An Available Copy Protocol Tolerating Network Partitions

Jehan-François Pâris

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

ABSTRACT

Maintaining in a consistent state multiple copies of the same data is a complex task, especially when the copies reside on sites that can be separated from each other by network partitions. All existing replication controls that tolerate network partitions use *quorums* to provide mutual exclusion and prevent inconsistent updates. Unfortunately these protocols require a minimum of $n + 2$ voting sites to guarantee that the data will remain accessible in the presence of n site failures. As a result, they provide much lower data availabilities than protocols that exclude communication failures.

We present here a replication protocol that extends the *available copy* approach to environments where communication failures may cause network partitions. Our protocol assumes that replicas monitor the communication paths linking them with their peers and can therefore detect network partitions. As a result, each individual replica can safely establish whether it has remained up to date or is likely to have missed some updates.

We evaluate under standard Markovian assumptions the availability of a replicated file consisting of two replicas managed by our *robust available copy protocol* when the two replicas are separated by one gateway. We find our protocol to perform much better than (1) dynamic-linear voting with three replicas, (2) dynamic-linear voting with two replicas and one witness, (3) voting with ghosts with two replicas, and (4) voting with bystanders with two replicas.

Keywords: distributed consensus, replicated data, replication control, network partitions.

1. Introduction

Many distributed systems maintain multiple copies—or *replicas*— of some critical data at different sites within the system. This approach has the major advantage of reducing read access times and allowing continuous access to the data even if some copies become unreachable owing to site failures or network partitions. Allowing write access in the presence of equipment malfunctions introduces however another problem because these malfunctions are likely to prevent some replicas to receive all updates. Unless the proper precautions are taken, the replicated data would end in an inconsistent state where no replica would have received all updates. Special *replication control protocols* have been devised to prevent this occurrence. These protocols achieve a distributed consensus among all operational replicas and provide them with a single consistent view of the replicated data.

Voting protocols probably constitute the best known class of replication control protocols [6, 7]. They offer two major advantages over other replication control protocols: First they guarantee the consistency of the replicated data in the presence of site failures and network partitions while other protocols can only handle site failures. Second their correctness is easy to verify since they rely on intersecting quorums to prevent disjoint writes and to guarantee that every read will necessarily access at least one of the most recently updated replicas.

This conceptual simplicity comes however with a hefty cost: quorum requirements do not allow full access to replicated data unless a majority of the replicas can be reached. As a result, $2n + 1$ replicas are needed to allow full access in the presence of n simultaneous replica failures.

Numerous solutions have been proposed to overcome this limitation. *Dynamic voting* (DV) [5] and *dynamic-linear voting* (DLV) [10] adjust quorums to reflect changes in replica availability and network topology. A majority of replicas from a previous majority can thus become the new current majority. A third protocol proposed by Barbara et al. adjusts replica weights instead of excluding replicas [4]. These three protocols greatly improve the availability and reliability of replicated data with more than two replicas. The *generalized quorum consensus* protocol (GQC) [9] exploits available type specific properties of data files to allow more flexible write quorum assignments. *Voting with witnesses* (VWW) [12], *voting with ghosts* (VWG) [17] and *voting with bystanders* (VWB) [13] share the common thread of introducing auxiliary entities that are used by the protocol to improve the availability of the replicated data.

While these approaches significantly ease quorum requirements, they still provide lower availabilities than replication protocols that do not rely on intersecting quorums to enforce consistency. The *available copy* (AC) protocol [2] is probably the best known of these quorum-free protocols. Unlike voting protocols, the AC protocol needs only *two* replicas to protect against a single site failure. It also allows data to be read from *any* live replica. This excellent performance comes however at a price: the AC protocol does not guarantee data consistency in the presence of partial communication failures as it cannot prevent conflicting updates on both sides of a network partition.

While several mechanisms for minimizing the inconvenience caused by conflicting updates have been proposed and implemented [18], none of these mechanisms can guarantee a satisfactory resolution to all conflicts. We propose instead to prevent inconsistent updates by making replicas responsible for the integrity of their communication paths. Inconsistent updates will be avoided because the sites of at least one side of the partition will notice they cannot communicate with the replicas on the other side and disqualify themselves from processing further access requests.

The remainder of this paper is organized as follows: Section 2 describes in detail the AC protocol and introduces our pessimistic available copy protocol. Section 3 presents a brief analysis of replicated file availability under our new protocol and section 4 has our conclusions.

2. The Pessimistic Available Copy Protocol

The AC protocol is based on the observation that a replica will remain up to date as long as it is notified of all writes. Broadcasting all writes to all live replicas would therefore ensure that reads can be satisfied by any live replica. This approach has three important consequences: First, a replicated file managed by the AC protocol remains available as long as one live replica remains accessible. Second, replicas that reside on sites that have failed need to be marked non-available or *comatose* until they are brought up to date. Finally, the protocol does not guarantee data consistency in the presence of communication failures as it then becomes impossible to guarantee that all live replicas receive all write requests.

Recovering from a total failure requires finding the replica that failed last and upgrading its state from comatose to live. The original AC protocol [2] assumed instantaneous detection of failures and instantaneous propagation of this information. Since then, protocols that do not rely on these assumptions have been devised. One of them, the *naive available copy* (NAC) protocol, does not maintain any state information and waits until all replicas of the replicated file have recovered to ascertain which replica failed last. Another variant, the *optimistic available copy* (OAC) protocol, only maintains state information at write and recovery times. These protocols have been found to perform nearly as well as the original AC protocol, which was found to perform much better than all quorum based protocols [11].

2.1. General Principle

Since the AC protocol does not guarantee data consistency in the presence of network partitions, its usefulness is limited either to applications where conflicting updates are not a problem or to environments where network partitions are not likely to occur. This is the case if all replicas are on the same Ethernet or on the same token ring as they are both immune to the kind of partial failures that can create network partitions.

Many local-area networks consist of several Ethernets or token rings linked by selective repeaters or gateway hosts. The key difference with conventional point-to-point networks is that sites that are on the same Ethernet or token ring will never be separated by a partition. We will call these unpartitionable entities *network segments* after van Renesse and Tanenbaum [17].

Consider now the replicated file X represented on Figure 1. It consists of two replicas, R_1 and R_2, located on sites A and B respectively. Observe that the two sites communicate with each other through a third site G. They are therefore on two distinct network segments. Any failure of G would result in a partition of X into two non-communicating subsets of sites, namely $\{A\}$ and $\{B\}$. Consider now the entity B' that includes both gateway G and site B. We can envision a modified AC protocol that would treat B' as the site holding the replica located on B. Such protocol would view a failure of the gateway G as a partial failure of the site B' and mark the replica comatose. This would eliminate any risk of inconsistent updates as comatose replicas remain unavailable until they have fully recovered.

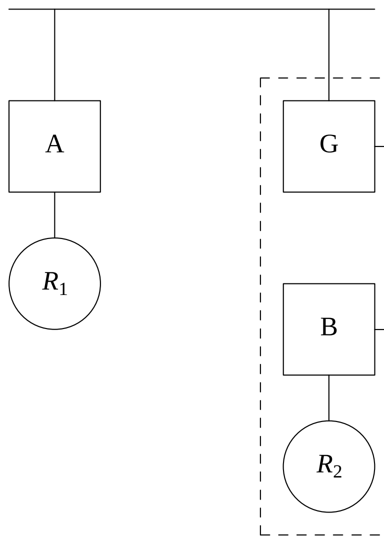

Figure 1: An file with two replicas on two network segments

A similar approach could be used with the replicated file represented on Figure 2. It consists of three replicas R_1, R_2 and R_3 located on three distinct network segments linked by the gateways G and H. Failures of either gateway G or gateway H would result in a network partition. We can here define two aggregate sites B' and C' such that B' includes site B and gateway G while C' includes site C and gateway H. A modified AC protocol that would run on site A and the two aggregate sites B' and C' would protect the replicated file against any inconsistencies resulting from a network partition as it would always mark comatose all replicas located on one side of the failing gateway.

In the two previous examples, we have extended the AC protocol by having the protocol (a) monitoring potential sources of network partition and (b) reacting to a partition by marking comatose all replicas on one side of the partition. We will therefore associate with each site S a *communication domain* C_S containing all the hardware and software resources that S needs to reach the communication domains of the other sites. For instance, the communication domains of B and C on figure 1 are respectively G and H while the communication of A is empty as A needs only G to communicate with B and H to communicate with C. Communication domains can overlap as it would have been the case if we had selected $C_A = \{G\}$, $C_B = \varnothing$ and $C_C = \{G, H\}$. Our *robust available copy protocol* (RAC) will operate exactly as conventional AC protocols with the only difference that replicas that reside on sites whose communication domains have failed need to be marked comatose and will remain in that state until they recover.

One simple optimization comes to mind. Consider again the replicated file X on figure 1 and assume that site A has failed while B and G are operational. The replica on site B is now the only live replica of X. The gateway G can be safely excluded from the communication domain of B and the replica located on that site does not need to be marked comatose if G fails before A recovers.

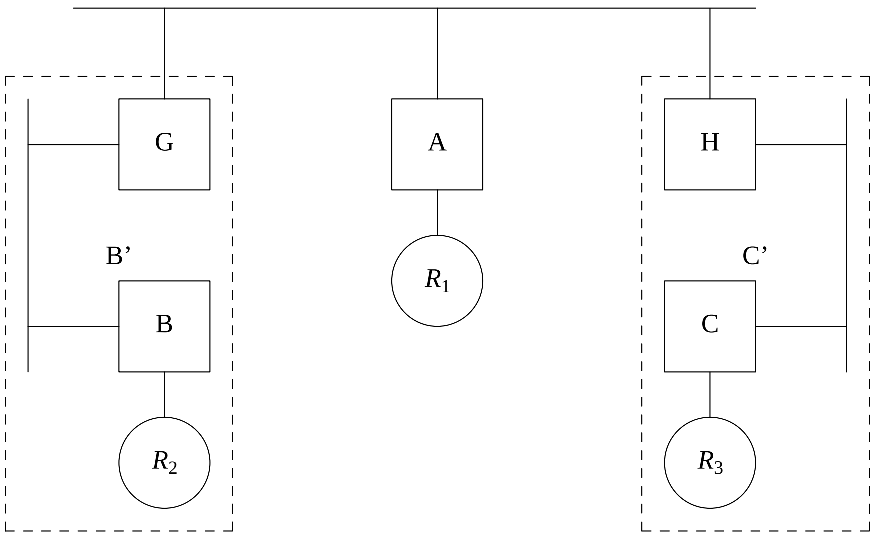

Figure 2: An file with three replicas on three network segments

We propose therefore to allow adjustments in the membership of communication domains to reflect changes in replica availability. The communication domain of a site *S* will therefore be redefined as all the hardware and software resources that *S* needs to reach the communication domains of the other sites currently holding *live replicas* of the replicated file.

2.2. Formal Definition

We consider replicated data files consisting of a set of replicas residing on distinct sites of a local area network. These sites can fail and can be prevented from exchanging messages owing to failures in the communication subnet. We will assume that sites not operating correctly will immediately stop operations and that all messages delivered to their destinations will be delivered without alterations in the order they were sent. Byzantine failures are expressly excluded.

We will focus our attention on replicated files that contain uninterpreted values. Two primitive operations on these files will be defined: a *read* operation that returns the current value of the file and a *write* operation that modifies that value in an arbitrary fashion. To guarantee single-copy serializability, we will disallow concurrent write operations and strictly enforce a single writer policy.

To simplify comparisons between replicas, we will associate to each replica a *timestamp* or *version number* that is increased on each successful write.

Definition 2.1. *The* communication domain C_S *of a site S consists of all the hardware and software resources that S needs to reach the communication domains of the other sites currently holding live replicas* of the replicated file.

Communication domains can overlap and can be empty. They can be adjusted to reflect changes in replica availability. The protocol will not guarantee data consistency if the communication domains of any pair of live replicas do not contain all the resources needed by the two replicas to communicate with each other.

Definition 2.2. *A replica is said to be* available *or* live *if neither the site on which the replica resides nor the communication domain of that site have experienced a failure since the last time the replica was written to or repaired.*

A live replica is necessarily up to date as it cannot have missed any write. Live replicas are the only ones that can perform read and write requests.

Definition 2.3. *A replica is said to be* dead *if it resides on a site that is not operational.*

Definition 2.4. *A replica is said to be* unavailable *or* comatose *if either*

(a) *the site holding the replica has recovered from a failure but the replica has not yet been repaired, or*

(b) *the communication domain of the site holding the replica has experienced any failure since the last time the replica was written to or repaired.*

Comatose replicas are prevented from performing any reads or writes. The correctness of the protocol requires that replicas experiencing a failure in their communication failure notice that failure before processing any read or write.

Write Protocol 2.1. *A write must*

(a) *be propagated to the communication domains of all live replicas, and*

(b) *be acknowledged by at least one of them.*

Read Protocol 2.2. *A read can be performed on any live replica.*

Recovery Protocol 2.3. *To be repaired, a comatose replica must compare its version number with that of*

(a) *any live replica, or*

(b) *one of the replicas that failed last, if no live replica currently exists.*

Should the version numbers differ, the content of the replica is to be copied from that of any up to date replica.

Note that these read, write and recovery protocols are identical to the ones used by the AC protocol.

The original AC protocol assumed that site failures are easily detected and notification of their occurrence can be easily broadcast to all surviving sites [8]. In general, failures are hard to detect in a reliable manner. Time-outs are the most common method to achieve this goal, but they can delay processing and are unreliable with heavily loaded sites. Our RAC protocol does not require instantaneous detection of failures. It follows the same approach as the *optimistic available copy* protocol [11] and maintains site information only at write and recovery time. Although consistency is never compromised, recovery time increases as the state information ages and becomes out-of-date.

We assume that each gateway has a local clock that can generate monotonically non-decreasing timestamps. These local clocks need not be synchronous. We require each gateway to maintain a *boot timestamp* b that is to be reset to the current value of the gateway clock every time the gateway recovers from a failure.

Each site holding a replica r will maintain four data structures. These are:

(1) a *version number* v_r that is incremented after every successful write operation,

(2) a *was-available set* W_r that enumerates the replicas that were alive when the set was updated last,

(3) a *gateway vector* $\mathbf{G}^r$ whose i-th entry G_i^r contains the set of gateways used by r to communicate with the communication domain of replica i, and

(4) a *timestamp vector* $\mathbf{z}^r$ containing the last collected values of the local times of the gateways in the communication domain of the replica.

Was-available sets can be maintained inexpensively by ascertaining which replicas are operational when the replicated file is first accessed and by sending this information along with the first write; the second write will contain the set of replicas which received the first write and so forth. By delaying the relaying of information this way, we minimize communication costs. However, it makes it necessary to compute the closure of the was-available set with respect to a recovering site r in order to find the last site to fail. The *closure* of a was-available set W^r, written $C^*(W^r)$, is given by:

$$C^*(W^r) = \bigcup_{i=0}^{n} C^k(W^r)$$

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

A major difference with conventional available copy protocols is that replicas receiving read or write requests must first find if they are still alive or have become comatose after the failure of a gateway in their communication domain. As seen in Figure 3, the algorithm starts by establishing the current communication domain of the replica r. This communication domain D is obtained by taking the union of all entries in the gateway table that correspond to replicas in transitive closure of its was-available set. The protocol then requests from every site in D its boot timestamp b_s and its local time t_s. S is the set of sites that replied and X the set of sites that recovered from a failure since the last time r recorded their boot timestamp in z^r. The replica is assumed to be alive if every site s in D returns a boot timestamp b_s lesser than or

```
function SELF_CHECK(r : replica)
begin
    let $W_r$ be the was-available set of $r$
    let $\mathbf{G}^r$ be the gateway vector of $r$
    let $\mathbf{z}^r$ be the timestamp vector of $r$
    $V = C^*(W_r)$
    $D = \bigcup_{s \in V} G_s^r$
    $<S, \mathbf{b}, \mathbf{t}> \leftarrow$ START($D$)
    $X = \{s \in S | b_s > z_s^r\}$
    if $S = D$ and $X = \varnothing$ then
        for all $s$ in $D$ do
            $z_s^r = t_s$
        od
        return(LIVE)
    else
        start RECOVER($s$)
        return(COMATOSE)
    fi
end SELF_CHECK
```

Figure 3 Self-Check Algorithm

equal to the value recorded in z_s^r. The timestamp vector of r is then updated by replacing each z_s^r by the corresponding t_s. The replica is assumed to be comatose if one or more sites in D did not answer the request or returned a boot timestamp b_s greater than the value recorded in z_s^r.

Note that the algorithm could be somewhat simplified if all sites were to maintain synchronous clocks as there would be no need to collect local clock values from all gateways nor to maintain a vector of timestamps for each replica. A protocol assuming synchronous clocks would maintain a single *gateway timestamp* and set it to the network local time every time the self-check procedure is executed successfully.

The recovery algorithm is run each time a site holding a replica recovers from a failure or detects its replica is comatose while running the self check algorithm. The algorithm starts by marking the replica COMATOSE and waits until either

(1) one LIVE replica s can be found, or

(2) all replicas in the closure of its was-available set W_r have recovered. The current version of the file is then obtained by computing the highest version number of all replicas in $C^*(W_r)$ and selecting one replica s with $v_s = v_{\max}$.

The algorithm then updates the replica and its version number v_r. The new was-available set of r is computed by including r in the was-available set of s and notifying s that r should be included in its was-available set. The protocol then completes the recovery of r by requesting the current local times of all gateway sites in $\mathbf{G}$ and using these times to update the timestamp vector of r.

Note that a recovery protocol assuming synchronous clocks would only need to set the *gateway timestamp* of r to the current network time.

```
procedure RECOVER($r$ : replica)
begin
    let $W_r$ be the was-available set of $r$
    let $U$ be the set of all replicas
    let $\mathbf{G}^r$ be the gateway vector of $r$
    let $\mathbf{z}^r$ be the timestamp vector of $r$
    state($r$) $\leftarrow$ COMATOSE
    select
        when $\exists t \in U$ with $state(t)$ = LIVE =>
            skip
    or
        when all replicas in $C^*(W_r)$ have recovered =>
            $v_{\max} = \max_{r \in C^*(W_r)} \{v_r\}$
            select $t \in C^*(W_r)$ with $v_t = v_{\max}$
    end select
    repair $r$ from $t$
    $v_r \leftarrow v_t$
    $W_r \leftarrow W_t \cup \{r\}$
    $W_t \leftarrow W_r$
    state($r$) $\leftarrow$ LIVE
    $D = D \bigcup_{s \in U} G_s^r$
    $\mathbf{t} \leftarrow$ START($D$)
    for all $s$ in $D$ do
        $z_s^r = t_s$
    od
end RECOVER
```

Figure 4: Recovery Algorithm

2.3. Discussion

To prove the correctness of the RAC protocol, we need to prove two propositions:

(1) the closure of the was-available set of any replica always contains the name of a current replica of the file, and

(2) a replica that has become comatose owing to the failure of a site in its communication domain will always detect this change of state before answering any access request.

Proving the first proposition can be done by observing first that the condition holds in the initial state where all replicas are alive and $W_r = U$ for all $r \in U$. It suffices then to show that the three types of events that can change the state of the replicated file, namely, write operations, site failures and replica recoveries, preserve the condition and guarantee that $C^*(W_r)$ contains the name of a current replica of the file for all $R \in U$. A complete description of the proof can be found in [11].

Proving the second proposition requires showing that the self-check algorithm will always detect all gateway failures that might have affected the state of a replica. The self-check algorithm makes two essential assumptions about these failures. First, it assumes that all gateway failures will be clean failures and that a failing gateway will immediately halt. Hence, Byzantine failures are excluded. Second, it assumes that a gateway recovering from a failure will immediately update its boot timestamp. This second assumption specifically excludes transient failures unless the gateway has some means to record their occurrence.

The exclusion of all types of Byzantine failures is a common feature of all existing replication control protocols. This exclusion is necessary as protocols tolerating Byzantine failures would be much slower and have a much higher message traffic overhead. In this sense, the RAC protocol is not less robust than any other voting or non-voting replication control protocol.

Most system failures are good approximations of a clean failure because failures are detected before serious damage is done. There are however at least two types of failures that need to be handled separately as they do not follow this paradigm. These are corrupted network tables and buffer overflow errors.

Gateways can have their routing tables corrupted by software errors. As a result, the gateway may stop forwarding packets for one or more sites while continuing to handle correctly packets directed to other destinations. To detect this type of failure, the gateway must have some procedure for checking the integrity of its routing tables, such as recomputing their checksum, and execute this procedure before answering any request for its current boot time.

Buffer overflow errors occur when a site is overloaded and packets arrive in its hardware buffer at a faster rate than it can fetch and process them. One solution is to have a process continuously monitoring packet arrivals and detecting overflows.

3. Stochastic Analysis

In this section we present an analysis of the availability provided by our protocol and compare it with the availabilities afforded by the best protocols that tolerate network partitions. We will assume here that the availability of a replicated file is the stationary probability of the file being in any state permitting access. $A_S(n)$ will denote the availability of a file with n replicas managed by the protocol S.

Our model consists of a set of sites with independent failure modes that are connected via a network composed of network segments linked by gateways. 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 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.

These assumptions are required for a steady-state analysis to be tractable and have been made in most probabilistic studies of the availability of replicated data [1, 3, 10-15]. Combinational models that do not require any assumptions about failure and repair distributions have been proposed [16, 17] but these models cannot distinguish between live and comatose replicas.

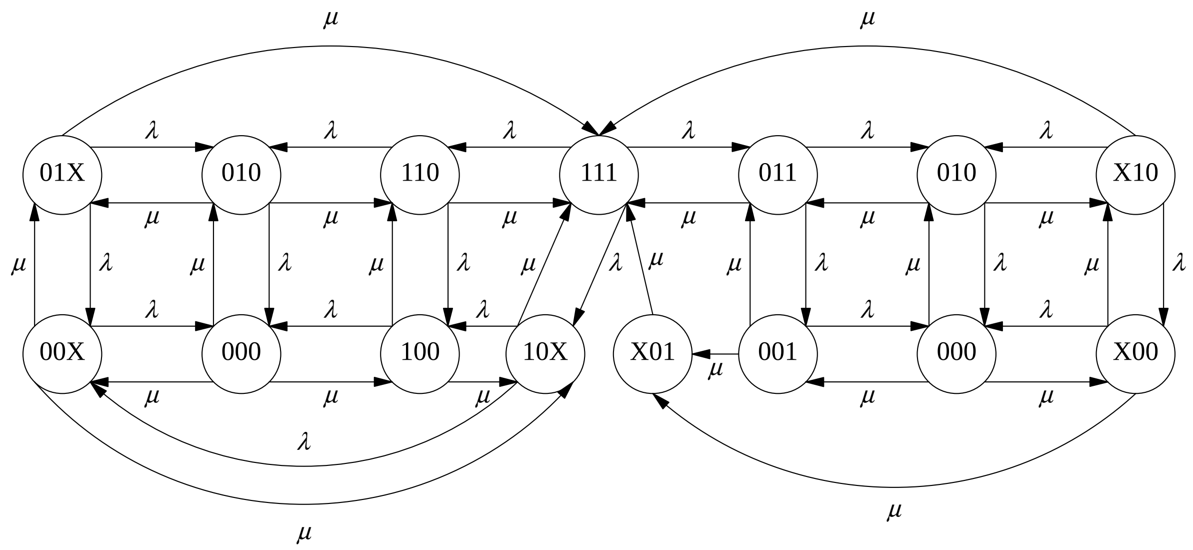

Figure 5: State transition rate diagram for two replicas separated by a gateway and managed by the RAC protocol

Estimating the availability of a replicated file becomes much more difficult when network partitions are considered because the number of potential states of a replicated file subject to network partitions is much larger than when network partitions are not considered. We will focus our analysis on the case of a replicated file consisting of *two* replicas separated by a *single gateway.* Although larger configurations can be analyzed using the same approach, we did not include them here owing to space considerations.

Figure 5 shows the state transition rate diagram for a replicated file consisting of two full replicas separated by a single gateway managed by the RAC protocol. Each of its fifteen states is represented by a triple $<uvw>$ where u and w are the states of the two replicas (1 if live, 0 if dead and X if comatose) and v is the state of the gateway G (1 if operational, 0 otherwise). States where the replicated file is unavailable are identified by one or two prime marks (′ ″): A single prime indicates that replica A is the sole live replica while two primes indicates that B is the sole live replica.

A replicated file with its two replicas and its gateway fully operational is in state $<111>$. A failure of the gateway puts replica B in a comatose state and brings the file into state $<101_A>$ until the gateway gets repaired. If replica A fails before the gateway gets repaired, the file enters state $<001_A{}'>$ and from there to $<000_A{}'>$ if replica B fails too. A recovery of replica A would bring the replicated file either from state $<001_A{}'>$ to state $<101_A>$ or from state $<000_B{}'>$ to state $<100>$.

A replicated file in state $<111>$ that loses replica B enters state $<110_A>$. The file can either recover and return to state $<111>$ or fail again and enter either state $<100_A>$ or state $<010_A{}'>$. A recovery from $<100_A>$ can bring the file either to $<110_A>$ or to $<101_A>$.

Finally, a replicated file in state $<111>$ that loses replica A enters state $<011_B>$. Since replica B is now the only live replica, it can exclude the gateway G from its communication domain. The replicated file will therefore remain available as long as replica B remains available. Transitions between states remain otherwise similar to those observed between states where A is the only current replica.

The availability of the replicated file is given by the sum of the probabilities of the system being in a state permitting access:

$$A_{RAC}(1+1) = p_{111} + p_{110_A} + p_{101_A} + p_{100_A} + p_{011_B} + p_{101_B} + p_{001_B}$$

where p_{uvw} is the probability that the replicated file is in state $<uvw>$.

The graph in Figure 6 presents a comparison of the availability of two replicas separated by a gateway and managed by RAC with those afforded by four other replica configurations. This comparison was made for values of the failure rate to repair rate ratio $\rho = \lambda/\mu$ varying between 0 and 0.20. The first value corresponds to perfectly reliable sites and the latter to sites that are repaired five times faster than they fail and have an individual availability of 0.833.

The first benchmark configuration consisted of three replicas on two network segments separated by a gateway and managed by the dynamic-linear voting protocol [10], which is the best extant voting protocol. We had to include an additional replica since voting protocols require a minimum of three voting entities to improve upon the availability of an unreplicated file. The availability of this configuration is represented by the curve labeled D(3). Its derivation can be found in [14].

The second configuration consisted of two replicas on two network segments and one witness on the gateway linking the two segments managed by the DLV protocol. The availability of this configuration is represented by the curve labeled D(2+W). We selected this configuration to test whether locating the witness on the gateway could not have the same beneficial effect under the DLV protocol as under the RAC protocol.

The third benchmark configuration consisted of two replicas on two network segments separated by a gateway and

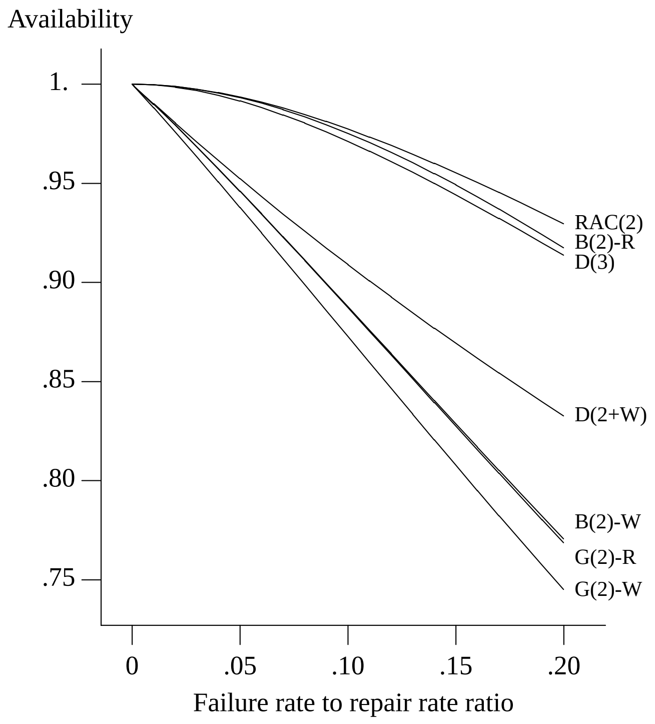

Figure 6: Compared availabilities of AC and RAC with two replicas and MCV with three.

managed by the voting with ghosts (VWG) protocol [15, 16]. Since the VWG protocol uses different quorums for reads and writes, its read and write availabilities are represented by two curves each labeled G(2)-R and G(2)-W.

Finally, the curves labeled B(2)-R and B(2)-W respectively represent the read and write availabilities of our fourth benchmark configuration, namely, two replicas on two network segments managed by voting with bystanders [13].

The graph shows the excellent performance of the RAC protocol and the disappointing performance of the DLV protocol with two replicas and a witness located on the gateway. It shows in particular that two replicas on two network segments managed by the RAC protocol always outperform three replicas managed by the best voting protocol.

4. Conclusions

Replication control protocols relying on quorum consensus require at least three replicas to provide a better availability than that of an unreplicated file. Even then, they tend to provide much poorer data availabilities than protocols that do not rely on quorum consensus, such as the available copy protocol and its variants.

We have presented here a novel replication protocol that extends the available copy approach to environments where communication failures may cause network partitions. Our *robust available copy* protocol assumes that replicas monitor the communication paths linking them with their peers and can therefore detect network partitions. As a result, each individual replica can safely establish whether it has remained up to date or is likely to have missed some updates.

We have shown that two replicas managed by the robust available copy protocol provide availability that is superior to that provided by three replicas managed by the best voting protocol. More work is still required to devise efficient implementations of the protocol.

Acknowledgements

We thank Elizabeth Pâris, for her editorial comments.

References

[1] M. Ahamad and M. H. Ammar, ‘‘Performance Characterization of Quorum-Consensus Algorithms for Replicated Data,’’ *IEEE TSE,* SE-15, 4 (1989), pp. 492-496.

[2] P. A. Bernstein and N. Goodman, ‘‘An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases,’’ *ACM TODS* 9, 4 (1984), pp. 596-615.

[3] J. Bechta Dugan and G. Ciardo, ‘‘Stochastic Petri Net Analysis of a Replicated File System,’’ *IEEE TSE,* SE-15, 4 (1989), pp. 394-401.

[4] D. Barbara, H. Garcia-Molina, and A. Spauster, ‘‘Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment,’’ *ACM TOCS* 7, 4 (1989), pp. 394-426.

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

[6] C. A. Ellis, ‘‘Consistency and Correctness of Duplicate Database Systems,’’ *Operating Systems Review,* 11 (1977).

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

[8] 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 Symp.* (1983), pp. 8-15.

[9] M. Herliyi, ‘‘A Quorum-Consensus Replication Method for Abstract Data Types,’’ *ACM TOCS,* 4, 1 (1986), pp. 32-53.

[10] S. Jajodia and D. Mutchler, ‘‘Enhancements to the Voting Algorithm,’’ *Proc. 13th VLDB Conf.* (1987), pp. 399-405.

[11] J.-F. Paris and D.D.E. Long ‘‘On the Performance of Available Copy Protocols,’’ *Performance Evaluation, 11 (1990), pp 9-30.*

[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 Bystanders,’’ *Proc. 9th ICDCS,* (1989), pp. 394-401.

[14] J.-F. Pâris, ‘‘Evaluating the Impact of Network Partitions on Replicated Data Availability,’’ *Proc. 2nd IFIP DCCA Working Conf.,* (1991).

[15] J.-F. Pâris and Qun Rose Wang, ‘‘On the Performance of Voting with Ghosts,’’ *Proc. Int. Symp. on Applied Computer Sciences,* (1993), (also available as Technical Report UH-CS-93-08).

[16] C. Pu, J. D. Noe and A. Proudfoot, ‘‘Regeneration of Replicated Objects: A Technique and its Eden Implementation,’’ *IEEE TSE,* Vol. SE-14, No. 7 (1988), pp. 936-945.

[17] R. van Renesse and A. Tanenbaum, ‘‘Voting with Ghosts,’’ *Proc. 8th ICDCS,* (1988), pp. 456-462.

[18] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel and D. C. Steere, ‘‘Coda: A Highly Available File System for a Distributed Workstation Environment,’’ *IEEE TC,* 39, 4 (1990), pp. 447-459.