Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data

Article *in* IEEE Transactions on Computers · October 1991

DOI: 10.1109/12.83661 · Source: IEEE Xplore

1 author:

Akhil Kumar
Pennsylvania State University
148 PUBLICATIONS **4,384** CITATIONS

SEE PROFILE

Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data

Akhil Kumar

Abstract—**This paper describes a new algorithm for managing replicated data. The standard quorum consensus method is an expensive means of maintaining consistency across multiple copies of an object if the number of copies is large because it typically requires at least a majority of the copies to form a quorum for a write operation. Our method is based on organizing the copies of an object into a logical, multilevel hierarchy, and extending the quorum consensus algorithm to such an environment. Several properties of the method are derived and optimality conditions are given for minimizing the quorum size. It is shown that, given a collection of n copies of an object, our method allows a quorum to be formed with $n^{0.63}$ copies versus $\lceil \frac{n+1}{2} \rceil$ copies in the case of the majority voting algorithm. Tradeoffs between ours and three other quorum-based methods are discussed, and the main features of each method are highlighted.**

Index Terms—**Availability, databases, distributed systems, quorum consensus, replication, synchronization, voting.**

I. Introduction

REPLICATION of data increases system reliability in order to satisfy the high availability requirements in various applications. Clearly, if multiple, identical copies of a data object reside on several computers with independent failure modes, then the system would be more reliable. The disadvantage, however, is that the read and write operations performed on a replicated object by various concurrent transactions must be synchronized. For instance, two write operations from independent transactions must not be allowed to simultaneously update different copies of the object. In order to achieve this synchronization between multiple copies, possibly residing at different sites, additional communications and processing cost is incurred as compared to the case in which only one copy of an object exists.

The majority voting method [17] and the quorum consensus method [10] are well-known algorithms for synchronizing operations on multiple copies of a data object. Both are based on assigning votes to copies, and defining a read and a write quorum. A read operation must assemble enough votes to form a read quorum, while a write must assemble a write quorum. The quorums are chosen in such a way that any pair of one read and one write, or a pair of two write quorums intersects, i.e., there is at least one copy in common between the two quorums in the pair. In majority voting, for instance, each copy is assigned a single vote, and a majority of votes must be assembled to perform any operation. In the quorum consensus method, the votes assigned to various copies and the read and write quorum sizes are allowed to vary. Subsequently, researchers have proposed many variants of this basic algorithm such as: dynamic voting [6], [12], [16], voting with witnesses [16], the missing writes algorithm [7], and the views approach [8]. Protocols for dynamically changing votes and quorum sizes are discussed in [11] and [3].

A major problem with the quorum consensus method is that the number of copies required for a quorum increases linearly with the total number of copies present of an object. Consequently, to perform a write operation on 100 copies of an object would require a quorum of at least 51 copies. Although one cannot conceive of a database application where synchronization among such a large number of copies is required at present, such a need is likely to grow in the future with the proliferation of distributed systems.

Theoretically, one could reduce the size of a quorum by assigning unequal votes to the various copies of an object as suggested in [13]. However, this optimization does not scale very well and will not be helpful if the number of copies is very large, say 50 or more, because it will exhibit drawbacks of a centralized system, such as excessive reliance on one or a few copies.

Our objective is to develop a synchronization scheme that scales well even for a large number of copies of an object. Moreover, it should result in high availability and minimize the size of a quorum. Consequently, we pose the following problem: given n *identical* copies of an object, what is the minimum number of copies required to synchronize both read and write operations in a fully distributed algorithm where each copy assumes equal responsibility for synchronization? (The majority voting method requires $\lceil \frac{n+1}{2} \rceil$ copies, but can one do better?)

This paper generalizes the quorum consensus scheme into a multilevel algorithm called **hierarchical quorum consensus**, and shows that given a collection of n copies of an object, the minimum size of a quorum is $n^{0.63}$ copies. A smaller quorum size results in a lower cost of synchronization.

To illustrate the basic idea behind our algorithm, consider a group of nine copies of an object organized into three subgroups, each of size three (see Fig. 1). Our algorithm allows an object to assemble a quorum consisting of two copies each from any two subgroups. The total size of a quorum is thus reduced to four, instead of five as would be the case in the majority voting algorithm. It is easy to see that all quorums produced in this way will intersect each other, and therefore the algorithm is correct. Moreover, an identical

Manuscript received September 7, 1989; revised November 9, 1990.

The author is with the Graduate School of Management, Cornell University, Ithaca, NY 14853.

IEEE Log Number 9101686.

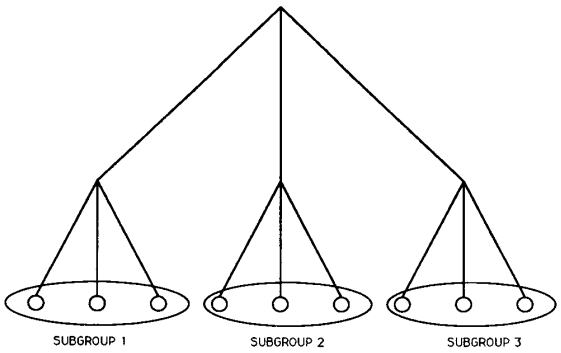


Fig. 1. An example of nine copies organized into three subgroups.

collection of intersecting quorums cannot be produced by any vote assignment within the framework of the quorum consensus method. On the other hand, if three copies from subgroup 1 and one each from subgroups 2 and 3 are available in Fig. 1, then the majority voting method is able to form a quorum, while our method is not. This tradeoff is also studied later in this paper.

The organization of this paper is as follows. Sections II and III discuss the basic principle behind the algorithm, along with a proof of correctness and the implementation details. Section IV shows that our method produces certain collections of intersecting quorums that cannot be produced in the quorum consensus method. It is also shown how a given number of copies can be organized into an optimal, multilevel hierarchy so as to minimize the quorum size. In Section V, an availability comparison against majority voting is performed. Finally, Section VI discusses tradeoffs between ours and three other quorum-based methods.

II. Basic Concepts

We first describe the quorum consensus algorithm, and then discuss the principle behind our algorithm and show its correctness.

Quorum Consensus algorithm [10]: This algorithm describes how access by read and write operations to n copies of a replicated object can be synchronized. A read operation must lock q_r copies, and a write must lock q_w copies, such that

$$q_r + q_w > n, \quad \text{and}$$
$$2q_w > n.$$

These two constraints are called the quorum intersection conditions. Basically, a write operation must assemble a quorum of q_w copies, while a read must gather at least q_r copies, and in each case identify the copy with the highest version number. (A version number is maintained with each copy, and is incremented every time an update takes place.) Under these conditions, the read and the write quorums are said to intersect, i.e., every pair consisting of one read and one write, or two write quorums has at least one copy in common between the two quorums. Moreover, any set of q_r copies will always contain at least one copy with the most recent version.

Hierarchical Quorum Consensus: Our algorithm, called hierarchical quorum consensus, is based on logically organizing a set of copies of an object in a database into a multilevel tree (of depth m) with the root as level 0 (see Fig. 2). The physical copies of an object are stored only in the leaves of this tree or at level m, while the higher level nodes of the tree correspond to logical groups. A node at level i, where i varies from 0 to $m-1$, is viewed as a logical group which in turn consists of l_{i+1} subgroups at level $i+1$. For example, Fig. 1 is a two-level hierarchy where the root has 3 level 1 subgroups, each in turn containing three physical copies. A quorum is associated with each level and to access a logical group at a certain level, a quorum consisting of its subgroups must first be assembled. We first define the concept of a quorum and then derive the conditions for a correct replica control protocol in this context.

Definition 1: A read (write) quorum at level i is defined as the number of subgroups of a level $i-1$ group that must be locked by a read (write) operation to obtain read (write) access to the group. The read (write) quorum at level i is denoted by r_i (w_i).

Notice that this is a recursive definition. Therefore, each level i group must in turn assemble r_{i+1} of its subgroups at level $i+1$, and so on. This would eventually translate into a quorum consisting of physical copies of the object. Now, we define the condition for the replica control protocol to be correct.

Definition 2: A replica control scheme is correct if all possible pairs of a) read and write quorums, and b) write quorums intersect (i.e., have at least one physical copy in common between the two quorums in the pair).

Theorem 1: A replica control scheme which requires a read operation to (recursively) assemble a quorum of r_1 level 1 subgroups, and a write operation to (recursively) assemble a quorum of w_1 level 1 subgroups is correct if the read quorum r_i and the write quorum w_i are such that

a) $r_i + w_i > l_i$, and
b) $2w_i > l_i$.

for all levels $i, i = 1, \cdots, m$.

Proof: (by induction) The above theorem is proved by starting at the root of the tree (level 0) and showing that the above conditions force intersection between physical copies for any two write quorums or a read and a write quorum.

(Base step) We first show that a read and a write quorum will intersect at level 1. Clearly, since $r_1 + w_1 > l_1$ (from the hypothesis), any pair of a read and a write quorum will have at least one subgroup at level 1 in common, i.e., they must intersect. Moreover, since $2w_1 > l_1$, any pair of write quorums at level 1 must also intersect.

(Inductive step) Next, we show that, in general, if two quorums intersect at level i, then with an appropriate choice of r_{i+1} and w_{i+1}, they must also intersect at level $i+1$. Recall that our definition for a quorum is a recursive one; therefore, a read (write) quorum at level i, involves the formation of r_i (w_i) read (write) quorums at level $i+1$, and so on. Each subgroup that participates in a read (write) quorum at level i must in turn assemble $r_{i+1}(w_{i+1})$ of its

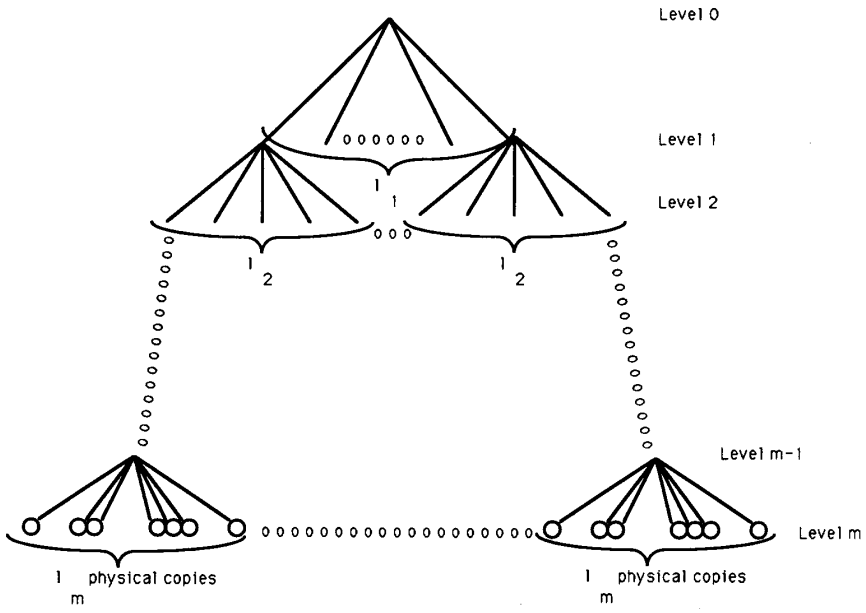


Fig. 2. An m-level hierarchy.

subgroups at level $i+1$ to form a read (write) quorum at that level. Now, since $r_{i+1} + w_{i+1} > l_{i+1}$ and $2w_{i+1} > l_{i+1}$ (for $i = 0, \cdots, m-1$), a read and a write, or two write quorums that have a common subgroup at level i, must also have at least one subgroup at level $i+1$ in common between them. Therefore, quorums that intersect at level i also intersect at level $i+1$.

By induction on i, it follows that any pair of read and write, or two write quorums that intersect at level 1 will in turn force intersection between subgroups at level i provided $r_k + w_k > l_k$ and $2w_k > l_k$, for all values of k from 1 through i. Since $i = m$ corresponds to the level of physical copies, it follows that the two quorums must have at least one physical copy in common.

Corollary 1: The number of physical copies in a read quorum is $r_1 r_2 \cdots r_m$, and the number of copies in a write quorum is $w_1 w_2 \cdots w_m$, where r_i (w_i) is the size of the read (write) quorum at level i.

Proof: (for read quorum) The root must assemble r_1 level 1 groups. Each selected level 1 group must in turn assemble r_2 level 2 subgroups, and so on. Therefore, the actual number of physical copies assembled in a read quorum is $r_1 r_2 \cdots r_m$. (The proof for the size of the write quorum is identical.) □

III. HQC Algorithm

This section describes the HQC algorithm in detail, and later gives an example. As mentioned above, the physical copies are organized into an m-level hierarchy (with the root as level 0) such that a node at level $i-1$ corresponds to a group consisting of l_i subgroups. Physical copies are present only at the leaf level (level m) of this hierarchy.

Each physical copy is assigned an m-subscript label (or identifier) of the form $o_{i_1 i_2 \cdots i_m}$ where i_j varies between 1 and l_j, for j varying from 1 to m. For instance, in a two-level hierarchy of 9 copies such as Fig. 1, with l_1 and l_2 each 3, the copies are labeled as o_{11}, o_{12}, o_{13}, o_{21}, o_{22}, o_{23}, o_{31}, o_{32} and o_{33}. Similarly, in a three-level hierarchy of 27 copies, the copies are labeled as o_{ijk}, where i, j, and k vary from 1 to 3.

The basic idea behind the algorithm is that a read operation must (recursively) assemble a quorum of r_1 level 1 groups (i.e., each selected level 1 group must assemble r_2 level 2 subgroups, and so on). Similarly, a write operation must (recursively) assemble a quorum of w_1 level 1 groups. For example, in the two-level hierarchy of Fig. 1, assume that r_1, r_2, w_1, and w_2 are each set to 2. Clearly four copies are required for read or write synchronization, and one possible quorum is o_{12}, o_{13}, o_{31} and o_{32}. For another example, consider the three-level hierarchy mentioned above, and assume that each of r_1, r_2, r_3, w_1, w_2 and w_3 is 2. In this case, eight copies are required for synchronization and a possible quorum is o_{122}, o_{123}, o_{131}, o_{132}, o_{311}, o_{312}, o_{322} and o_{323}.

The complete listing of the algorithm is given in pseudo-code below and a brief description follows.

```
Algorithm 1
{
if ((assemble(1, q[1]) == 1)
    return(yes) /*request successful */
```

```
    else{
    unlock all copies locked;
    return(no); /*request unsuccessful */
      }
}

assemble(l_no, quorum)
{
num_examined = 0;
num_locked = 0;
tot_num = children[l_no];
while (num_locked < quorum) and (num_examined < tot_num){
  if (l_no < m)
    num_locked = num_locked + assemble (l_no + 1, q[l_no + 1])
        /*recursive call to assemble */
  else       /*at level m, the lowest level */
    if (copy o[i[1], i[2], . . . ,i[m]] is unlocked) {
        lock copy o[i[1], i[2], . . . i[m]];
        num_locked ++;
                          }
  num_examined ++;
              } /* end of while loop */
if (num_locked < quorum)
    return(0);
  else
    return(1);
}
```

The array element *children* [*l_no*], also referred to as *tot_num*, represents the number of children or subgroups in a level $l\_no - 1$ group, while $q[l\_no]$ is the number of those subgroups required to form a quorum. The variable *num_examined* refers to the number of subgroups examined at a certain level, and *num_locked* to the number of subgroups actually locked at that level. Finally, physical copies are identified by an m-subscript label as described above.

The main section of the algorithm makes a call to the recursive function **assemble** to form a quorum of size $q[1]$ at level 1. Function **assemble** returns 1 if it is successful in assembling the desired quorum at any level; else, it returns 0. This function calls itself recursively with two parameters: level number, *l_no* and the required quorum size at that level, $q[l\_no]$. Since a successful call to the function returns a 1, it results in the variable *num_locked* being incremented by 1. At level m, the lowest level, locks are obtained on physical copies. At this level, before locking a copy, a check is first made to ensure that it is not already locked on behalf of another operation.

For an example, consider Fig. 3. It shows a collection of 27 copies of an object organized into a three-level hierarchy where l_1, l_2, and l_3 are each 3. Table I shows the various possible combinations of read and write quorum values at each level, and also the total number of copies required for read and write synchronization. Notice that our method allows a write quorum to be formed with as few as eight copies, while the majority consensus method would require 14.

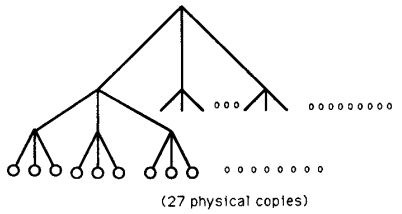


Fig. 3. Organizing 27 copies into a three-level hierarchy.

IV. Properties

In this section, two interesting and useful properties of the hierarchical quorum consensus (HQC) method are derived. The first shows that all quorums based on the HQC algorithm do not have an equivalent single-level representation. The second property shows that the minimum size of a quorum for synchronizing operations on n copies is $n^{0.63}$. An algorithm for organizing the copies into an m-level hierarchy is also given.

A. Hierarchical Versus Single-Level

Theorem 2: All quorums based on hierarchical quorum consensus do not have an equivalent single-level vote assignment.

Proof: Consider a collection of nine copies organized into a two-level hierarchy such that the root has three subgroups at level 1, and each subgroup in turn contains three physical copies at level 2 (see Fig. 1). Further, assume that

TABLE I
POSSIBLE READ AND WRITE QUORUMS IN A THREE-LEVEL HIERARCHY OF 27 COPIES

No.	r_1	w_1	r_2	w_2	r_3	w_3	r	w
1.	1	3	1	3	1	3	1	27
2.	1	3	1	3	2	2	2	18
3.	1	3	2	2	2	2	4	12
4.	2	2	2	2	2	2	8	8

r_1, w_1, r_2 and w_2 are each 2. We now show, by contradiction, that a single level assignment of votes that would produce a set of quorums identical to those of the two-level scheme does not exist.

In trying to find an equivalent single-level vote assignment, the nine copies are treated as one group, all at the same level. Let us assume that the copies in subgroup 1 are assigned votes v_{11}, v_{12} and v_{13}, while those in subgroups 2 and 3 are assigned votes v_{2j} and v_{3j}, respectively, for all values of j from 1 to 3. If such an assignment of votes does exist and produces the same set of quorums as the two-level scheme, then the following four expressions must be true (where s is the sum of all votes):

$$v_{11} + v_{12} + v_{21} + v_{22} > s/2 \quad (1)$$

$$v_{22} + v_{23} + v_{31} + v_{32} > s/2 \quad (2)$$

$$v_{11} + v_{13} + v_{21} + v_{23} > s/2 \quad (3)$$

$$v_{12} + v_{13} + v_{31} + v_{32} > s/2. \quad (4)$$

Furthermore, since the sum of all votes is s, it means that

$$v_{11} + v_{12} + v_{13} + v_{21} + v_{22} + v_{23} + v_{31} + v_{32} + v_{33} = s. \quad (5)$$

Now from expressions (1), (2), and (5),

$$v_{22} > v_{13} + v_{33} \quad (6)$$

Since all votes are positive, it obviously follows that

$$v_{22} > v_{13}. \quad (7)$$

By a similar reasoning, from (3), (4), and (5), one may argue that

$$v_{13} > v_{22}. \quad (8)$$

This is a contradiction. Hence, the given two-level scheme does not have an equivalent single-level vote assignment. □

[5] describes a replica control scheme based on multidimensional voting. In this method, a vector of votes is associated with each copy, and a corresponding quorum vector is also defined. Therefore, a quorum can independently be formed on several dimensions. For a quorum to be complete, it is necessary to assemble individual quorums on at least a stipulated minimum number of dimensions. This is similar to the concept of coteries [9] and produces an even larger set of intersecting quorums than the HQC method. However, all copies of an object do not assume equal responsibility for synchronization because the votes associated with them are not equal.

B. Minimum Quorum Size

In previous sections, an arbitrary scheme was assumed for subdividing n copies of an object into a hierarchy. Now we turn to describe an algorithm for constructing a hierarchy such that the size of a write quorum in terms of the total number of physical copies is minimized. In the following discussion, note that since any two write quorums must intersect at all levels, clearly, the size of a write quorum at any level is minimum when it is equal to a majority of the total number of subgroups within a group at that level. Also, the read quorum need not be any larger than the write quorum at any level in order to satisfy the quorum intersection conditions. Our algorithm is described first, and a proof of optimality follows.

Algorithm 2: Repeatedly divide n by 3 and rewrite it in the form $n' = 3^k \times 5^b$ where b is either 0 or 1, and n' is the least upper bound of n.

Corollary 2: The number of levels in the minimum quorum size solution is k if b is 0, and $k+1$ if b is 1. Moreover, the size of a read or a write quorum is 2^k if b is 0 and 3×2^k if b is 1.

Proof: It follows directly from the algorithm. □

Definition 3: In a single group of n copies of an object, the size of a majority quorum is defined by the function *maj*(n) as follows:

$$\text{maj}(n) = n/2 + 1, \quad \text{for even } n$$

$$\text{maj}(n) = (n+1)/2, \quad \text{for odd } n.$$

Lemma 1: If a group of n copies is organized into a two-level hierarchy with n_1 subgroups at level 1, and n_2 level 2 subgroups of each level 1 group (i.e., $n_1 \times n_2 = n$), this will result in a quorum of the same (or smaller) size as compared to the one-level case if

$$\text{maj}(n) \geq \text{maj}(n1) \times \text{maj}(n2).$$

Proof: The left-hand side represents the majority quorum size in a single group of n copies. On the other hand, the right-hand side is the product of the individual majority quorums at levels 1 and 2, and represents the quorum size in the two-level case. Therefore, the above inequality is simply the condition for the quorum in the one-level case to be greater than (or equal to) the quorum in the two-level case. □

Lemma 2: If a group of n copies ($n > 5$) is divided into a two-level hierarchy such that there are $n_1 = 3$ subgroups at level 1, and each level 1 group in turn contains $n_2 = \lceil n/3 \rceil$ level 2 subgroups, the total quorum size will remain the same or be smaller than the quorum size in the single-level case.

Proof: There are two parts in the proof. First, from Lemma 1, the general condition for the above rule to hold is

$$\lceil n/2 \rceil + 1 \geq 2(\lceil \lceil n/3 \rceil / 2 \rceil + 1).$$

This expression is rewritten as

$$\min(\lceil n/2 \rceil) \geq \max(2\lceil \lceil n/3 \rceil / 2 \rceil) + 1.$$

This can further by reexpressed as (without the min, max and ceiling signs)

$$n/2 \geq 2((n+2)/3) + 1)/2 + 1.$$

The above expression has a solution: $n \geq 16$. Therefore, any number larger than 15 can be rewritten as $3 \times n_2$, without causing an increase in the quorum size.

Second, one may verify by inspection that this lemma is also true for all values of n from 6 through 15.

Hence, the lemma is proved. □

Theorem 3: Algorithm 2 minimizes the quorum size required for a write operation.

Proof: The basic idea behind the algorithm is to reduce a number n into smaller factors in order to realize savings in the number of copies required to form a quorum. The proof is by contradiction. Assume that n can be factored into a form: $n_1 \times n_2 \times \cdots \times n_k$, and that this leads to the smallest quorum. Further assume that this form is different from our standard form: $3^k 5^b$ where k is a nonnegative number and b is 0 or 1. We now show that one can always transform this form into our standard form without incurring any increase in the size of the quorum.

If the factors of n do not conform to the standard form it means that

1) either, a factor is different from 3 and 5,
2) or there is more than one occurrence of 5.

As shown in Lemma 2 above, any number $n > 5$ can be rewritten in the standard form by repeated subdivision by 3 without increasing the quorum size. Moreover, it is evident that a factor of 2 can be replaced by 3 (and a factor of 4 by 5), again without increasing the quorum size.

Now, turning to the second case where more than one factor is 5, note that an occurrence of 5^2 may be rewritten as 3^3 thereby reducing the quorum size; thus, all multiple occurrences of 5 can be replaced by 0 or 1 occurrences.

Hence, all other forms can be transformed into our form without increasing the quorum size, and this completes the proof. □

An intuitive explanation for this result is as follows. Majority voting is a special case of the hierarchical quorum consensus method with all the copies at one level. Given n copies of an object, the hierarchical quorum consensus method aims to reduce the total size of the quorum by organizing the copies into a multilevel, logical structure. Therefore, one might expect that a larger number of levels would result in a smaller quorum. To maximize the number of levels, one may organize a hierarchy in which each group contains two subgroups. However, in this case both the subgroups must participate in a quorum at any level (since $maj(2) = 2$), and hence, all the copies at the physical level would be required to form a quorum. This is clearly a poor solution. The next-best option for maximizing the number of levels is to have three subgroups in each group, and, now only two of the three subgroups would be required to participate in a quorum at any level (since $maj(3) = 2$). Hence, this alternative produces the smallest quorum size.

C. Quorum Size Comparison

Having determined the optimal number of levels above, we now turn to compare our method against majority consensus in terms of quorum size. As discussed above, for a collection of n copies, the quorum size is minimized by organizing them into a hierarchy where each group contains three subgroups. The number of levels in this hierarchy is $\log_3 n$, and the size of a quorum at each level is 2. Consequently, the total quorum size is $2^{\log_3 n}$ copies. By simple algebraic manipulation[1] this is rewritten as $n^{0.63}$. Contrast this with the majority consensus method where the quorum size is $\lceil (n+1)/2 \rceil$. Therefore, it is straightforward to show that the quorum is smaller in our method for all values of n greater than 8. It is also evident that in our method the quorum size will grow at a much slower rate with respect to n than in the majority voting method.

V. Availability Analysis

In this section, an availability comparison between our method and majority voting is carried out.

The availability of a replicated object is defined as the probability that both a read and a write quorum can be formed assuming that each copy of the object is independently up with probability p. This is naturally an important parameter of interest because it reflects the fraction of time during which a replicated object is accessible for read and write operations.

We shall compare the HQC method against majority voting in terms of availability. As mentioned earlier, majority voting is a special case of the quorum consensus method with all copies assigned single votes and the size of the quorum set equal to a majority of the total number of votes. In general, if there are n copies of an object and a quorum of q copies is required, the availability is given by the following expression:

$$A_{\text{maj}} = \sum_{i=q}^{i=n} \binom{n}{i} p^i (1-p)^{n-i}.$$

Since a minimum of q copies are necessary for a quorum, the above expression is just the sum of the individual probabilities of exactly i copies being up, where i varies from q to n. It is easy to see that setting q equal to $\lceil \frac{n+1}{2} \rceil$ as in majority voting maximizes the probability of forming both read and write quorums [2].

The availability of the HQC method is computed by working upwards from the lowest level of the hierarchy one level at a time. The availability of a subgroup at level i is denoted by A_i, and A_0 is the availability at the root level or the overall availability. The availability of a group at level $m-1$ is the probability that a quorum of size q_m can be formed from l_m subgroups, where each is independently up with probability p. Therefore, in general the availability of a group at level i is expressed in terms of the availability of its subgroups at level $i+1$. Consequently, the overall availability A_0 is computed from the following recurrence.

$$A_{\text{HQC}} = A_0$$

$$A_i = \sum_{j=q_i}^{j=l_i} \binom{l_i}{j} A_{i+1}^j (1 - A_{i+1})^{l_i - j}$$

$$A_m = p.$$

[1] Say $2^{\log_3 n} = x$. Taking log to the base 3 on each side, $\log_3 n \cdot \log_3 2 = \log_3 x$. On simplification, $\log_3 x = \log_3 n^{0.63}$, or $x = n^{0.63}$.

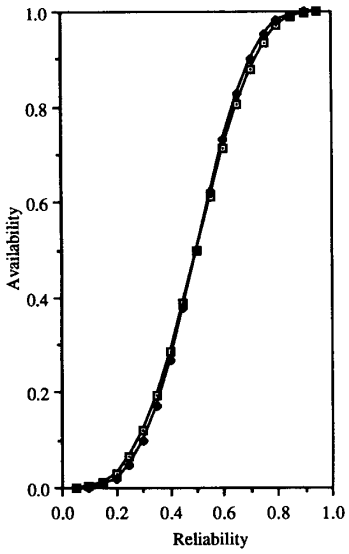


Fig. 4. Availability versus site reliability (nine copies).

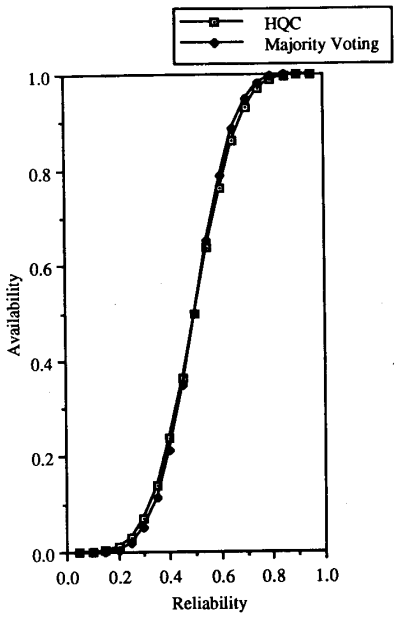


Fig. 5. Availability versus site reliability (15 copies).

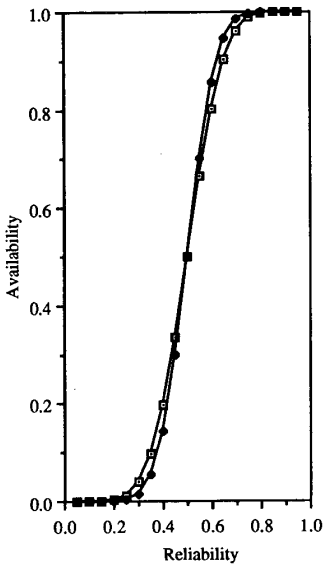


Fig. 6. Availability versus site reliability (27 copies).

This recurrence does not have a solution because it contains nonlinear terms.

To compare the two methods, the availability of each algorithm was plotted against p for the following cases: nine copies, 15 copies, 27 copies and 45 copies. In each case, the copies were organized into a multilevel hierarchy by applying Algorithm 2. These plots are shown in Figs. 4–7. The graphs show that in all cases the HQC algorithm does better than majority voting for p less than 0.5, while for values of p greater than 0.5, the HQC algorithm is inferior to QC; however, when the number of copies is large, and p is greater than 0.8, the availability in the two cases is comparable. Finally, in the range of p from 0.5 to 0.8, the majority voting algorithm does better than HQC.

VI. Tradeoffs Between HQC and Related Algorithms

This section discusses various tradeoffs between HQC and three other quorum-based methods: quorum consensus, the $\sqrt{N}$ algorithm [14] and the tree protocol [1]. The quorum consensus algorithm has already been described in Section II, while a brief description of the $\sqrt{N}$ algorithm and the tree protocol follows.

The $\sqrt{N}$ algorithm is based on associating coteries with each copy, and preassigning them. Given n copies, each copy participates in $\sqrt{n}$ coteries and the size of each coterie is also $\sqrt{n}$. It is shown in [14] that any pair of such coteries intersects and, thus, leads to mutual exclusion. No distinction is made between read and write coteries in this algorithm.

The tree protocol is based on organizing a set of n physical copies as nodes of a binary tree of $\log_2 n$ levels. [1] shows that a quorum is formed by including all the copies along any path that starts at the root and terminates at the leaves of this tree. In case a copy (or node) along a path is unavailable, then a quorum can still be formed by substituting for that copy with all the copies along two paths starting from the two child nodes of the unavailable node to the leaf level of the tree. Therefore, this algorithm has the feature of "graceful degradation," i.e., the size of a quorum is small when all copies are up, and increases gradually as more copies go down.

The discussion here is based on five criteria and is summarized in Table II. The first two criteria are the best and worst case quorum sizes, respectively. The $\sqrt{N}$ algorithm can form a quorum with fewer copies than the HQC method ($n^{0.5}$ versus $n^{0.63}$) in both the best and worst case. The best case

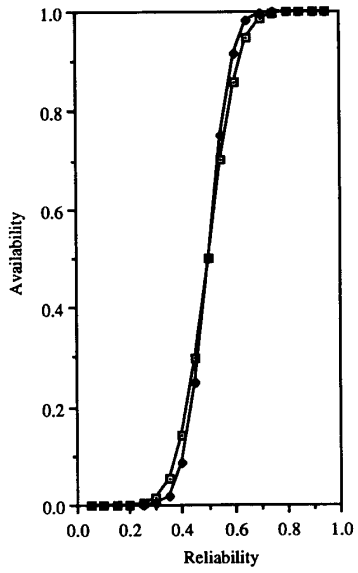


Fig. 7. Availability versus site reliability (45 copies).

TABLE II
TRADEOFFS BETWEEN FOUR QUORUM-BASED ALGORITHMS

	QC	HQC	$\sqrt{N}$ Algo.	Tree Algo.
1) quorum size (best case)	$\lceil \frac{n+1}{2} \rceil$	$n^{0.63}$	$n^{0.5}$	$\log_2 n$
2) quorum size (worst case)	$\lceil \frac{n+1}{2} \rceil$	$n^{0.63}$	$n^{0.5}$	$\lceil \frac{n+1}{2} \rceil$
3) Is algo. fully distributed?	yes	yes	yes	no
4) Cost of one node failure (worst case)	1	1	$n^{0.5}$	$\log_2 n$
5) Can read/write quorums be varied?	yes	yes	no	no

quorum size is the smallest for the tree algorithm; however, its worst case size is as large as that for the quorum consensus algorithm.

The third criterion considered was the extent to which each algorithm is fully distributed. While in the other three algorithms all copies assume equal burden for synchronization, the tree protocol differs in that it places greater burden on the copies (or nodes) which are higher in the hierarchy as compared to the ones that are lower. Therefore, it is not a fully distributed algorithm in this sense.

The next criterion was the impact that a single node failure would have on the size of a quorum. For the $\sqrt{N}$ algorithm to form a quorum all the nodes in a given coterie must be up. If all but one are available, a new coterie must be formed with another set of $\sqrt{n}$ nodes. In the case of the tree protocol, the failure of, say, the root node would require a quorum of size $2 \log_2 n$, i.e., twice the minimum size. On the other hand, in the HQC and QC algorithms the cost of a single node failure is only 1.

Finally, the possibility of varying the quorum sizes for read and write operations was considered. In an environment where there is a predominance of read operations, it would be useful to treat the two kinds of operations differently, and have a smaller read quorum and a larger write quorum. The $\sqrt{N}$ algorithm and the tree protocol, being mutual exclusion algorithms, do not allow this. On the other hand, this can easily be done in the QC and HQC methods.

This discussion shows that although HQC dominates the QC algorithm, there is no clear winner among the remaining three algorithms.

VII. CONCLUSION

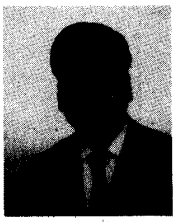
In a replicated data environment, multiple copies of an object must be kept synchronized by means of a replica control scheme. The quorum consensus method is based on assigning votes to copies, and typically requires a majority of the total number of copies to participate in a quorum. In this paper, we introduced a new algorithm, also based on voting, and showed that it is possible to reduce the size of a quorum from $\lceil \frac{n+1}{2} \rceil$ copies (as in majority voting) to $n^{0.63}$ copies. It was also shown that the HQC method produces certain intersecting sets of quorums that cannot be produced in a single-level vote assignment.

The analytical expressions for the availability of the HQC and majority voting methods were given and the two methods were compared in terms of availability for different number of copies. It was found that each method did better than the other one for various ranges of p values; however, overall HQC compared quite favorably to majority voting. Furthermore, the tradeoffs between HQC and three other related algorithms, the quorum consensus method, the $\sqrt{N}$ algorithm, and the tree protocol were discussed, and the important features of each method were highlighted.

REFERENCES

[1] D. Agrawal and A. El Abbadi, "An efficient and fault-tolerant algorithm for distributed mutual exclusion," in *Proc. Eight Annu. ACM Symp. Principles Distributed Comput.*, 1989.
[2] M. Ahamad and M.H. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *IEEE Trans. Software Eng.*, vol. 15, pp. 492–495, Apr. 1989.
[3] D. Barbara, H. Garcia-Molina, and A. Spauster, "Protocols for dynamic vote reassignment," in *Proc. ACM Conf. Principles Distributed Comput.*, 1986, pp. 195–205.
[4] S.Y. Cheung *et al.*, "Optimizing vote and quorum assignments for reading and writing replicated data," in *Proc. Fifth IEEE Int. Conf. Data Eng.*, Los Angeles, CA, Jan. 1989.
[5] S.Y. Cheung *et al.*, "Multi-dimensional voting: A general method for implementing synchronization in distributed systems," in *Proc. Tenth Int. Conf. Distributed Comput. Syst.*, Paris, June 1990.
[6] D. Davcev and W. Burkhard, "Consistency and recovery control for replicated data," in *Proc. Tenth Symp. Oper. Syst. Principles*, Dec. 1985.
[7] D.L. Eager and K.C. Sevcik, "Achieving robustness in distributed database systems," *ACM Trans. Database Syst.*, vol. 8, no. 3, pp. 354–381, Sept. 1983.
[8] A. El Abbadi, D. Skeen, and F. Cristian, "An efficient, fault-tolerant protocol for replicated data management," in *Proc. 4th ACM SIGACT–SIGMOD Symp. Principles Database Syst.*, Portland, OR, Mar. 1985, pp. 215–228.
[9] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841–860, Oct. 1985.
[10] D.K. Gifford, "Weighted voting for replicated data," in *Proc. 7th ACM SIGOPS Symp. Oper. Syst. Principles*, Pacific Grove, CA, Dec. 1979, pp. 150–159.

[11] M. Herlihy, "Dynamic quorum adjustment for partitional data," *ACM Trans. Database Syst.*, vol. 12, no. 2, pp. 170–194, June 1987.
[12] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proc. 1987 ACM SIGMOD*, San Francisco, CA, May 1987, pp. 227–238.
[13] A. Kumar and A. Segev, "Optimizing voting-type algorithms for replicated data," Lecture Notes in Computer Science, vol. 303, J. W. Schmidt, S. Ceri, and M. Missekoff, Eds. New York: Springer-Verlag, Mar. 1988, pp. 428–442.
[14] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, May 1985.
[15] J. F. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *Proc. Sixth Int. Conf. Distributed Comput. Syst.*, May 1986.
[16] J. F. Paris and D. Long, "Efficient dynamic voting," in *Proc. Fourth IEEE Int. Conf. Data Eng.*, Los Angeles, CA, Jan. 1988.
[17] R. H. Thomas, "A majority consensus approach to concurrency control," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180–209, June 1979.



Akhil Kumar received the B. Tech degree in electrical engineering from the Indian Institute of Technology, New Delhi, in 1978, the M.B.A. degree from the Indian Institute of Management, Ahmedabad, in 1980, and the Master's degree in computer science in 1986 and the Ph.D. degree in management information systems in 1988 from the University of California, Berkeley.

He is an Assistant Professor of Information Systems in the Johnson Graduate School of Management, Cornell University, Ithaca, NY. His research interests are in database systems, distributed systems, and expert systems. He has prior work experience in industry for four years in the area of systems analysis and design.