The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data

Nabor das Chagas Mendonça
and
Ricardo de Oliveira Anido

Relatório Técnico DCC–02/93

Fevereiro de 1993

The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data

Nabor das Chagas Mendonça
and
Ricardo de Oliveira Anido

Abstract

Voting is the traditional mechanism used for maintaining the consistency of replicated data in distributed systems. One of the problems involved in voting mechanisms is the size of the quorums needed on each access to the data. To reduce the quorum size needed for voting, some protocols for copy consistency organize the copies into some logical structure, and use the this structure during the voting process.

We present two new protocols for maintaining copy consistency. Both protocols organize the copies into a ring structure, and use the adjacency property to reduce the read and write quorums. The *flat ring* protocol uses one single ring and achieves a read quorum of two copies (constant), and a write quorum equal to the majority of copies. The *hierarchical ring* protocol is a generalization of the flat ring protocol and uses a multi-level ring structure. The read quorum in the hierarchical ring protocol is independent on the total number of copies, and the write quorum is, in general, smaller than the majority of the copies. Both protocols are tolerant to multiple failures and do not need any special reconfiguration procedures when failures occur.

1 Introduction

Data replication is usually used to improve reliability in distributed systems. Keeping copies of the same logical data at different sites avoids that the data become inaccessible due to failures in the system, making the system more reliable. Another potential advantage is better access response time, since the data may be accessed from a nearer site, or even locally.

One problem that must be solved when using replication is how to maintain the copies in a consistent state when multiple accesses are being made. There must exist a control protocol responsible for synchronizing the accesses so that the logical data is kept consistent. This protocol must also be tolerant to processor and network failures.

Voting was one of the first mechanisms used to maintain consistency of replicated data. A simple protocol that uses voting is to allow access to the data item only if permissions from the majority of copies are obtained [18]. A variation of this simple protocol is to assign votes to each copy, and to allow access if a majority of the total number of votes is obtained [12]. In both cases, the quorum required for reading and writing may vary, provided that the sum of read and write quorums is greater than the total of votes, and that the write quorum is at least equal to the majority of the votes. Other variations of the voting protocol have been proposed, such as dynamic voting [7, 13, 17], voting with witnesses [16], voting based on virtual partitions [1], voting with fragmentation [3] and multidimensional voting [5].

The main problem in protocols that use voting is the size of the quorums needed on each access to the replicated data. To solve this problem, some authors have proposed to organize the copies into some logical structure, and then use this structure to reduce the quorum sizes [2, 9, 14, 4]. In [9] the copies are organized into a $\sqrt{n} \times \sqrt{n}$ grid (where n is the number of copies), reducing the quorums to size $O(\sqrt{n})$; [14] uses hierarchical quorum consensus to reduce the quorum size to $n^{0.63}$; [4] reduces the quorum to size $O(\log n)$ by organizing the copies into a tree structure.

We propose a new scheme to maintain copy consistency that also uses a logical structure to construct the quorums. The copies are organized into a ring structure, and the adjacency properties are used to reduce considerably the read quorum without increasing unacceptably the write quorum. We present two protocols. In the first one the copies are organized into a single ring. The read quorum is constant, equal to two, and the write quorum is the majority of the alternating copies in the ring. Availability is high for read operations but low for the case of write operations with a large number of copies. We call this the *flat* ring protocol.

In the second protocol the copies are organized into a multi-level hierarchical ring structure, where the elements in a ring at one level are rings at the level immediately below, and the copies are the elements at the lowest level rings. The read quorum is exponential with number of levels, independent on the number of copies; the write quorum depends on the number of elements at each level, and is always less than or equal to the majority of the copies. We call this the *hierarchical* ring protocol. Compared to the flat protocol, the hierarchical protocol achieves a much better write availability with a slight decrease in read availability. Both protocols are tolerant to failures and network partitioning, with the advantage that in case of failures there is no need to reconfigure the logical structure.

In the next section we describe the system model adopted. Section 3 gives the motivation for this work, and describes the protocols in detail. A discussion about the availability provided by the protocols is presented in section 4. Section 5 compares the hierarchical protocol with related work. Finally, section 6 presents the conclusions.

2 Model

A distributed system consists of a collection of independent processors which communicate only by exchanging messages through a communication system. Each processor has its own local memory, and is also referred as a *node*. A node may become inaccessible due to its own fai-

lure, to other nodes failures, or to failures in the communication system[1]. Failures may lead to network partitioning, where the system is divided in groups of nodes called partitions [10]. Nodes within the same partition can communicate, but nodes from different partitions become inaccessible to each other.

Data is replicated by storing copies of the same logical data item at different nodes. Two operations, read and write, are allowed on replicated data. Before performing the operation a node must obtain permission from a number of copies (quorum) using a control protocol.

To represent the quorums we will use the *coterie* notation [11]. Let S be the set of nodes that have a copy of the replicated data. A *coterie* over S is defined as a pair of sets $W = \{w_1, \ldots, w_n\}$ and $R = \{r_1, \ldots, r_m\}$ such that their elements are subsets of S and have the following properties:

1. Minimality: there is no pair (a, b) of elements in W (R) such that a is a proper subset of b.

2. Intersection: every pair of elements in W has a non-empty intersection, and any element in R has a non-empty intersection with every element in W.

Sets R and W will represent, respectively, the read (q_r) and write (q_w) quorums.

In relation to protocol correctness we will adopt the *one-copy serializability* [8] criterion, which states that, in order to maintain the consistency in replicated copies, two write operations, or one write operation and one read operation, cannot take place independently. Note that the quorums constructed following the definition of coteries satisfy this criterion.

[1] We consider that both nodes and communication system components are *fail-stop*, i.e., they do not behave maliciously but rather become inoperative in the presence of failures.

3 The Ring Protocols

This section gives the motivation for this work, and describes the flat protocol and the hierarchical protocol in detail.

3.1 Motivation

All existing control protocols for replicated data rely on the tradeoff between the cost for reading, the cost for writing, data availability, and the protocol tolerance to node failures. For example, the read-one write-all scheme needs only one copy as read quorum, but has the inconvenience of having a write quorum equal to the total number of copies (thus not tolerating a single node failure)[2].

The main motivation for our work was to develop a protocol which had a constant (low) cost for reading, while maintaining an acceptable cost for writing, since we are interested in systems where read operations are much more frequent than write operations.

We will present the flat protocol first; the hierarchical ring protocol, being a generalization of the flat protocol, will be presented later.

3.2 The Flat Ring Protocol

In this protocol, all copies of a replicated data item are organized into a ring structure. Read and write quorums will be obtained by specific algorithms for quorum construction. These algorithms use the adjacency information to guarantee quorum intersection, and to maintain the quorum sizes small.

A read quorum is formed by getting access permission from any two adjacent copies in the ring. A write quorum is formed by getting access permission from half of alternating copies plus any other copy, thus requiring the majority of the total number of copies.

As an example, consider a replicated data item with six copies (figure 1). The following sets of nodes are eligible for read quorum: {1,2}, {2,3},

[2] We compare our protocol to existing ones, relative to this tradeoff, in section 5.

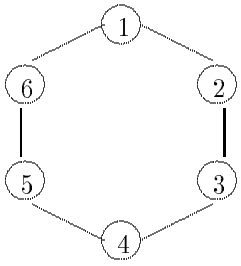

Figure 1: Six copies organized into a logical ring structure.

{3,4}, {4,5}, {5,6} e {6,1}. Sets eligible for write quorum are: {1,3,5,6}, {1,3,5,4}, {1,3,5,2}, {2,4,6,1}, {2,4,6,3} e {2,4,6,5}.

Notice that the eligible quorums are coteries, satisfying the minimality and intersection properties[3].

3.2.1 The Ring Structure

Let $S = \{s_1, \ldots, s_n\}$ be the set of nodes that store a copy of the replicated data item. A ring structure is easily imposed on it by defining the successor ($\mathsf{Suc}(i)$) and predecessor ($\mathsf{Pred}(i)$) operations. These operations use the existing adjacency relations in S (s_1 precedes s_2 and s_2 succeeds s_1, for example) and the fact that, in a ring, s_1 succeeds s_n and s_n precedes s_1. Both operations receive as an argument the position i ($1 \leq i \leq n$) of a node $s_i \in S$, and return i successor or predecessor, depending on the operation (figure 2).

Another operation defined is $\mathsf{GetPermission}(i)$, which returns the value $TRUE$ (success) if node s_i allows access to its own copy of the item. $\mathsf{GetPermission}(i)$ returns $FALSE$ when either node s_i refuses access or it cannot be contacted (due to s_i's own failure or network partitioning).

[3] The fact that the quorums are distinct and have the same size shows that they satisfy the minimality property; the intersection property will be shown later, when proving the protocol correctness.

```
Pre(pos)
{
  if (pos == 1)
    return(n)
  else
    return(pos-1)
}
```

```
Suc(pos)
{
  if (pos == n)
    return(1)
  else
    return(pos+1)
}
```

Figure 2: The successor and predecessor operations in the flat ring protocol.

3.2.2 Quorum Construction

To obtain a read quorum it suffices to get access permission from any pair of adjacent copies in the ring. The read construction quorum algorithm (figure 3) will traverse the ring until either a quorum is obtained or all copies have been examined (in which case the quorum was not obtained and the request for reading is refused).

A write quorum is obtained by getting access permission from the majority of copies in such a way that every pair (s_i, s_j) of adjacent nodes in the ring has at least one node in the quorum ($s_i \in q_w$ *or* $s_j \in q_w$). The algorithm is shown in figure 4.

3.2.3 Proof of Correctness and Non-equivalence with Vote Assignment

As mentioned in section 2, in order to show that the protocol is correct it is sufficient to prove that it implements *one-copy serializability*. In other words, we must show that it does not permit two conflicting operations (two write operations, or one write and one read operations) to occur at the same time. We do that by proving the following lemmas.

Lemma 1 *In a ring of size n, the flat protocol guarantees that there is a non-empty intersection between any read and write quorums.*

```
GetReadQuorum()
{
    quorum = num_examined = 0
    copy = i (1 ≤ i ≤ n)
    while ((quorum == 0) && (num_examined < n)){
        if (GetPermission(copy)){
            copy = Suc(copy)
            quorum = GetPermission(copy)
            num_examined++
        }
        copy = Suc(copy)
        num_examined++
    }
    return(quorum)
}
```

Figure 3: The algorithm for constructing read quorums in the flat ring protocol.

Proof. It follows from the quorum construction algorithms. The read quorum is formed by two adjacent nodes in the ring; the write quorum construction algorithm guarantees that there is no pair of adjacent nodes in the ring so that both nodes in the pair are not part of the write quorum. Therefore, there is always a non-empty intersection between any read and write quorums. □

Lemma 2 *In a ring of size n, the flat protocol guarantees that there is a non-empty intersection between any write quorums.*

Proof. It follows from the fact that the write quorum is formed by the majority of copies ($\lfloor \frac{n}{2} \rfloor + 1$) in the ring. Since $2(\lfloor \frac{n}{2} \rfloor + 1) > n$, it is guaranteed that the intersection between any write quorum is non-empty. □

An interesting property of the flat protocol is that the *coterie* it generates cannot be generated by any vote assignment in the voting protocol [12].

```
Try(copy)
{
    quorum = fail = num_examined = 0
    while ((!fail) && (num_examined < ⌊n/2⌋)){
        if (GetPermission(copy)){
            copy = Suc(Suc(copy))
            num_examined++
        }
        else fail = 1
    }
    if (fail) return(0)
    else{
        if (!odd(n)) copy = Pred(copy)
        quorum = GetPermission(copy)
        return(quorum)
    }
}

GetWriteQuorum()
{
    quorum = num_examined = 0
    copy = i (1 ≤ i ≤ n)
    while ((!quorum) && (num_examined < n)){
        quorum = Try(copy)
        copy = Suc(copy)
        num_examined++
    }
    return(quorum)
}
```

Figure 4: The algorithm for constructing write quorums in the flat ring protocol.

Lemma 3 *There is no vote assignment equivalent to the flat protocol.*

Proof. By contradiction. Consider the ring shown in figure 1. Let $v_1, \ldots, v_6$ be the votes assigned to the 6 copies and V_t the total number of votes. Consider the two eligible write quorum sets of copies {1,3,4,5} and {1,2,4,6}, and two other sets that are not eligible quorums {1,2,3,4} and {1,4,5,6}. For a vote assignment to be equivalent to the flat protocol the following must hold:

$$\left.\begin{array}{lcll} v_1 + v_3 + v_4 + v_5 & > & V_t/2 & (1) \\ v_1 + v_2 + v_4 + v_6 & > & V_t/2 & (2) \end{array}\right\} \text{there is a quorum}$$

$$\left.\begin{array}{lcll} v_1 + v_2 + v_3 + v_4 & < & V_t/2 & (3) \\ v_1 + v_4 + v_5 + v_6 & < & V_t/2 & (4) \end{array}\right\} \text{there is not a quorum}$$

Solving (1) and (3) we conclude that $v_2 < v_5$. Solving (2) and (4) we conclude that $v_5 < v_2$, which is a contradiction. Therefore, there is no vote assignment (using positive integers) that satisfies both conditions, and the lemma follows. □

3.3 The Hierarchical Ring Protocol

The hierarchical protocol is a generalization of the flat ring protocol. The structure used is a multi-level hierarchy of rings, where the elements at the lowest level rings are the data item copies, and the elements at higher levels are rings of the level immediately below. Rings at different levels may have varying number of elements, but rings at the same level have always the same number of elements — the total number of copies will be the product, over all levels, of the number of elements in the rings at each level. (figure 5).

Quorum construction is similar to the flat protocol, but happens level by level, from the lowest to the highest. A read quorum, for example, would be formed by getting read access permission from any two adjacent elements at the highest level. To get read access permission from an element at one level means to form a read quorum in the ring corresponding to that element. This procedure is repeated recursively until

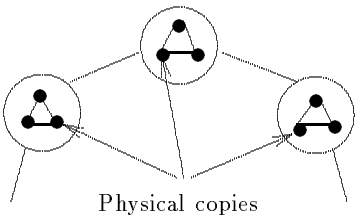

Figure 5: An hierarchical logical ring structure.

the lowest level is reached. At the lowest level the access permission is asked from the copies. The write quorum is constructed in a similar way.

Consider the example shown in figure 6. The replicated data item has 15 copies organized in a 2-level ring hierarchy. At the first level the copies are grouped in rings with three elements (copies) each. At the second level the ring is composed by five elements (rings of the first level). Some eligible read quorums are {1,2,13,14}, {2,3,4,5} and {7,8,11,12}, from 45 possible quorums. Some eligible write quorums are {1,2,7,8,10,11}, {4,5,11,12,14,15} and {7,9,13,15,2,3}, from 135 possible quorums. Notice that the eligible quorums in the hierarchical protocol are also *coteries*.

In the traditional voting protocol with 15 copies and a read quorum of four copies, there would be necessary a write quorum of at least 12 copies. In the hierarchical protocol a write quorum of only six copies is enough.

3.3.1 The Hierarchical Ring Structure

Let $S = \{s_1, \ldots, s_n\}$ be the set of nodes that store a copy of a replicated data item. A hierarchical ring structure is imposed on this set by organizing the copies into a L-level hierarchy in such a way that copies are elements at level 0, and others logical elements (rings) are defined at higher levels. An element at level 1 is a ring composed by m_1 copies. An element at level 2 is a ring composed by m_2 elements at level 1. Gene-

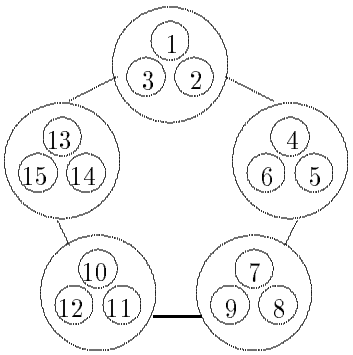

Figure 6: Fifteen copies organized in a 2-level ring hierarchy.

ralizing, an element at level i is a ring composed by m_i elements at level $i-1$ ($1 \leq i \leq L$). The total number of copies is given by $\prod_{i=1}^{i=L} m_i$.

To simplify the description of the quorum construction algorithms we use the following notation: an element position at level i is defined uniquely only with respect to the element to which it belongs at level $i+1$ (figure 7). This notation is used to simplify a reference to any element at any level by the quorums constructions algorithms. An element of a ring at level i, for example, is identified by its position p ($1 \leq p \leq m_i$) inside that ring.

Since rings in the hierarchical protocol may have a different number of elements, we have to modify the successor and predecessor operations defined in the flat protocol, introducing the size of the ring as a new argument (figure 8). The operation GetPermission(i) remains the same, with $1 \leq i \leq n$.

3.3.2 Quorum Construction

In the hierarchical ring protocol the quorums are formed iteratively, level by level. Quorum construction at each level of the ring hierarchy is similar to quorum construction at the flat protocol. A read quorum at

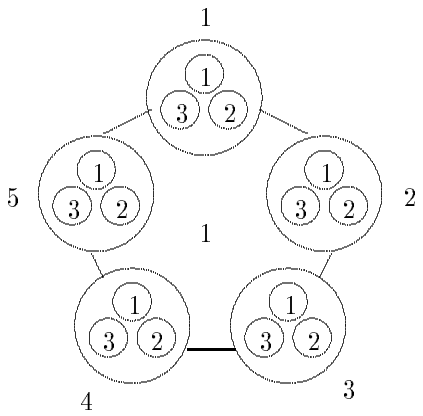

Figure 7: Element positions with respect to the next higher level.

```
Pre(pos,size)
{
    if (pos == 1)
        return(size)
    else
        return(pos-1)
}
```

```
Suc(pos,size)
{
    if (pos == size)
        return(1)
    else
        return(pos+1)
}
```

Figure 8: Successor and predecessor operations in the hierarchical ring protocol.

a level i is formed by successfully constructing two other read quorums in two adjacent elements at level $i-1$. A write quorum at level i is formed by successfully constructing other write quorums in a majority of alternating elements at level $i-1$.

The read and write quorum construction algorithms are defined recursively. A request for access permission to an element at level i is implemented by calling the algorithm recursively for the corresponding elements at level $i-1$. At level 0, the request is made directly to the copies. Two arguments are needed to the algorithm: the current level and the global position (relative to level L) of the first element at the current level. The former identifies the level where the algorithm is currently trying to form the quorum. The latter is used to calculate the global position of a copy at level 0 and will be the argument of the GetPermission(i) operation. Figures 9 and 10 show, respectively, the read and write quorum construction algorithms.

3.3.3 Quorum Size

Let L be the number of levels in the ring hierarchy, $m_i (1 \leq i \leq L)$ be the number of elements at level $i-1$ that compose an element at level i, and $n = \prod_{i=1}^{i=L} m_i$ be the total number of copies. The read and write quorum sizes in the hierarchical ring protocol are:

$$q_r = 2^L \qquad and \qquad q_w = \prod_{i=1}^{i=L} \lfloor \frac{m_i}{2} \rfloor + 1$$

Notice that the read quorum size is exponential with L and is independent from the values of n and m_i. Therefore, to reduce the read quorum we should minimize L. The larger the value of m_i, the smaller will be the proportion q_r/n. The write quorum is always less than or equal to the majority of the total number of copies ($\prod_{i=1}^{i=L} \lfloor \frac{m_i}{2} \rfloor + 1 \leq \lfloor (\prod_{i=1}^{i=L} m_i)/2 \rfloor + 1$), and decreases when L increases.

In a system where the number of read operations is much larger than the number of write operations, the value L must be set to the as large

```
ReadQuorum(level,first)
{
    if (level > 0){
        quorum = num_examined = 0
        element = i (1 ≤ i ≤ m[level])
        while ((quorum) && (num_examined < m[level])){
            if (ReadQuorum(level-1,(m[level-1]*(first+element-2))+1)){
                element = Suc(element,m[level])
                quorum = ReadQuorum(level-1,(m[level-1]*(first+element-2))+1)
                num_examined++
            }
            element = Suc(element,m[level])
            num_examined++
        }
        return(quorum)
    }
    else return(GetPermission(first))
}

GetReadQuorum()
{
    return(ReadQuorum(L,1))
}
```

Figure 9: Algorithm for constructing read quorums in the hierarchical ring protocol.

```
Try(element,level,first)
{
    if (level > 0){
        quorum = fail = num_examined = 0
        while ((!fail) && (num_examined < ⌊m[level]/2⌋)){
            if (WriteQuorum(level-1,(m[level-1]*(first+element-2))+1)){
                element = Suc(Suc(element,m[level]))
                num_examined++
            }
            else fail = 1
        }
        if (fail) return(0)
        else{
            if (!odd(m[level])) element = Pred(element,m[level])
            quorum = WriteQuorum(level-1,(m[level-1]*(first+element-2))+1)
            return(quorum)
        }
    }
    else return(GetPermission(first))
}

WriteQuorum(level,first)
{
    quorum = num_examined = 0
    element = i (1 ≤ i ≤ m[level])
    while ((!quorum) && (num_examined < m[level])){
        quorum = Try(element,level,first)
        element = Suc(element,m[level])
        num_examined++
    }
    return(quorum)
}

GetWriteQuorum()
{
    return(WriteQuorum(L,1))
}
```

Figure 10: Algorithm for constructing write quorums in the hierarchical ring protocol.

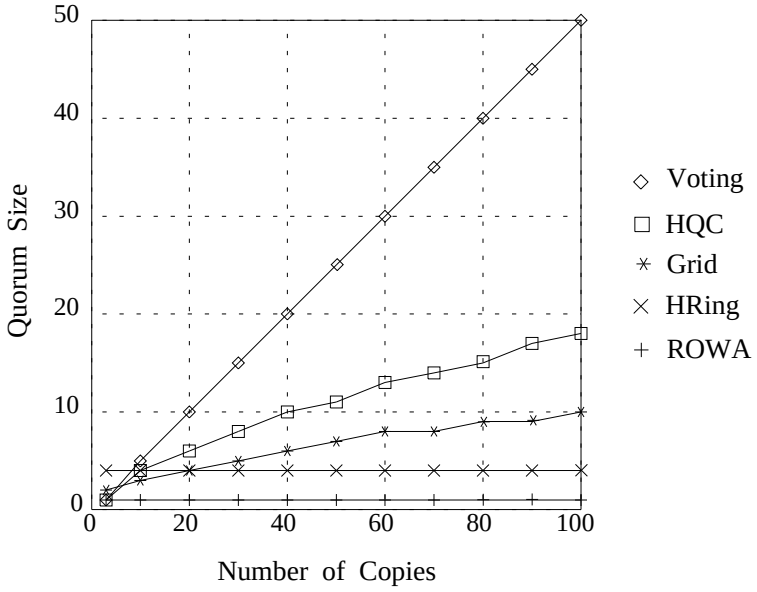

Figure 11: Expected read quorum size.

as possible, considering the resulting read quorum size. Another factor to consider when determining the values of L and m_i is data availability. Data availability of read and write operations in the flat protocol and in the hierarchical protocol will be discussed in section 4.

A particular instance of the hierarchical protocol is the case where the number of elements per ring is the same at all levels. Let $m_i = d$ $(1 \leq i \leq L)$. The number of levels is $L = \log_d n$ and the resulting quorum sizes are:

$$q_r = n^{\log_d 2} \qquad and \qquad q_w = (\lfloor \frac{d}{2} \rfloor + 1)^{\log_d n}$$

In this case, if we make, for example, $d = \sqrt{n}$, we have $L = 2$, $q_r = 4$ and $q_w = \frac{n}{4} + \sqrt{n} + 1$ (for d even) or $q_w = \frac{n+2\sqrt{n}+1}{4}$ (for d odd) — a small

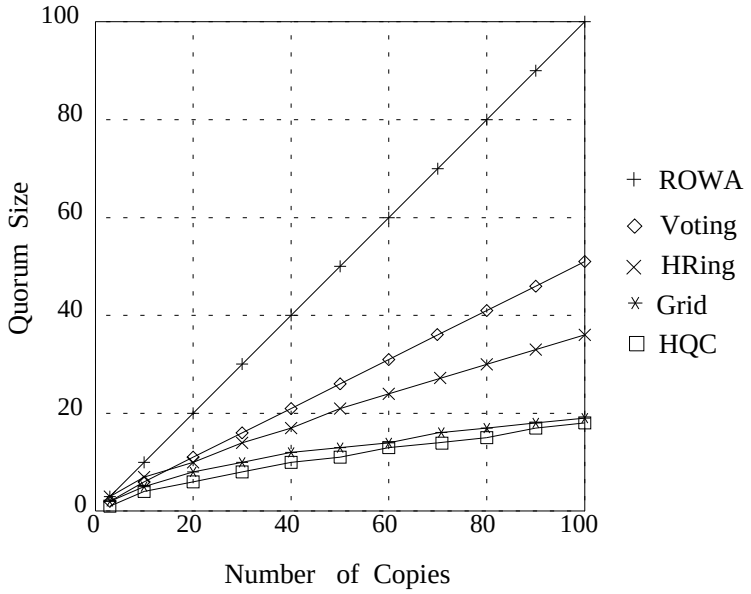

Figure 12: Expected write quorum size.

and constant read quorum, and a write quorum which is asymptotically less than the majority of the total number of copies.

Figures 11 and 12 show comparative plots of the expected read and write quorum sizes. We compare the hierarchical protocol (with $L = 2$; $m_1 = m_2 = \sqrt{n}$) against the read-one write-all scheme (ROWA), voting [18], hierarchical quorum consensus (HQC) [14] and the grid protocol [9]. Considering the tradeoff between the read and write operations costs for a large number of copies, in systems where read operations are more frequent, the hierarchical protocol has outstanding advantages.

3.3.4 Proof of Correctness and Non-equivalence with Vote Assignment

As in the flat protocol, we must show here that the hierarchical ring protocol does not allow two conflicting operations to occur at the same

time. We do that by proving the following lemmas.

Lemma 4 *In a L-level ring hierarchy, the hierarchical ring protocol guarantees that there is a non-empty intersection at level 0 between any read and write quorums.*

Proof. By induction on the levels of the ring hierarchy.
Basis. We have to show that the lemma holds for the highest level of the ring hierarchy. So, we also have to prove the following lemma:

> **Lemma 5** *In a L-level ring hierarchy, the hierarchical ring protocol guarantees that at level L there is a non-empty intersection between any read and write quorums.*
> *Proof.* L is the highest level in the ring hierarchy and has a single ring composed of m_L elements. Since quorums in the hierarchical protocol are constructed in similar way to the flat protocol, the proof follows from Lema 1. □

Hypothesis. There is a non-empty intersection at a level i ($1 \leq i \leq L$) of the ring hierarchy between any read and write quorums.
Step. We must show that if there is a non-empty intersection between the read and write quorums at level i, there must also be a non-empty intersection at level $i-1$.
If there is a non-empty intersection at level i, there must be at least one element at level $i-1$ that belongs to both the read and the write quorums. That element may be either a copy at level 0, which completes the proof, or a ring, composed by m_{i-1} elements at level $i-2$, where read and write quorum was formed. If the element is a ring, by Lemma 1, there must be a non-empty intersection between the quorums at level $i-2$ and, therefore, there must be a non-empty intersection between them at level $i-1$. Hence, the lemma follows. □

Lemma 6 *In a L-level ring hierarchy, the hierarchical ring protocol guarantees that there is a non-empty intersection at level 0 between any pair of write quorums.*

Proof. The proof is analogous to the proof of Lemma 4. □

We now show that the *coterie* generated by the hierarchical ring protocol cannot be generated by any vote assignment in the voting protocol [12].

Lemma 7 *There is no vote assignment equivalent to the hierarchical ring protocol.*

Proof. By contradiction. Consider the ring shown in figure 6. Let $v_1, \ldots, v_{15}$ be the votes assigned to the 15 copies and V_t the total number of votes. Consider the two eligible write quorum sets of copies {1,2,7,8,10,11} and {4,5,10,12,13,14}, and two other sets that are not eligible quorums {1,2,7,8,10,13} and {4,5,10,11,12,14}. In a way analogous to the proof of Lemma 3, we conclude that $v_{13} < v_{11}$ and $v_{11} < v_{13}$, which is a contradiction. Hence, the lemma follows. □

4 Availability

In this section we analyse the probability of forming a quorum, in both the flat protocol and in the hierarchical protocol, considering the reliability of the nodes.

In a failure free environment, every quorum requested by a node needing to access the replicated data item is successfully formed (providing that other nodes are not taking part in some other write operation). When failures are allowed to happen in the system, there may occur situations where a node cannot form the quorum required to the operation it needs to perform. Such situations must be avoided by the control protocol.

In the flat protocol, a read quorum can be formed as long as there are at least two accessible nodes which are adjacent in the ring. The write quorum can be formed while there are no two inaccessible adjacent nodes in the ring. Therefore, in the worst case, the flat protocol will tolerate failures of $\lceil \frac{n}{2} \rceil - 1$ nodes (n is the total number of copies) for read quorum construction, and of a single node for write quorum construction. In the

best case, it will tolerate failures of $n-2$ nodes for read quorum, and of $\lceil \frac{n}{2} \rceil - 1$ nodes for write quorum. In a similar manner, the hierarchical protocol, in the worst case, will tolerate failures of $\prod_{i=1}^{i=L} \lceil \frac{m_i}{2} \rceil - 1$ nodes for read quorum construction, and of $2^L - 1$ nodes for write quorum construction. In the best case, it will tolerate failures of $n - 2^L$ nodes for read quorum, and of $\prod_{i=1}^{i=L} \lceil \frac{m_i}{2} \rceil - 1$ nodes for write quorum construction.

Availability can be analysed by the likelyhood that a read or write quorum is successfully formed, considering the reliability of the nodes. We assume that a node is independently accessible with probability ρ ($0 < \rho < 1$), and take ρ as a measure of the reliability of the nodes. We first derive expressions for the availability in the flat protocol of read and write operarions. The availability of the operations in the hierarchical protocol are then expressed as recurrences based on the availability in the flat protocol.

To derive the availability of the read and write operations in the flat protocol we need to calculate the probability that some specific situations happen in the ring structure. These probabilities of interest, expressed recursively, are:

- Probability that two accessible (good) adjacent nodes exist in a ring of size $n+1$ when it is known that one node is inaccessible (bad):

$$P_b^{gg}(n,\rho) = \begin{cases} 0 & (n=0) \\ 0 & (n=1) \\ (1-\rho)P_b^{gg}(n-1,\rho) \;+\; \rho^2 & \\ +\;\; \rho(1-\rho)P_b^{gg}(n-2,\rho) & (n \geq 2) \end{cases}$$

- Probability that two accessible adjacent nodes exist in a ring of size $n+1$ when it is known that one node is accessible:

$$P_g^{gg}(n,\rho) = \begin{cases} \rho & (n=1) \\ \rho \;\; + \;\; \rho(1-\rho) & \\ +\;\; (1-\rho)^2 P_b^{gg}(n-2,\rho) & (n \geq 2) \end{cases}$$

- Probability that two inaccessible adjacent nodes exist in a ring of size $n+1$ when it is known that two specific adjacent nodes are accessible and inaccessible, respectively:

$$P_{gb}^{bb}(n,\rho) = \begin{cases} 0 & (n=1) \\ 1-\rho & (n=2) \\ \rho P_{gb}^{bb}(n-1,\rho) \ + \ (1-\rho)^2 & \\ +\ \rho(1-\rho)P_{gb}^{bb}(n-2,\rho) & (n \geq 3) \end{cases}$$

- Probability that two accessible adjacent nodes and no two inaccessible adjacent nodes exist in a ring of size $n+1$ when it is known that one node is inaccessible:

$$P_{b}^{gg\wedge\neg bb}(n,\rho) = \begin{cases} 0 & (n=0) \\ 0 & (n=1) \\ \rho(1-\rho)P_{b}^{gg\wedge\neg bb}(n-2,\rho) & \\ +\ \rho^2(1-P_{gb}^{bb}(n-1,\rho)) & (n \geq 2) \end{cases}$$

- Probability that two accessible adjacent nodes and no two inaccessible adjacent nodes exist in a ring of size $n+1$ when it is known that one node is accessible:

$$P_{g}^{gg\wedge\neg bb}(n,\rho) = \begin{cases} \rho & (n=1) \\ \rho(1-P_{b}^{gg}(n-1,1-\rho)) & \\ +\ \rho(1-\rho)(1-P_{gb}^{bb}(n-1,\rho)) & \\ +\ (1-\rho)^2 P_{b}^{gg\wedge\neg bb}(n-2,\rho) & (n \geq 2) \end{cases}$$

With these probabilities it is straightforward to calculate the availability in the flat protocol. The availability of read operations AF_r (the probability that two accessible adjacent nodes exist) in a ring of size n is given by the following expression:

$$AF_r(n,\rho) = \rho P_{g}^{gg}(n-1,\rho) \ + \ (1-\rho)P_{b}^{gg}(n-1,\rho).$$

To form a write quorum in the flat protocol there must not be any two inaccessible adjacent nodes in the ring (in order to get half of the alternating nodes) and there must be two accessible adjacent nodes (to complete the majority). So, the availability of write operations AF_w in a ring of size n is given by:

$$AF_w(n,\rho) = \rho P_g^{gg\wedge\neg bb}(n-1,\rho) \; + \; (1-\rho)P_b^{gg\wedge\neg bb}(n-1,\rho).$$

The availability in the hierarchical protocol must be calculated one level at a time. The availability of a level is expressed in terms of the availability of the level immediately below, and is calculated similarly to the flat protocol. Let L be the number of levels in the ring hierarchy, and m_i ($m_i \geq 2; 1 \leq i \leq L$) be the number of $i-1$ level elements that compose an element at level i. The availability of read operations AH_r in the hierarchical protocol is given by the following recurrence:

$$AH_r(i) = \begin{cases} \rho & (i=0) \\ AF_r(m_i, AH_r(i-1)) & (1 \leq i \leq L) \end{cases}$$

The availability of write operations is calculated in an analogous way, and is given by:

$$AH_w(i) = \begin{cases} \rho & (i=0) \\ AF_w(m_i, AH_w(i-1)) & (1 \leq i \leq L) \end{cases}$$

Availability of read operations in the flat protocol is much higher than the availability in the voting protocol (specially for small values of ρ), and approaches rapidly to 1 when n is large. Availability of write operations, however, is low in a comparison to the voting protocol, and decreases even further when the number of copies increases. This is due to the constraint imposed by the flat protocol that the write quorum must be formed with a majority of alternating copies in the ring,

The generalization into a ring hierarchy made in the hierarchical protocol allows a better relation between read and write availability. Keeping the number of elements per ring small increases the availability of

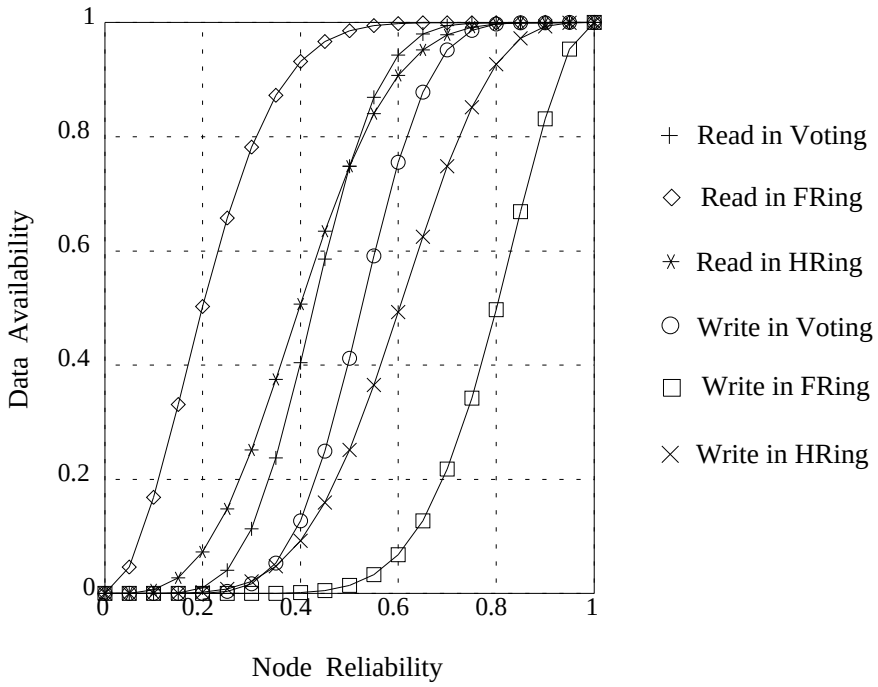

Figure 13: Data availability with twenty copies.

write operations, but decreases the availability of read operations (and vice versa). Depending on the acceptable size of the read and write quorums (which in turn depends on the number of elements per ring and on the number of levels in the ring hierarchy), it is possible to achieve an availability of write operations comparable to the voting protocol (specially for high values os ρ), with an availability of read operations even better. Figure 13 shows a comparative plot of availability in the flat protocol, the hierarchical protocol and the voting protocol. We considered an example with 20 copies and, in the hierarchical protocol, $L = 2$, $m_1 = 4$ and $m_2 = 5$.

5 Related Work

In this section we compare the hierarchical protocol to other existing protocols for maintaining the consistency of replicated data.

In the traditional voting mechanism [18], the write quorum must have at least the majority of the copies and the read quorum must be the complement of the write quorum plus any other copy. By those restrictions, reducing the read quorum implies increasing the write quorum. With a read quorum of a single copy, the voting mechanism falls into the extreme case of read-one write-all, where a read operation is made accessing only one copy, but write operations need access to all copies. A single node failure would be sufficient to make the replicated data item unavailable. In the hierarchical ring protocol, the write quorum is equal to or less than the majority of the copies (the latter case being the more frequent); the read quorum is small when compared to the total number of copies and much smaller than the complement of the write quorum.

By maintaining information about the system configuration, the virtual partitions protocol [1] achieves a one copy read quorum without increasing the write quorum. That protocol, however, requires additional procedures (rounds of messages) when failures occur in order to update the configuration information at each node. The hierarchical ring protocol needs a reduced read quorum (a minimum of just two copies is possible) without raising communications costs with reconfigurations procedures — the static hierarchical ring structure is sufficient to ensure copy consistency even in the presence of failures.

Other protocols that organize the copies in logical structures to maintain consistency are [9, 14, 4]. Those protocols achieve quorums smaller than the quorums in voting, and that do not grow linearly with the total number of copies. In the grid protocol [9], the copies are organized into a grid of dimension $\sqrt{n} \times \sqrt{n}$ (where n is the total number of copies). A read quorum is formed by getting access permission from at least one copy at every column, and the write quorum is formed by a read quorum plus permissions from every copy in at least one row. The read and write quorums, thus, are of size $\sqrt{n}$ and $2\sqrt{n} - 1$, respectively. The

hierarchical quorum consensus protocol [14] uses a ternary tree where the copies are at the leaves. A logical node in the tree votes favorably to the quorum construction if it receives a favorable vote from the majority of its children. Repeating this process up to the highest level of the tree, the quorum would be formed only if the root votes favorably. The read and write quorums are of the same size and equal to $n^{0.63}$. In the tree quorum protocol [4], the copies are organized into a logical tree structure, and the quorums are constructed by getting permissions from copies that are part of specific traversals on the tree. In the best case, the write quorum is proportional to $O(\log n)$, and the read quorum needs accessing only the copy corresponding to the root of the tree.

In both the grid and the hierarchical quorum consensus protocols, the read quorum is small but grows, although not linearly, when the total number of copies increases. [15] proposed a hierarchical grid protocol that improves data availability but keeps the same quorum sizes of the grid protocol. In the hierarchical ring protocol, the read quorum is small and, if the number of levels in the ring hierarchy is not changed, remains constant when the total number of copies increases. With $m_i = 3 (1 \leq i \leq L)$, the hierarchical ring protocol achieves the same results as the hierarchical quorum consensus does — including quorum sizes and availability.

Assuming a critical point of view, the tree quorum protocol should not be considered fully distributed, since the roles played by the copies are not symmetric (the same criticism can also be made to the weighted voting protocol [12] and its variants and extensions). In the tree quorum protocol a read quorum of just one copy can only be formed if permission from the root copy is obtained. By contrast, the (few) copies needed to form the read quorum in the hierarchical ring protocol may be any copy of the replicated data item.

Multidimensional voting [5] is another scheme for generating *coteries* in distributed systems and, in particular, for replicated data control. It consists in a generalization of the weighted voting originally proposed in [12]. To each copy of the replicated data item it is assigned a vector of dimension k (k positions), and to each vector position it is assigned

an independent number of votes. The quorums are formed by obtaining the majority of the votes in l predetermined vector positions. In [15] it is shown that any *coterie* has an equivalent multidimensional voting assignment. So, there exists also a multidimensional voting assignment equivalent to the *coterie* generated by the ring protocols. In multidimensional voting, however, nodes are not aware of their positions in the equivalent logical structure, and, therefore, cannot take advantage of this information to further reduce the quorum sizes.

6 Conclusion

We proposed a new protocol for replicated data control in which the read quorum is small (and in many situations may be constant) without increasing the write quorum. First we presented the flat ring protocol, where copies of a replicated data item are organized in a single ring structure. We then generalized the flat ring protocol and described the hierarchical ring protocol, where copies are organized in a multi-level ring hierarchy. Both protocols use the adjacency information of the ring structure to maintain copy consistency and to achieve smaller read quorums. The availability provided by the hierarchical ring protocol is comparable to the availability in voting for write operations, and much better for read operations. The protocols are fault-tolerant and do not need any additional reconfiguration procedure when failures occur. Considering systems where the number of read operations is much larger than the number of write operations, the hierarchical ring protocol has remarkable advantages.

Acknowledgments

The authors wish to thank Cláudio L. Lucchesi for his help in calculating the probabilities involved in the analysis on data availability.

References

[1] A. El Abbadi and S. Toueg. Maintaining Availability in Partioned Replicated Databases. *ACM Transactions on Database Systems*, pages 264–290, June 1989.

[2] D. Agrawal and A. El Abbadi. Exploiting Logical Structures of Replicated Databases. *Iformation Processing Letters*, 33(5):255–260, January 1990.

[3] D. Agrawal and A. El Abbadi. Storage Efficient Replicated Databases. Technical Report TRCS90-5, University of California, Department of Computer Science - Santa Barbara, 1990.

[4] D. Agrawal and A. El Abbadi. The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. Technical Report TRCS90-5, University of California, Department of Computer Science - Santa Barbara, 1990.

[5] Mustaque Ahamad, Mostafa H. Ammar, and Shun Yan Cheung. Multidimensional Voting. *ACM Transactions on Computer Systems*, 9(4):399–431, November 1991.

[6] Daniel Barbara and Hector Garcia-Molina. The Reliability of Voting Mechanisms. *IEEE Transactions on Computers*, C-36(10):1197–1208, October 1987.

[7] Daniel Barbara, Hector Garcia-Molina, and Annemarie Spauster. Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment. *ACM Transactions on Computer Systems*, pages 394–426, November 1989.

[8] Philip A. Bernstein and Nathan Goodman. The Failure and Recovery Problem for Replicated Databases. *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 114–122, August 1983.

[9] S. Y. Cheung, M. Ammar, and M. Ahamad. The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data. *Proceedings of the 6th IEEE International Conference on Data Engineering*, pages 438–445, 1990.

[10] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in Partioned Networks. *ACM Computing Surveys*, pages 341–370, September 1985.

[11] Hector Garcia-Molina and Daniel Barbara. How to Assign Votes in a Distributed System. *Journal of the ACM*, 32(4):841–860, October 1985.

[12] D. K. Gifford. Weighted Voting for Replicated Data. *Proceedings of the 7th Symposium on Operating Systems Principles*, pages 150–162, December 1979.

[13] Sushil Jajodia and David Mutchler. Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. *ACM Transactions on Database Systems*, pages 230–280, June 1990.

[14] Akhil Kumar. Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data. *IEEE Transactions on Computers*, 40(9):996–1004, September 1991.

[15] Akhil Kumar and Shun Yan Cheung. A High Availability $\sqrt{N}$ Hierarchical Grid Algorithm for Replicated Data. *Iformation Processing Letters*, 40(6):311–316, December 1991.

[16] Jehan-François Pâris. Voting with Witnesses: A Consistency Scheme for Replicated Files. *Proceedings of the 6th IEEE Conference on Distributed Computing Systems*, pages 606–612, June 1986.

[17] Jin Tang and N. Natarajan. A Scheme for Maintaining Consistency of Replicated Files in Partioned Distributed Systems. *Proceedings of the 5th IEEE International Conference on Data Engineering*, pages 530–537, 1989.

[18] Robert H. Thomas. Majority Concensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, pages 180–209, June 1979.

Relatórios Técnicos

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br