Dynamic Management of Highly Replicated Data

Jehan-François Pâris Perry Kim Sloope

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

Abstract

We present an efficient replication control protocol for managing replicated data objects that have more than five replicas. Like the *grid protocol*, our *dynamic group* protocol requires only $O(\sqrt{n})$ messages per access to enforce mutual consistency among n replicas. Unlike other protocols aimed at providing fast access, our protocol adapts itself to changes in site availability and network connectivity, which allows it to tolerate $n-2$ successive replica failures.

We evaluate under standard Markovian assumptions the availability of a replicated object consisting of n replicas managed by our *dynamic group* protocol when the n replicas are on the same LAN segment and find it to equal that of an object with the same number of replicas managed by the *dynamic-linear voting* protocol.

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

1. Introduction

Many large-scale distributed systems have data that are replicated at a large number of sites within the system. Managing replicated data is a difficult task because site failures and network partitions can introduce inconsistencies among the replicas. Replication control protocols were designed to avoid this problem and provide the users with a consistent view of the data. They are said to enforce *one-copy serializability* [5].

Recent years have seen the development of several protocols specifically tailored to allow faster access to the replicated data. These protocols achieve their objective by reducing the number of sites that need to be contacted in order to validate an access request. As a result, these protocols are also much less fault-tolerant and provide mediocre data availabilities.

We present a *dynamic group* (DG) protocol that achieves both fast access and high data availability. Like the *grid protocol*, our DG protocol requires only $O(\sqrt{n})$ messages per access to enforce mutual consistency among n replicas. It differs from the grid protocol by being dynamic and reorganizing itself when it detects a change in the number of available sites or the connectivity of the network. As a result, it can tolerate $n-2$ successive replica failures while the grid protocol can fail in the presence of $\sqrt{n}$ failures.

2. Previous Work

Voting protocols [6] are the largest and best-known class of replication control protocols. Voting protocols guarantee the consistency of replicated data 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 [7]. 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 protocols. Voting has however two major disadvantages. First, it requires $2n+1$ replicas to guarantee full access to the replicated data in the presence of n simultaneous replica failures. Hence *three* replicas are required to protect against a single failure. Second, the number of messages required to collect a quorum grows linearly with the number of replicas, which makes voting protocols unsuited for managing highly replicated data.

We have seen in recent years the development of several replication control protocols specially tailored for the management of highly replicated data. These protocols include *voting with tokens* [14], the *grid protocol* [4], the *tree quorum* protocol [1], the *hierarchical quorum consensus* protocol [9, 10] and *location-based replication* [17]. The most attractive of these protocols is the grid protocol of Cheung *et al.* because it only requires $O(\sqrt{n})$ messages per access while distributing the transaction processing load among all replicas. The grid protocol organizes the set of replicas of a data object into a logical grid with p rows and m columns as in:

A	D	G	J
B	E	H	K
C	F	I	L

A read quorum consists of one replica from each column. Thus the set *{B, E, I, L}* is a read quorum while the set *{A, B, C, D, E, F, G}* is not. A write quorum consists of

all the replicas in one column plus one replica from each of the $m-1$ remaining columns.

While this organization only requires $O(m)$ messages per read and $O(p+m-1)$ messages per write, it also makes the replicated data much more vulnerable to replica failures as the simultaneous failure of all replicas within a column suffices to make the replicated data unavailable [10]. Increasing the size of the columns while reducing their number is not a solution since it would decrease the probability that a write will find at least one complete column. Instead, we propose to allow for dynamic adjustments in the grid topology to reflect changes in replica availability. These adjustments will allow the protocol to tolerate a higher number of successive site failures.

3. Dynamic Group Protocol

The *dynamic group* protocol is to operate in distributed environments where some of the sites contain full replicas of data objects. These sites can fail and can be prevented from exchanging messages by failures of the communication subnet. We assume that sites not operating correctly will immediately stop operations and that all messages delivered to their destinations will be delivered without alterations. Byzantine failures are expressly excluded.

Read and write are the two primitive operations that can be performed on the replicated object. Concurrent reads are permitted, but to guarantee one-copy serializability, writes are performed exclusive of other operations.

Our protocol organizes the replicas of a data object into small groups of equal size that are not necessarily disjoint. These groups correspond to the columns of the grid protocol. The set of all groups for a given data object will constitute a *group set* for that data object.

A read quorum consists of either one replica from every group or all replicas from one group. Hence, read access can continue when all replicas from one group are dead. A write quorum will require all replicas from one group plus one replica from every other group.

When the failure of a site is detected, the groups are dynamically rearranged in order to protect the availability of the replicated data object against subsequent failures. The number of groups may therefore decrease as sites fail, but the number of replicas in each group will remain constant. Regrouping is not permitted when a write quorum cannot be reached because a write quorum is necessary to enforce mutual exclusion. If the number of groups falls below three, then the replicated data could become unavailable with the failure of only two sites. For this reason the protocol reverts to dynamic-linear voting whenever three groups cannot be formed.

The *dynamic-linear voting* (DLV) is a dynamic counterpart of majority consensus voting [8]. The DLV protocol maintains a count of all replicas thought to be accessible and allows access to the replicated data as long as a majority of the replicas in that set remain accessible. The set is updated after a majority vote whenever the protocol detects (a) that a replica has become unreachable or (b) that a replica believed to be unreachable can be reached again. Since the set represents the majority view of the state of the replicas, it is known as the *majority block*. The DLV protocol differs from earlier dynamic voting protocols by its ability to resolve the ties that occur whenever exactly one half of the replicas in the majority block are accessible while the other members of the majority block cannot be reached. The protocol then allows access to the replicated data only if the set of accessible replicas includes the highest numbered replica in a linear ordering of the current majority block.

With the DG protocol, network partitioning may in some instances prevent a write quorum from being formed, but DG will typically continue to provide read access in at least one partition. Although write availabilities may suffer somewhat, read availabilities will be very high.

A more detailed description of our protocol follows.

3.1. Accessing the Replicated Data

The DG protocol organizes the replicas of a data object into small subsets or *groups* of replicas that are to play the same role as the columns in the grid protocol. A collection of groups is a *group set* or a *quorum set* [3]. However, the set of all minimal groups is a quorum set.

Definition 2.1 *A replica is said to be* live *if it resides on a site that is operational.*

Definition 2.2 *A replica is said to be* dead *if it resides on a site that is not operational.*

Definition 2.3 Quorum Set. *Let U be the set of all replicas of a replicated object. A set of groups* $Q \subset 2^U$ *is a* quorum set *under U iff*

(a) $H \in Q$ *implies* $H \subseteq U$ *and* $H \neq \emptyset$, *and*

(b) *there are no* $H, J \in Q$ *such that* $H \subset J$,

A *quorum set S* such that all groups $H, J \in S$ have a non-empty intersection is a *coterie* [3].

Locking one group of replicas does not guarantee exclusive access to the data since the groups do not always intersect. The intersection property can be satisfied by locking one group and one member from each group in the quorum set.

Rule 1: Write Rule. *The quorum for a write operation consists of: one complete group of live replicas from the quorum set Q* and *one live replica from every other group in Q.*

The write rule guarantees that a write will leave at least one group in the quorum set fully current and that all other groups will contain at least one current replica.

Rule 2 : Read Rule. *The quorum for a read operation consists of either one complete group of live replicas,* or *one live replica from every group in Q.*

Since a write operation will update one complete group and one replica in each group, a read quorum will contain at least one current replica.

Note that read and write quorums are not necessarily minimal. For instance, the quorum set *{{A B, C}, {D, E, F},{G, A, B}}*, which describes the configuration:

A	D	G
B	E	A
C	F	B

allows the read quorums *{A, D, G}* and *{A, D}*.

Since a read can be satisfied by accessing either all sites in a group or one site from every group, the maximum size of a read quorum is $\max(m, p)$. Similarly, a write quorum will require $(m+p-1)$ sites if all m groups are disjoint and at most $(m+p-1)$ sites if sites can belong to more than one group. Hence our protocol will require $O(\sqrt{n})$ messages per access as long as m and p remain $O(\sqrt{n})$.

Each replica will maintain some state information that will be utilized by the protocol in determining quorums. These are a version number, a group number, and a quorum set. The version number v is incremented after each successful write operation. The group number g is the ordinal number of the last successful regrouping in which the replica participated. The quorum set G represents the view the replica has of the quorum set.

```
procedure READ(d : data_object)
begin
   let U be the set of all replicas
   let G_0 be the local quorum set
   let g_0 be the local group number
   select s ∈ G_0
   ⟨R, v, g, G⟩ ← START(s)
   v_max = max_{r∈R}{v_r}
   g_max = max_{r∈R}{g_r}
   if R = s and g_max = g_0 then
      perform the read on any r : v_r = v_max
   else
      FLAG_FOR_REGROUP
      ⟨R, v, g, G⟩ ← START(U)
      v_max = max_{r∈R}{v_r}
      g_max = max_{r∈R}{g_r}
      G_max = G_r : g_r = g_max
      if ∀ s ∈ G_max, R ∪ s ≠ ∅ or ∃ s ∈ G_max, s ⊂ R then
         perform the read on any r : v_r = v_max
      fi
   fi
end READ
```

Figure 1: Read Algorithm

Figures 1, 2, and 3 present the algorithms for the read, write, and recovery operations. An explanation is necessary for several items in these algorithms. The START operation initiates the access request and returns the information needed to determine if a quorum is possible. These are R, the set of sites responding to the access request, and three vectors **v**, **g**, and **G**. These vectors respectively contain the version numbers, the group numbers, and the quorum sets of the responding replicas. FLAG_FOR_REGROUP sets a flag that indicates to the protocol that regrouping is necessary. Regrouping then begins after the access operation is complete. The COMMIT operation completes a write operation by transmitting the new version number to the updated sites.

The read algorithm is fairly simple. In order to perform a read, one group of replicas or one member of each group must be locked. We will call such a collection a *read-group*. A read request by a user will be directed to one of the sites with a replica. From that site a multicast for a read request is made to the other members of the same group. By multicasting to members of the same group as that of the requesting site, the amount of message passing is kept to a minimum. The reply will consist of the version number, the group number, and the quorum set of each of the responding sites. If a read-group is formed then the maximum version numbers and group numbers are calculated. If the group numbers of the responding sites are the same, then the read proceeds from one of the sites with the maximum version number. If a read-group is not formed or the group numbers do not match, then a read broadcast is made to the all replicas. The maximum group numbers and version numbers are then calculated. If a read-group can be formed from the sites with the maximum group number, then the read is performed from one of the sites with a maximum version number. Regrouping is then initiated since non-matching group numbers or the inability to form a read-group from any one group indicates that one or more sites have failed.

A write operation is made in a similar manner. Recall that a write must lock a full group of replicas and one replica from every other group. This is necessary to ensure that competing write-quorums are not formed. Additionally, all replicas in the write-quorum must have the maximum group number. As with a read operation, the write request is directed to one of the sites with a replica. The request is then broadcast to a selected subset of replicas that represents a write-quorum in the quorum set G_0 of the replica that is coordinating the write. This set of replicas consists of the group in which the initiating replica resides, and one randomly chosen member from each of the remaining groups. Again the replies will consist of a version number, a group number, and a quorum set. The maximum group number and version number are calculated. If all replicas respond and they have the same group number, then any replicas in the write-group that do not have current version numbers, are updated from a site with a current replica. The write is then made to the replicas forming the write-group and the algorithm terminates. At the same time, the file version numbers of those sites are incremented. If, during the course of gathering a write-quorum, replicas with group numbers lower than the calculated maximum are encountered, then these replicas did not participate in the last regrouping and do not have a current view of the quorum-set. If all replicas in the selected write-quorum did not respond then it indicates that some sites have failed or could not communi-

```
procedure WRITE(d : data_object)
begin
   let U be the set of all replicas
   let G_0 be the local quorum set
   let g_0 be the local group number
   select s ∈ G_0
   select t : ∀ z ∈ G_0−{s} t ∩ z ≠ ∅
   ⟨R,v,g,G⟩←START(s ∪ t)
   v_max = max_{r∈R}{v_r}
   g_max = max_{r∈R}{g_r}
   if R = s and g_max = g_0 then
      for all r ∈ R : v_r ≠ v_max do
        RECOVER(r)
     od
      perform the write
      COMMIT(R, v_max+1)
   else
      FLAG_FOR_REGROUP
      ⟨R,v,g,G⟩←START(U)
      v_max = max_{r∈R}{v_r}
      g_max = max_{r∈R}{g_r}
      G_max = G_r : g_r = g_max
      if ∃ T ⊂ R : (∃ s ∈ G_max : s ⊂ T and
          ∀ t ∈ G_max : t ∪ R ≠ ∅) then
        for all r ∈ T : v_r ≠ v_max do
          RECOVER(r)
          od
          perform the write
          COMMIT(T, v_max+1)
      else
        ABORT(R)
      fi
   fi
end WRITE
```

Figure 2: Write Algorithm

```
procedure RECOVER (x : replica)
begin
   let U be the set of all replicas
   repeat
      let G_0 be the local quorum set
      let g_0 be the local group number
      select s ∈ G_0
      ⟨R,v,g,G⟩←START(s)
      v_max = max_{r∈R}{v_r}
      g_max = max_{r∈R}{g_r}
      if R = s and g_max = g_0 then
         repair x from any r : v_r = v_max
      else
         FLAG_FOR_REGROUP
         ⟨R,v,g,G⟩←START(U)
         v_max = max_{r∈R}{v_r}
         g_max = max_{r∈R}{g_r}
         G_max = G_r : g_r = g_max
         if ∀ s ∈ G_max, R ∪ s ≠ ∅ or
               ∃ s ∈ G_max, s ⊂ R then
            repair x from any r : v_r = v_max
         fi
      fi
   until successful
end RECOVER
```

Figure 3: Recovery Algorithm

cate due to network partitioning. In either case a write-quorum cannot be formed from this set of replicas. A second attempt at a write-quorum is then made by contacting all sites with replicas. If a write-quorum can be formed from the responding replicas then the write proceeds as above. This time upon successful termination of the write algorithm, regrouping is initiated so that the quorum set can be updated and a high level of availability maintained.

A replica that is recovering from a site failure will have been excluded from the current quorum set if a regrouping occurred while the site was down. The purpose of the recovery algorithm is to integrate the site back into the quorum set, as well as to bring it up to date. It begins by attempting to gather a read quorum. If the attempt is successful, it determines the maximum group and file version numbers. If the maximum group number is the same as that of the recovering site, then regrouping has not taken place since the site failed and recovery continues by comparing its version number to that of the maximum version number. If they do not match, then the site retrieves a current copy from one of the sites with the maximum version number. If the version numbers match then an update is not needed and the algorithm terminates. If a read quorum can not be gathered, or the quorum set of the recovering replica is different from that of the responding sites, then a multicast is made to all replicas. Another attempt is then made to gather a read quorum and update the replica. The algorithm will repeat until recovery becomes possible. If outdated group numbers are detected during the recovery procedure, regrouping is performed after completion of the recovery.

3.2. Dynamic Regrouping

The expense of access operations is dependent on the size and number of groups in a quorum set. The groups do not necessarily need to be of equal size for this protocol to operate correctly, but groups of equal size will ensure that the cost of access operations is uniform throughout the system. Here we will only consider groups of equal size. Regrouping may result in a change in the number of groups, but the number of replicas in a group will remain constant. In general, for a given number of replicas, a larger group size increases the cost of a read operation and decreases the cost of a write operation. For example, 12 replicas could be formed into 4 groups with 3 replicas per group. This results in a minimum of 3 accesses for a read, and 6 replica updates for a write. If the configuration is 6 groups with 2 replicas per group, then a

minimum read operation is 2 accesses, and 7 replica updates for a write operation.

3.2.1. Initial Group Formation

If each group is to contain p replicas then there must be at least m groups where $m = \lceil n/p \rceil$. Additionally we will arrange the replicas in a lexicographical order. In the event that $m \times p > n$ the last group will be filled with sites of the highest lexicographical order. The lexicographical order is used to enhance the availability of the replicated file by limiting the distribution of replicas within the quorum set. This is best explained by example. Consider a configuration using 7 replicas with 3 replicas per group. The replicas are labeled A through G where A is the highest order replica and G is the lowest. It would have the following configuration:

A	D	G
B	E	A
C	F	B

If sites A and B were to fail, both read and write quorums would still be possible. Alternately if sites A and E had been chosen as the extra members of the last group, then write operations would fail if both A and E failed.

The appearance of replicas in more than one group has the benefit of causing write operations to be less expensive for some accesses. It also results in a more uneven distribution of the load since the sites appearing in more than one group will be called on more frequently for read operations. Since one of the goals of this protocol is to more evenly distribute the load, we will attempt to keep the number of replicas appearing in more than one group to a minimum.

3.2.2. Dynamic Group Reformation

Dynamic regrouping is initiated when one or more sites fail to respond to an access request, or when a site with a replica recovers from a failure. The host that is recovering from a failure or that performed the access operation initiates the regrouping by making a multicast for a regroup to all sites with replicas. The responding sites are locked to prevent response to any access requests or other regrouping requests. It is then determined if a write quorum can be formed from the responding replicas. This is necessary to ensure that competing quorum sets can not be formed in the event of network partitioning. If a write quorum can not be formed, regrouping is aborted. If a quorum is formed, regrouping proceeds in the same manner as in initial group formation, using only the replicas that responded to the regrouping request. The number of groups in the new quorum set is calculated based on the number of responding replicas and the number of replicas per group. As stated previously, the number of replicas per group is a constant value that does not change, only the number of groups in a quorum set may change. This may result in the loss or addition of one or more groups. Once the new quorum set is formed, one group of replicas and one replica from each group in the new quorum set are selected for updating, and are updated if needed. The updating is made from one of the current replicas that was in the write quorum in the previous quorum set. This is necessary because write operations do not update all replicas, but only those in the write quorum. Thus regrouping can result in the formation of groups that do not contain a current copy. By performing this updating, it is guaranteed that future access operations will always find at least one current replica. At this point the group number is incremented, and the new quorum set and group number are multicast to the participating replicas.

The following example serves to illustrate dynamic regrouping:

In this configuration all n replicas reside on the same LAN and $n = 10$ and $p = 3$. The value of m will then be 4. Since $m \times p > n$ the grouping will be as follows:

A	D	G	J
B	E	H	A
C	F	I	B

Given this arrangement a read operation will involve at least 3 sites and a write operation will involve at least 5 sites but not more than 6 sites. Suppose site A fails or becomes unreachable after a network partition. The number of groups is recalculated based on 6 replicas with 2 per group. This results in the loss of one group. We then have:

B	E	H
C	F	I
D	G	J

Reads will still involve 3 sites and writes will involve 5 or 6 sites. If site B were to fail then regrouping would again reoccur.

C	F	I
D	G	J
E	H	C

If site E were then to fail or to become unreachable it would be replaced by F in the first group and H in the second group.

C	G	J
D	H	C
F	I	D

The next replica failure would leave only 6 sites. Regrouping would result in the formation of only 2 groups. The protocol would then revert to dynamic linear voting.

4. Availability Analysis

In this section we present an analysis of the availability provided by our protocol. We will assume here that the availability of a replicated data object is the stationary probability of the object being in any state permitting access. $A_S(n)$ will denote the availability of an object 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 LAN segments linked by gateways. When a site fails, a site 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, in fact, have been made in most probabilistic studies of the availability of replicated data [2, 8, 12-13]. Combinational models that do not require any assumptions about failure and repair distributions have been proposed [15-16] but these models cannot distinguish between available states and recovery states.

The availability analysis of a replicated object subject to network partitions is complicated by the fact that replicas that cannot communicate with each other cannot recover from each other. As a result, the number of potential states of a replicated object subject to network partitions is much larger than when partitions are not considered. Hence, most recent studies of the availability of replicated data have either relied on simulation models or have totally neglected communication failures. We will further assume that the replicated data are frequently accessed and that site failures will always be rapidly detected.

Consider a replicated data object managed by the grid protocol and let us assume that its *n* replicas are arranged in *p* rows of *m* replicas each. The availability *A* of a single replica is linked with the failure and repair rates λ and μ by the well-known relation:

$$A = \frac{\mu}{\mu + \lambda} = \frac{1}{1+\rho},$$

where $\rho = \lambda / \mu$ [14]. The availability of the replicated data for the read operation $A_G^R(m,\ p)$ is equal to the probability that all columns contain at least one live replica:

$$A_G^R(m,\ p) = (1-(1-A)^p)^m$$

The availability of the replicated data for the write operation $A_G^W(m,\ p)$ is equal to the probability that all columns contain at least one live replica minus the probability that no column is complete:

$$A_G^W(m,\ p) = (1-(1-A)^p)^m - (1-A^p-(1-A)^p)^m$$

Consider now the same replicated object managed by the DG protocol. Every site failure reduces by one the number of live replicas and results in an attempt to regroup the remaining live replicas. Since the group assignments provided by the DG protocol never allow a replica to belong to all groups in the quorum set, the failure of a single replica cannot affect the availability of a write quorum. The regrouping will therefore succeed unless the remaining number of live replicas is $\leq 2p$, in which case the protocol reverts to a dynamic-linear voting protocol. Even then, the dynamic-linear voting protocol guarantees that the replicated data will remain protected against successive replica failures as long as there are at least two live replicas. Should one of the last two live replicas fail, the lexicographic order of the remaining live replica will be compared with that of the failed replica. There are two cases to consider:

(1) the failed replica has the higher value, which implies that the surviving replica cannot form a new majority and the replicated data become unavailable, and

(2) the failed replica has the lower value and the opposite conclusion is reached: the surviving replica forms a new majority and the replicated data remain available.

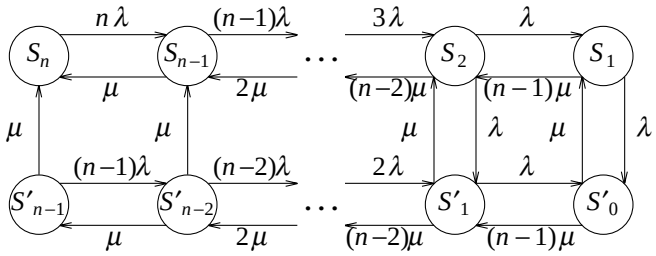

Figure 4: Transition Diagram for DG

Figure 4 contains the state transition rate diagram for a replicated object with *n* replicas managed by the DG protocol. Note that left-to-right and top-to-bottom transitions represent site failures while right-to-left and bottom-to-top transitions indicate site repairs. State S_n represents the state of the replicated object when all its *n* replicas are live. A failure of one of these *n* replicas brings the replicated object in state S_{n-1}. The failure will almost immediately result in a group reassignment since there are enough live replicas to achieve a write quorum. Successive site failures would bring the replicated object from state S_{n-1} to state S_{n-2} then from state S_{n-2} to state S_{n-3} and so forth until either one of the failed replicas recovers or the object reaches state S_2. Whenever the replicated object is in state S_2 and one of its last two live replicas fail, two transitions are possible:

(1) if the lexicographic order of the replica that failed is lower than that of the last live replica, the replicated object remains available and moves to state S_1;

(2) if the lexicographic order of the replica that failed is higher than that of the last live replica, the replicated object becomes unavailable and moves to state S'_1.

State S'_0 represents the state of the replicated object after all its replicas have failed. Recovering from state S_0 would bring the replicated object in state S_1 if the site that recovers has the higher ranked replica in the last majority partition or in state S'_1 if this is

not the case. Finally states S'_2 to S'_{n-1} represent the states of the replicated object when there are 2 to $n-1$ live replicas but the replicated object remains unavailable because the highest ranked of the last two live replicas when the replicated object was last in state S_2 has not yet recovered.

Consider now the equilibrium equation for the subsets of states S_n, S_{n-1}, ..., S_2, S_1 and S'_{n-1}, S'_{n-2}, ..., S'_1, S'_0:

$$\mu \sum_{j=0}^{n-1} p'_j = \lambda(p_1 + p_2)$$

where p_i is the probability for the replicated object being in state S_i.

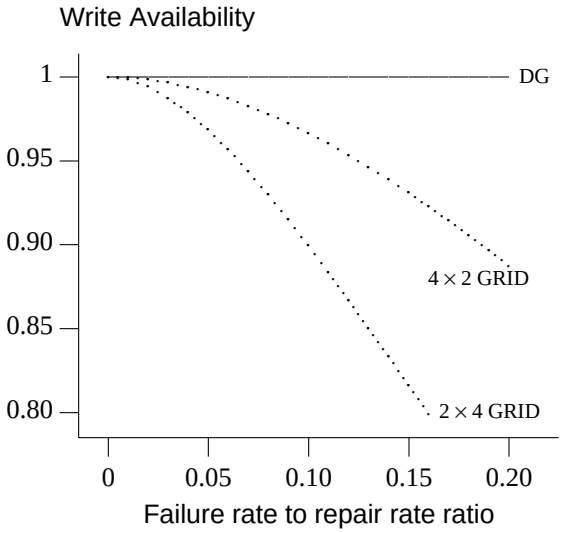

Figure 5: Availabilities for Eight Replicas

Observing that $p_1 + p'_1$ is the probability of having exactly *one* live replica out of *n*, we have

$$p_1 + p_1' = \frac{\binom{n}{1}\rho^{n-1}}{(1+\rho)^n}.$$

Similarly,

$$p_2 + p_2' = \frac{\binom{n}{2}\rho^{n-2}}{(1+\rho)^n}.$$

An upper bound for the probability of being in any of the *n* non-available states is then given by:

$$\sum_{j=0}^{n-1} p'_j < \rho\left(\frac{\binom{n}{1}\rho^{n-1}}{(1+\rho)^n} + \frac{\binom{n}{2}\rho^{n-2}}{(1+\rho)^n}\right)$$

Hence,

$$A_{DG}(n) = 1 - \sum_{j=0}^{n-1} p'_j > 1 - \frac{\binom{n}{1}\rho^n + \binom{n}{2}\rho^{n-1}}{(1+\rho)^n}$$

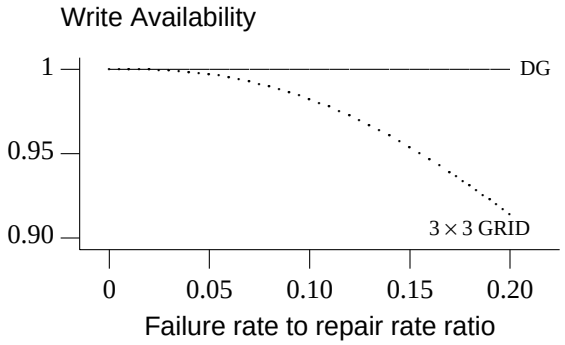

Figure 6: Availabilities for Nine Replicas

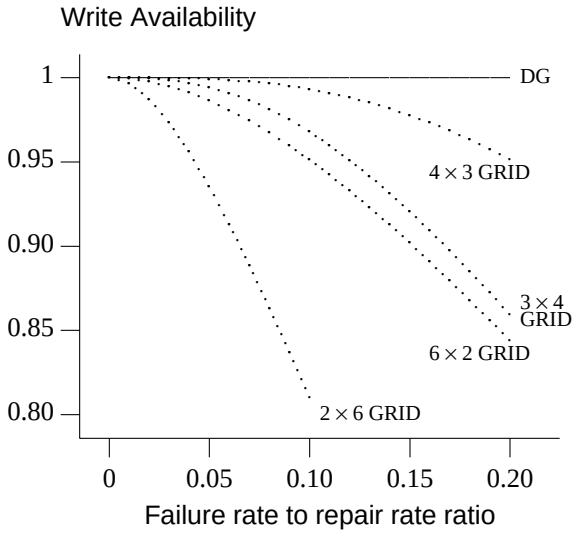

Figure 7: Availabilities for Twelve Replicas

Figures 5 to 7 shows the compared availabilities of the dynamic groups and grid protocols for eight, nine and twelve replicas respectively. All these availabilities were computed 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 solid line on the top of each graph represents the availability of the DG protocol while the dotted lines give the *write* availabilities of the grid protocol for various grid organizations. (*Read* availabilities are much higher.)

These three graphs display the excellent performance of our protocol. We need however to qualify these findings. Under the assumptions of independent failures, no partitions and frequent updates, the availability of the DG protocol is indeed equal to that of

the dynamic-linear voting protocol for the same number of replicas. The DG protocol would perform somewhat worse in situations where correlated replica failures and network partitions are present. For instance, the simultaneous failure of p replicas belonging to the same group would immediately disable write access while preventing the DG protocol from regrouping the remaining live replicas. A network partition that would isolate a whole group of replicas from the remainder of the quorum set would have the same effect. There are preventive strategies that can minimize the probability of such events occurring. One of these strategies is to spread groups over the network in such a way that no group entirely consists of replicas likely to fail simultaneously nor to be partitioned from the remainder of the quorum set. This may result in increased network traffic through the network gateways.

Another issue that was not considered in our model is the impact of the access rate on the performance of the protocol. Since the DG protocol only detects site failures and network partitions when an access is attempted, replicated data that are infrequently accessed are less likely to have their replicas promptly regrouped after a site failure or a network partition. An earlier investigation of the effect of access rates on the availability of dynamic voting protocols concluded that access rates of the order of one access per hour were enough to provide timely detection of most site failures and network partitions [11]. We can therefore conjecture that the access rates required to achieve a good performance with the DG protocol could be easily guaranteed by scheduling a few dummy accesses every hour.

5. Conclusions

We have presented an efficient replication control protocol for managing replicated data objects that have more than five replicas. Like the grid protocol, our *dynamic group* protocol requires only $O(\sqrt{n})$ messages per access to enforce mutual consistency among n replicas. It differs from the grid protocol by reorganizing itself every time it detects a change in the number of available sites or the connectivity of the network. As a result, it can tolerate $n-2$ successive replica failures and provides a data availability comparable to that of the dynamic-linear voting protocol.

More work needs to be done to evaluate the impact of simultaneous failures and network partitions and to devise grouping strategies minimizing the likelihood that any such event could disable access to the replicated data.

Acknowledgements

We thank Elizabeth Pâris for her editorial comments.

References

[1] D. Agrawal and A. El Abbadi, ‘‘The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data,’’ *Proc. 16th VLDB Conf.*, (1990).

[2] M. Ahamad and M. H. Ammar, ‘‘Performance Characterization of Quorum-Consensus Algorithms for Replicated Data,’’ *IEEE TSE*, Vol. SE-15, No. 4 (1989), pp. 492-496.

[3] D. Barbara and H. Garcia-Molina, ‘‘Mutual Exclusion in Partioned Distributed Systems,’’ *Distributed Computing*, Vol. 1 (1986), pp. 119-131.

[4] S. Y. Cheung, M. Ahamad and M. H. Ammar, ‘‘ The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data,’’ *Proc. 6th ICDE*, (1990), pp. 438-445.

[5] S.B. Davidson, H. Garcia-Molina, and D. Skeen, ‘‘Consistency in Partitioned Networks,’’ *ACM Computing Surveys*, Vol. 17, No. 3 (1985), pp. 341-370.

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

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

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

[9] A. Kumar, ‘‘Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data,’’ *IEEE TC*, Vol. TC-40, No. 9 (1990), pp. 996-1004.

[10] A. Kumar and S.Y. Cheung, ‘‘A High Availability $\sqrt{N}$ Hierarchical Grid Algorithm for Replicated Data,’’ *Information Processing Letters*, Vol. 40, (1991), pp. 311-316.

[11] D. D. E. Long and J.-F. Pâris, ‘‘A Realistic Evaluation of Optimistic Dynamic Voting,’’ *Proc. 7th SRDS*, (1988), pp. 129-137.

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

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

[14] J.-F. Pâris, ‘‘Efficient Management of Replicated Data,’’ *Proc. 2nd Int. Conf. on Database Theory*, LNCS # 326, Springer Verlag (1988), pp. 386-409.

[15] 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.

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

[17] P. Triantafillou and D.J. Taylor, ‘‘A New Paradigm for High Availability and Efficiency in Replicated Databases,’’ *Proc. 2nd IEEE SPDS*, (1990), pp. 136-143.

BLANK PAGE