requires that there be at least one operational site. If there is, then the read can take place from that site. The write operation requires that there be m replicas of the object to record the write. Should this not be the case, enough replicas must be regenerated to bring the number of replicas back to its initial value before the write can complete. Suppose that k sites holding replicas have failed, then there must be at least k spares available. These spares would also need to have enough space in their secondary store to accommodate the new replicas of the object. If there are not enough spares or space then the write will fail.

The recovery operation allows recovering sites to rejoin the replication if the replica held at that site is up-to-date. If it is not, then the recovering replica is discarded. Note that a site will never recover to find that it is up-to-date and there are already m replicas. This is because regeneration is only done on write, which invalidates all failed replicas. In the event of total failure, the algorithm specifies that the resource will be re-loaded manually. We have chosen to wait for the last replica to fail to recover since this removes unpredictable human behavior from the analysis. We must wait for $n+1$ sites to recover since there are always m up-to-date sites, and in the worst case n spares could recover before any of the failed replicas have recovered.

We assume that the replicas of the replicated data object reside on distinct *sites* of a computer network. When a site fails, a repair process is immediately initiated. Should several sites fail, the repair process will be performed in parallel on these failed sites. Since the replication algorithm does not operate correctly in the presence of partitions, we assume that the communications network linking the sites cannot fail.

We assume that individual site failures, individual site repairs and writes are independent events distributed according to exponential laws. We will denote by λ the individual site failure rate, by μ the individual site repair rate, and by π the write rate. Since regeneration is only replicated object.

It has long been established that the exponential model is robust, and although the results obtained are approximate, that it provides a good estimate of system behavior. Our assumption that failures are independent events is justified by our experience with our local network. Since the systems are loosely coupled, the failure of a single site does not affect the operation of the other sites.

For the sake of simplicity, we will assume that spare sites always have enough free space in secondary store to accommodate new replicas and that no regeneration will ever fail because of space constraints. This is not a restriction on the protocols, but a simplifying assumption for the sake of the analysis.

The availability A^o of a replicated data object with respect to a given operation o will be defined as the limiting value of the probability $p_o(t)$ that the operation could be performed successfully on the object at time t for t going to infinity:

$$A^o = \lim_{t\to\infty} p_o(t)$$

The behavior of a replicated object with m replicas and n spares managed by the Regeneration Algorithm can be represented by the Markov chain shown in figure 1. There are five classes of state transitions: the failure of replicas, the failure of spares, the repair of replicas, the repair of spares and regeneration. The failure of a replica is indicated by a transition from left to right, from state $\langle i, j\rangle$ to $\langle i-1, j\rangle$, $i>0$, with rate $i\lambda$. The failure of a spare is similarly modeled by a transition from top to bottom, from a state $\langle i, j\rangle$ to $\langle i, j-1\rangle$, $j>0$, with rate $j\lambda$.

The repair of a replica is indicated by a transition from right to left, from state $\langle i-1, j\rangle$ to $\langle i, j\rangle$, with rate $(m-i+1)\mu$. Such a transition is possible since we know that the replica must be up-to-date, since a write would have caused a regeneration to occur and put the system in state $\langle m, k\rangle$. Failures can occur following a write, in which case those failed replicas become out-of-date and are transformed into spares. The repair of a spare is indicated by a transition from bottom to top, from state $\langle i, j-1\rangle$ to $\langle i, j\rangle$, with rate $(n-j+1)\mu$.

The algorithm specifies that a regeneration take place when there are only $m-k$ replicas available, $k>0$, and there are at least k spares. A regeneration is modeled by a transition from state $\langle m-k, j\rangle$ to $\langle m, j-k\rangle$, $j\geq k$, with rate π. When a write operation occurs with rate π, and there are fewer than m replicas, a regeneration will take place.

Let p_{ij} denote the probability that the object is in state $\langle i, j\rangle$. The *read availability* of a replicated object with m replicas and n spare sites will be given by

$$A^r_{RA}(m, n) = \sum_{i=1}^{m}\sum_{j=0}^{n} p_{ij}$$

and the *write availability* is given by

$$A^w_{RA}(m, n) = \begin{cases} \sum_{i=m-n}^{m}\sum_{j=m-i}^{n} p_{ij} & \text{if } m > n \\ \sum_{i=1}^{m}\sum_{j=m-i}^{n} p_{ij} & \text{if } m \leq n \end{cases}$$

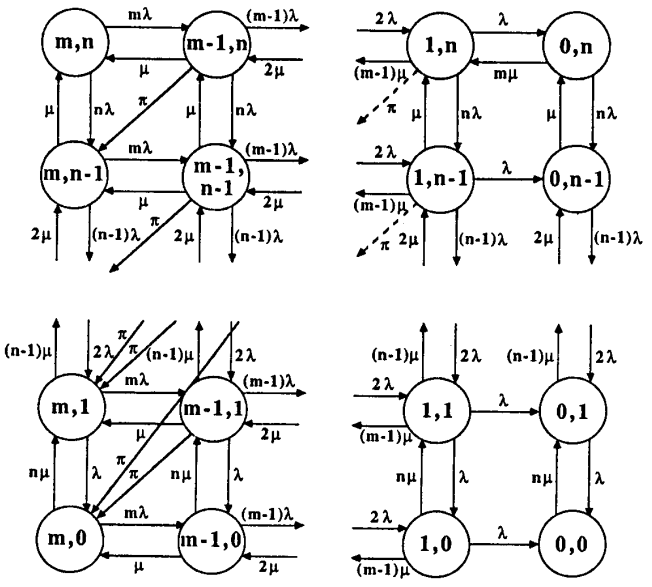

Figure 1: State Transition Diagram for Regeneration

As the system has exactly $(m+1)(n+1)$ distinct states, these expressions can be represented as quotients of polynomials of maximum degree $(m+1)(n+1)$ in λ, μ, and π. These quotients quickly become too complex to be manipulated other than by symbolic manipulation programs. For instance, we have for two replicas and a single spare:

$$A^r_{RA}(2,1)=\frac{2+10\rho+18\rho^2+12\rho^3+2\theta+7\rho\theta+\rho^2\theta}{(1+\rho)^3(2+4\rho+2\rho^2+2\theta+\rho\theta)}$$

with $\rho=\lambda/\mu$ and $\theta=\pi/\mu$. When $m>n$, any system state in which the total number of accessible replicas and spares $i+j$ is greater than or equal to m contains at least one up-to-date replica. The write availability of the data is then given by the formula for m-out-of-$(m+n)$ sites:

$$A^w_{RA}(m,n)=\frac{\sum_{k=0}^{n}\rho^k}{(1+\rho)^{m+n}}$$

Figures 2 and 3 display respectively the read and write availabilities of replicated data with two replicas and one, two or three spares for values of the failure-rate-to-repair-rate ratio ρ varying between 0 and 0.5. We assume for both graphs that the regeneration rate π is equal to ten times the site repair rate. In our experience, having π be ten times the site repair rate is a conservative estimate. For example, if site repairs take an average of an hour, regenerations would take six minutes. Twenty megabytes of data can easily be transferred over a local area network in this period. Figure 2 demonstrates the excellent read availability of replicated data managed by the Regeneration Algorithm while figure 3 displays a lower performance for write availability.

As figures 4 and 5 indicate, this difference is even more accentuated when the replicated data have three

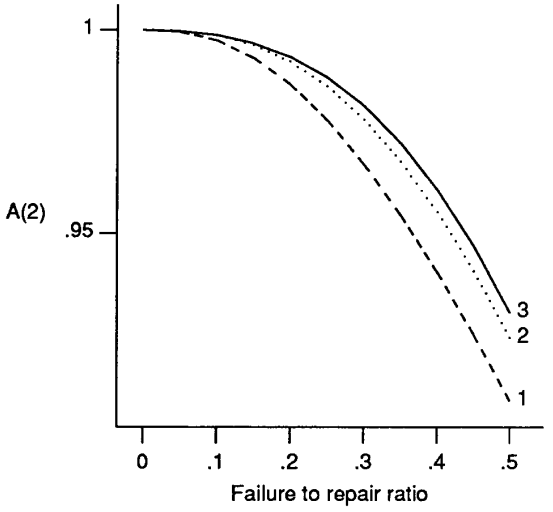

Figure 2: Read Availabilities for Two Replicas and Various Numbers of Spares

replicas. Read availabilities remain extremely high even for large values of ρ while write availabilities remain mediocre for one or even two spares, falling well below the availabilities of replicated data with three replicas and no spares managed by an *Available Copy* protocol.

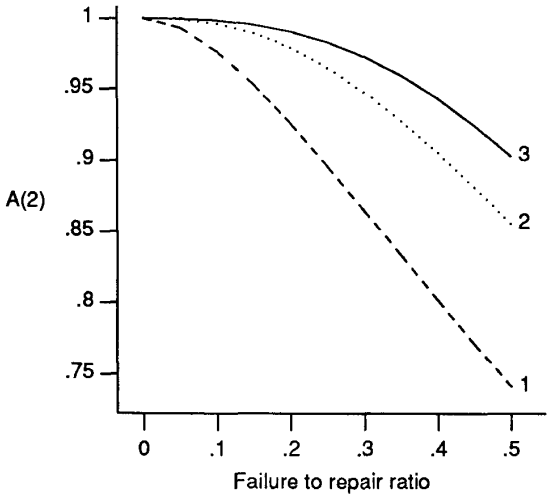

Figure 3: Write Availabilities for Two Replicas and Various Numbers of Spares

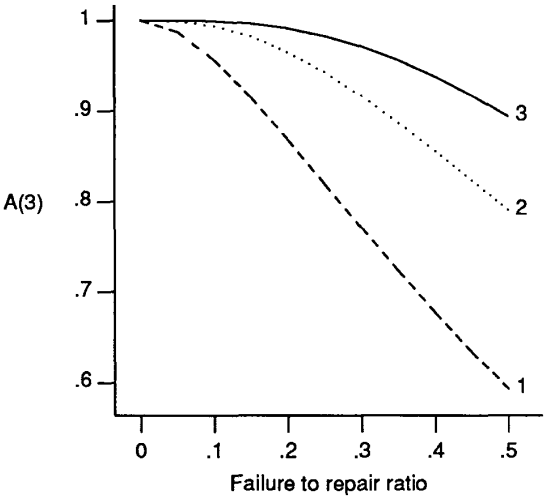

Figure 5: Write Availabilities for Three Replicas and Various Numbers of Spares

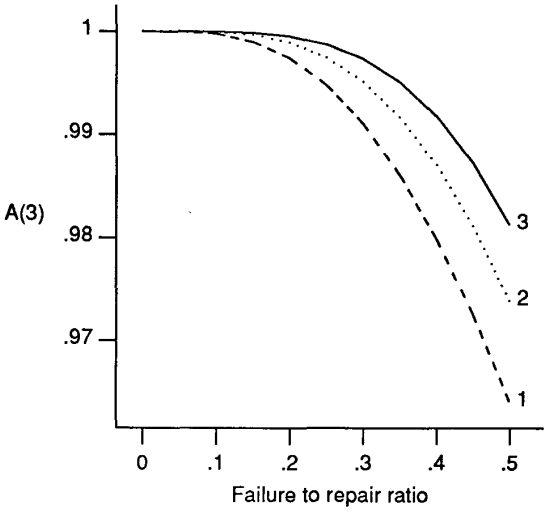

Figure 4: Read Availabilities for Three Replicas and Various Numbers of Spares

3. REGENERATIVE AVAILABLE COPY

When network partitions are known to be impossible, *Available Copy* protocols provide a simple means for maintaining data consistency. These protocols are based on the observation that replicas that have participated in all writes necessarily hold the most recent version of the data. The object remains available so long as at least one of these replicas remains accessible.

The write rule for an Available Copy protocol is extremely simple: *write to all available copies.* Since all available copies receive each write, they are kept in a consistent state: data can then be read from any available copy. If there is a replica of the data on the local site, then the read operation can be accomplished locally, avoiding any network traffic.

When a site recovers following a failure, if there is another site which holds the most recent version of the data then the recovering site can repair immediately. In the event of total failure it is not known by the recovering sites which of them hold the most recent version of the data until the last site to fail can be found. In order to speed recovery, it is desirable to ascertain as quickly as possible the last site, or set of sites, that failed.

There are several advantages in combining the regeneration and Available Copy approaches. As noted by Noe and Andreassian [10], it would considerably improve on write availability by allowing writes as long as one replica of the data remains accessible. Recoveries from total failures would also be accelerated as the protocol would keep track of replica states.

The behavior of a replicated object with m replicas and n spares managed by an Available Copies protocol with regeneration can be represented by a Markov chain organized as a series of $n+1$ layers. As shown on figure 6, each horizontal plane has $2m$ states having the same number of available spares but different numbers of available replicas.

Failures and recoveries in a layer occur exactly as they would in an Available Copy model [9]: uncomplemented states represent configurations where the replicated object has from m to zero available replicas while complemented states represent configurations where the last replica that failed remains unavailable and some of the sites holding other replicas have recovered. The failure of a replica is indicated by a transition from left to right, from state $\langle i, j \rangle$ to $\langle i-1, j \rangle$, $i>0$, with rate $i\lambda$. A failure can also occur from back to front, from state $\langle 1, j \rangle$ to $\langle 0, j \rangle$.

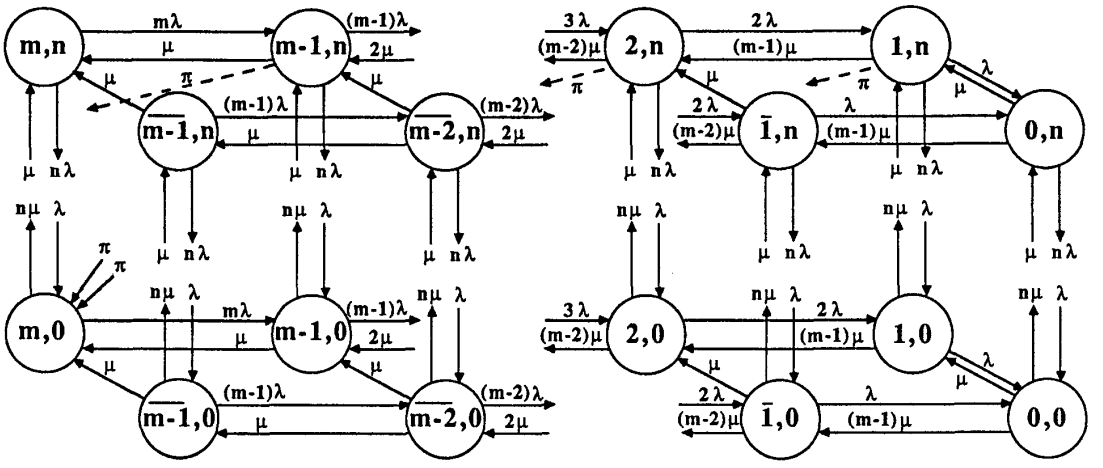

Figure 6: State-Transition Diagram for Available Copy with Regeneration

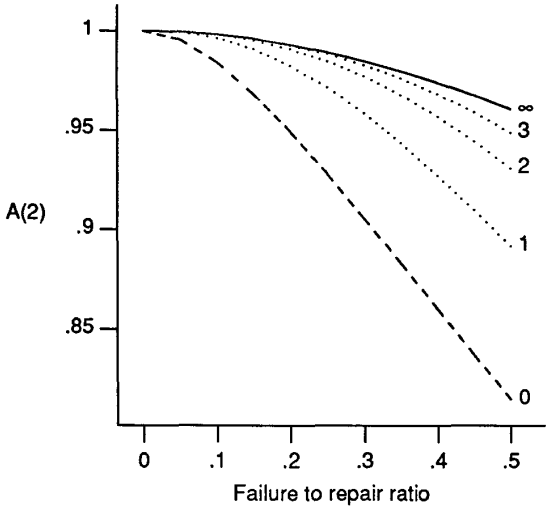

Figure 7: Availabilities for Two Available Copies with Regeneration and Various Numbers of Spares

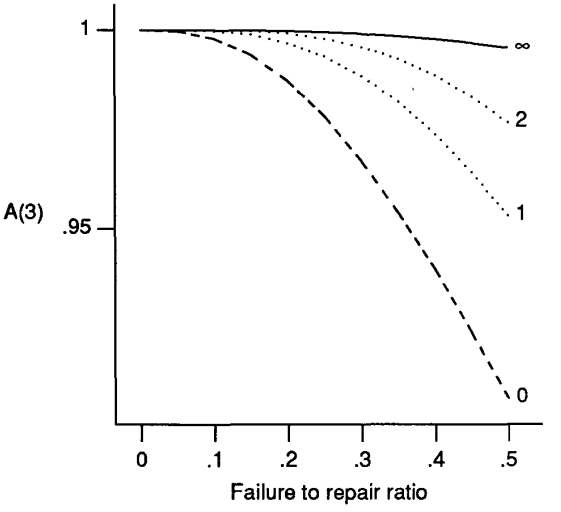

Figure 8: Availabilities for Three Available Copies with Regeneration and Various Numbers of Spares

Following such a failure the system is unavailable. The recovery of a replica is indicated by transition from right to left, from state $\langle i-1, j\rangle$ to $\langle i, j\rangle$, with rate $(m-i+1)\mu$. Recoveries can also occur from front to back, from state $\langle \overline{i-1}, j\rangle$ to $\langle i, j\rangle$, or from state $\langle 0, j\rangle$ to $\langle 1, j\rangle$, with rate μ. Such a transition indicates the recovery of the last site to fail.

The activity of spares is modeled by the transitions between layers. The failure of a spare is indicated by a transition from top to bottom, from a state $\langle i, j\rangle$ to $\langle i, j-1\rangle$, $j>0$, with rate $j\lambda$. The recovery of a spare is similarly modeled by a transition from bottom to top, from a state $\langle i, j-1\rangle$ to $\langle i, j\rangle$, with rate $(n-j+1)\mu$.

The regeneration process for Available Copy with regeneration is modeled by a transition from state $\langle m-k, j\rangle$ to $\langle m-k+l, j-l\rangle$, $l=\min(k, j)$. The Available Copy protocol attempts to regenerate as many replicas as possible, up to m, but, unlike the pure regeneration algorithm, is satisfied if it cannot regenerate all replicas.

Since the replicated data remain available as long as one replica is accessible, the read and write availabilities of the replicated object are identical and are given by

$$A_{ACR}(m, n)=\sum_{i=1}^{m}\sum_{j=0}^{n} p_{ij}$$

where p_{ij} is the probability of the replicated object being in state $\langle i, j\rangle$. As the system has exactly $2m(n+1)$ states,

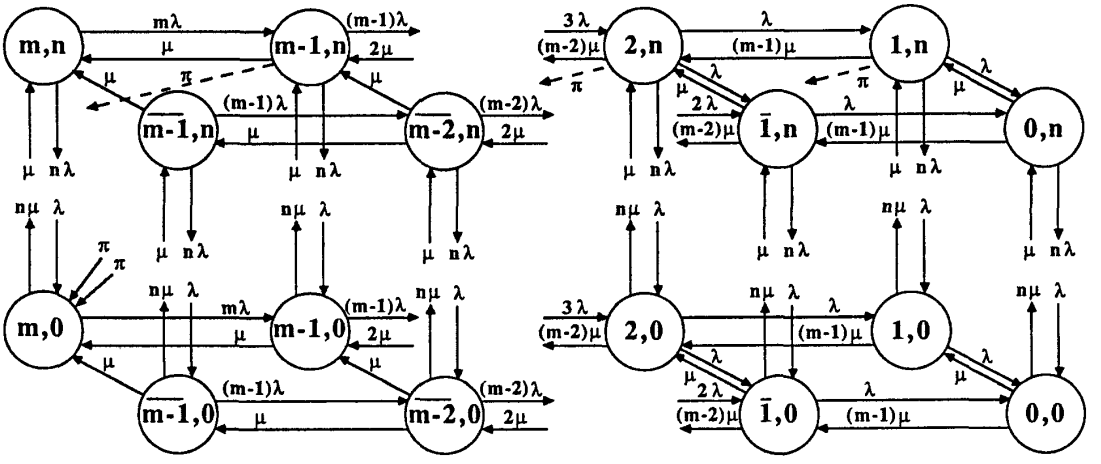

Figure 9: State-Transition Diagram for Dynamic Voting with Regeneration

this expression can be represented as a quotient of two polynomials of maximum degree $2m(n+1)$ in λ, μ and π.

Figures 7 and 8 display the availabilities of replicated data with respectively two and three available copies for various numbers of spares. We assume for both figures that the regeneration rate π is equal to ten times the site repair rate.

When no spare sites are included, we have the same availability as a regular Available Copy protocol. The addition of spare sites to the configuration has the effect of accelerating partial recoveries when replicas can be generated faster than sites can be repaired. Since the protocol keeps track of replica states, recovery from total failure only requires the recovery of the sites belonging to the last set of available replicas.

This protocol does have limitations. Since replicated data now have the same read and write availabilities, the higher write availabilities obtained with this protocol were achieved at the expense of lower read availabilities. Allowing writes when only one replica of the object remains available significantly increases the probability of irrecoverable failures, such as corruption of the only up-to-date replica. To achieve a higher level of protection in environments where this could be a problem, writes could be required to involve some minimum threshold of replicas m' with $1 < m' < m$.

4. REGENERATIVE DYNAMIC VOTING

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 susceptible to network partitions just as long-haul point-to-point networks are. Network partitions pose a special threat to replicated data since having replicas on both sides of a partition could allow the replicated data to be left in an inconsistent state. 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 consistency protocol based on quorum consensus.

Majority Consensus Voting [4,5] is the best known example of such a protocol. As it is a static protocol, it has the major disadvantage of only providing reliability and availability figures well below those provided by dynamic protocols.

Unlike Majority Consensus Voting, *Dynamic Voting* protocols [3] automatically adjust the necessary quorum of replicas required for an access operation 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 protocol checks if enough replicas remain available to satisfy the current quorums. If this is the case, these replicas constitute a new *majority block* and a new quorum is computed. To enforce mutual exclusion, recovered replicas that do not belong to the current majority block will not be 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 will maintain some state information. This information will include a *version number* identifying the last write recorded by the replica and either a *partition vector* [3] or both a *partition set* and an *operation number* [14] identifying the replicas belonging to the current majority block. These algorithms perform identically as long as the access rate is sufficient to keep the partition sets up-to-date.

The state-transition-rate diagram for Dynamic Voting with m replicas and n spare sites is shown in figure 9. We assume that network partitions are possible but have a negligible probability. This assumption keeps the comparison between the three protocols equitable and further simplifies our models. Since individual site failures and repairs are the only likely events, the diagram is very similar to the diagram we previously obtained for Available Copy. The two protocols behave similarly so long as two or more replicas remain accessible. When one of the two last replicas becomes inaccessible, the protocol has no way to ascertain if this results from an unlikely network partition or from a more likely site failure. To enforce mutual exclusion and to protect against network failures, the protocol then relies on a tie-breaking rule [7] and only

allows the data to remain available if the site holding the last accessible replica precedes the site holding the replica that became inaccessible according to some arbitrary static ordering of the sites in the network. All states $\langle 2, j \rangle$ with $j=1, \cdots, n$ have two outgoing failure transitions instead of one. The first transition, from $\langle 2,j \rangle$, with rate λ goes to state $\langle 1, j \rangle$ and corresponds to the situation where the site holding the last accessible replica precedes the site holding the replica that became inaccessible. The second transition with rate λ goes to state $\langle \tau, j \rangle$ and corresponds to the the other case.

As was the case for the protocol combining the Available Copy and the regeneration approaches, the read and write availabilities of the replicated data are identical and are given by

$$A_{DVR}(m, n) = \sum_{i=1}^{m} \sum_{j=0}^{n} p_{ij}$$

where p_{ij} is the probability of the replicated object being in state $\langle i, j \rangle$. As the system has exactly $2m(n+1)$ states, this expression can be represented as a quotient of two polynomials of maximum degree $2m(n+1)$ in λ, μ and π.

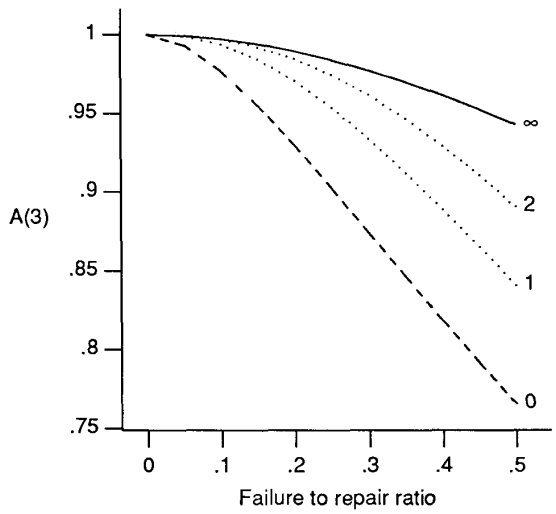

Figure 10: Availabilities for Dynamic Voting with Regeneration and Various Numbers of Spares

Regeneration only has the effect of accelerating partial recoveries and does not modify quorum size. As a result, our Dynamic Voting protocol with regeneration requires a minimum of three replicas to operate effectively. Figure 10 displays the availabilities of replicated data with three replicas and various numbers of spares between zero and infinity. As we have done before, we assume that the regeneration rate π is equal to ten times the site repair rate. In this case as well, the benefits of regeneration are clearly visible: adding even one single spare site to the three replicas has an immediate beneficial effect on the availability of the replicated data. Increasing the number of spares has a similar effect although the benefit of adding one extra spare tapers off more quickly than for our Available Copy protocol with regeneration.

5. THE COST OF REGENERATION PROTOCOLS

An analysis of regeneration-based protocols cannot be complete without a brief discussion of their operating costs. Two types of costs need to be considered here, namely, the costs of using the protocols under normal operating conditions and the costs of regenerating failed replicas.

In the absence of failures, Available Copy with Regeneration and Dynamic Voting with Regeneration operate exactly as conventional Available Copy or Dynamic Voting protocols do. As a result, they incur the same costs. Efficient implementations of Available Copy [9] and Dynamic Voting Protocols [14] have been proposed. These implementations would require very few changes to accommodate protocols with regeneration.

The perceived high costs of generating new replicas probably constitute the greatest obstacle preventing a wider use of regeneration-based protocols. It is indeed true that frequent regenerations of large files can unnecessarily slow writes while overtaxing the network bandwidth. However, the true impact of this phenomenon on the system performance must be assessed from a realistic perspective. Regenerations can only occur after one or more replicas have become inaccessible. In the absence of network failures, a replicated object will never regenerate at a higher rate than the combined failure rates of its replicas. Therefore, regenerations will remain infrequent events in most installations.

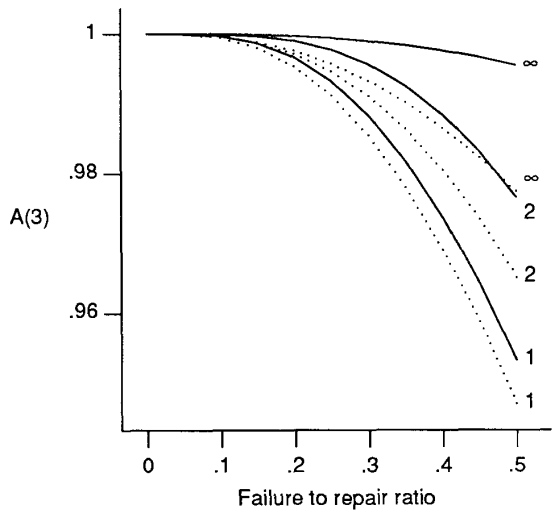

Figure 11: Availabilities for Three Available Copies with Mitigated Regeneration

However, there are a significant number of computing environments experiencing a relatively high number of site failures. It is often the case that most of these failures result from software errors. Unlike failures resulting from hardware malfunctions, software failures only require a system restart. This procedure can be performed automatically in most installations and typically requires from ten to twenty minutes [14]. A significant number of regenerations can be avoided by waiting for a fixed time interval in the same range before initiating the

regeneration of a replica that has just become unavailable. This *temporization* strategy would have very little impact on the availability of the replicated data as long as the rate at which failed replicas are regenerated continues to remain higher than their overall repair rate.

Another possible approach is to wait until the number of accessible replicas falls below some threshold $m'' < m$ before initiating any regeneration. For instance, such *mitigated regeneration* protocols would only regenerate when less than three of the five original replicas of an object remain available. Figure 11 compares the availability of mitigated and unmitigated Available Copy protocols with regeneration with three replicas. Dashed curves reflect the availabilities obtained using unmitigated regeneration while continuous curves reflect the availabilities obtained when one replica is regenerated every time a single replica remains available. As one can see, the mitigated protocol performs nearly as well as the protocol attempting to maintain three available replicas under all circumstances.

A third method for limiting the cost of regeneration applies only to Dynamic Voting protocols with regeneration. It consists of regenerating *witnesses* instead of full replicas. Witnesses are inexpensive to regenerate as they contain only state information [12]. A similar idea has recently been proposed by van Rennesse and Tanenbaum in the context of voting protocols that take into account the topology of the network on which the replicas reside [19].

6. CONCLUSIONS

Regeneration attempts to increase the reliability and availability of replicated data by generating new replicas when one or more of the replicas are missing. We have presented an availability analysis of the regeneration-based consistency control protocol that has been proposed by Pu [15-17]. We have also presented two regeneration protocols overcoming some of the limitations of that scheme. Our first protocol combines regeneration and the *Available Copies* approach to improve on the availability of replicated data. Like the original *Regeneration Algorithm,* it applies to environments where network partitions are impossible. Our second protocol combines regeneration and the *Dynamic Voting* approach to guarantee data consistency in the presence of network partitions while maintaining high availability. The availabilities of replicated data managed by both protocols were derived and found to improve significantly on the write availabilities provided by extant consistency protocols. Several techniques to reduce the costs of frequently regenerating large data objects were also discussed.

We found the Available Copy protocol with regeneration to perform better than Pu's Regeneration Algorithm and all Available Copy protocols not including regeneration. We hope that architects of systems implementing replicated objects will consider Available Copy with regeneration as the consistency protocol of choice for environments where network partitions are known to be impossible. Dynamic Voting with regeneration extends the benefits of regeneration to environments where network partitions are possible; it should be a prime contender in such environments.

Further work is still needed to assess the network traffic overhead resulting from regeneration, to estimate its additional storage costs, and to evaluate the performance of mitigated protocols that would only regenerate a fraction of the initial number of replicas. The applicability of regeneration to other consistency protocols should also be investigated; block-level protocols and protocols using witnesses are prime candidates for regeneration since the objects managed are small.

Acknowledgements

We wish to thank Walter A. Burkhard, Charles Bergan, Alexander Glockner and all the other members of the Gemini group for their help and their encouragement. This work has been done with the aid of MACSYMA, a large symbolic manipulation program developed at the Massachusetts Institute of Technology Laboratory for Computer Science. MACSYMA is a trademark of Symbolics, Inc.

References

[1] D. Barbara, H. Garcia-Molina and A. Spauster. "Policies for Dynamic Vote Reassignment." *Proc. 6th ICDCS,* (1986), pp. 37-44.

[2] P.A. Bernstein and N. Goodman. "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases." *ACM TODS,* 9 (1984), 596-615.

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

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

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

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

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

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

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

[10] J.D. Noe and A. Andreassian. "Effectiveness of Replication in Distributed Computing Networks." *Proc. 7th ICDCS,* (1987), pp. 508-513.

[11] J.D. Noe, A.B. Proudfoot and C. Pu. "Replication in Distributed Systems, The Eden Experience." *Proc. 1986 FJCC,* (1986), pp. 1197-1209.

[12] J.-F. Pâris. "Voting with Witnesses, A Consistency Scheme for Replicated Files." *Proc. 6th ICDCS,* (1986), pp. 606-612.

[13] J.-F. Pâris. "Voting with a Variable Number of Copies." *Proc. 16th FTCS,* (1986), pp. 50-55.

[14] J.-F. Paris and D.D.E. Long. "Efficient Dynamic Voting Algorithms." *Proc. 4th Int. Conf. on Data Engineering,* (1988), pp. 268-275.

[15] C. Pu. "Replication and Nested Transactions in the Eden Distributed System." Ph.D. dissertation, Computer Science Department, University of Washington (1986).

[16] C. Pu, J.D. Noe and A. Proudfoot. "Regeneration of Replicated Objects, A Technique and its Eden Implementation." *Proc. 2nd Int. Conf. on Data Engineering,* (1986), pp. 175-187.

[17] C. Pu, J.D. Noe and A.B. Proudfoot. "Regeneration of Replicated Objects, A Technique and its Eden Implementation." *IEEE TSE,* SE-14, 7 (July 1988), 936-945.

[18] R.H. Thomas. "A Majority Consensus Approach to Concurrency Control." *ACM TODS* 4, (1979), 180-209.

[19] R. van Rennesse and A. S. Tanenbaum. "Voting with Ghosts." *Proc. 8th ICDCS,* (1988), pp. 456-461