LBL-25468
UC-405

Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA, BERKELEY

Information and Computing Sciences Division

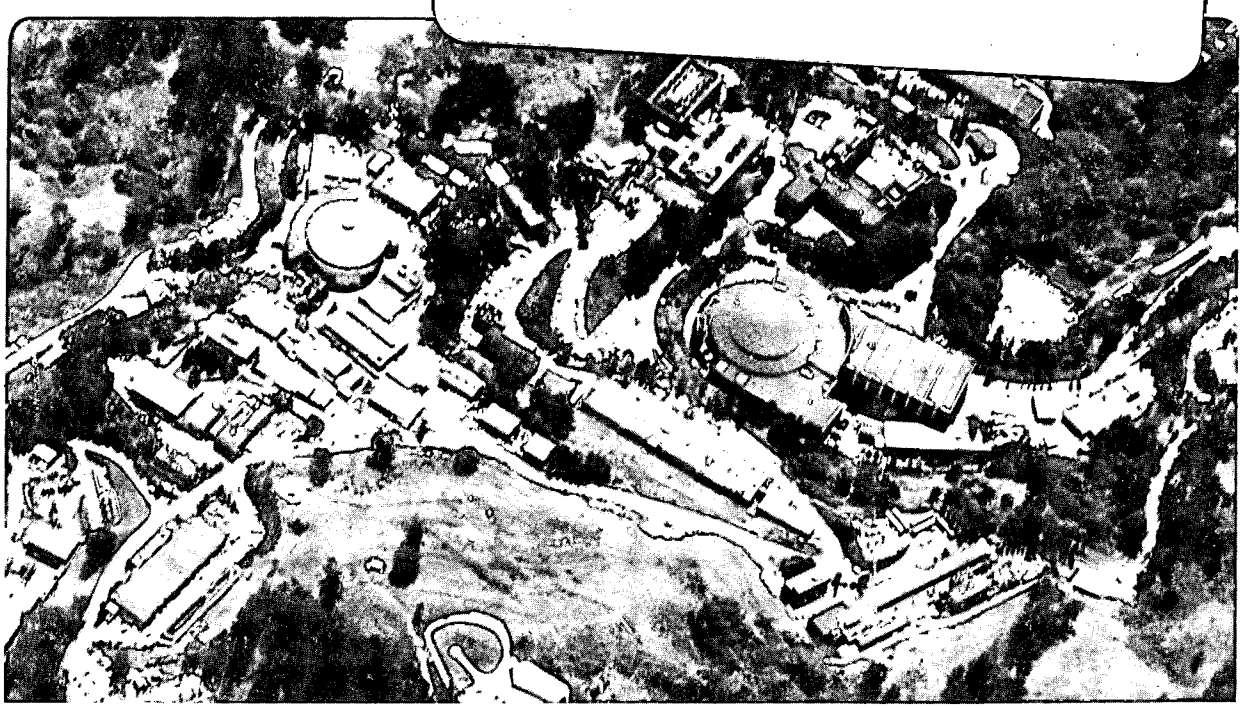
DEC 2 1988

Cost and Availability Tradeoffs in Replicated Data Concurrency Control

A. Kumar and A. Segev

June 1988

TWO-WEEK LOAN COPY
This is a Library Circulating Copy
which may be borrowed for two weeks.



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

LBL-25468 c.2

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-25468

Cost and Availability Tradeoffs in Replicated Data Concurrency Control

Akhil Kumar
School of Business Administration
University of California
Berkeley, California 94720

Arie Segev
School of Business Administration
University of California, Berkeley
and
Information and Computing Sciences Division
Computer Science Research Department
University of California
Berkeley, California 94720

June 1988

This research was supported by the Applied Mathematics Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098 and by an Arthur Andersen & Company Foundation Doctoral Dissertation Fellowship.

LBL-25468

COST AND AVAILABILITY TRADEOFFS IN REPLICATED DATA CONCURRENCY CONTROL

Akhil Kumar and Arie Segev

School of Business Administration
University of California, Berkeley
and
Information and Computing Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

Abstract

Techniques for optimizing a static voting type algorithm are presented. Our basic optimization models are based on minimizing communications cost subject to given reliability constraints. Two models are presented; in the first model the reliability constraint is failure tolerance, while in the second it is availability. Other simpler models that are special cases of these two basic models and arise from making simplifying assumptions such as equal vote values or constant inter-site communications costs are also discussed. We describe a semi-exhaustive algorithm and efficient heuristics for solving each model. The algorithms utilize a novel signature-based method for identifying equivalent vote combinations, and an efficient procedure for computing availability. Computational results for the various algorithms are also given. Finally, the optimized static algorithm was compared against the available copies method, a dynamic algorithm, to understand the relative performance of the two types of algorithms. We found that if realistic reconfiguration times are assumed, then no one type of algorithm is uniformly better. The factors that influence relative performance have been identified.

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research U.S. Department of Energy under contract DE-AC03-76SF00098 and by an Arthur Andersen & Co. Foundation Doctoral Dissertation Fellowship.

1. Introduction

A replicated data environment is one in which multiple copies of a file are present. By replicating data, system reliability may be increased to satisfy the high up-time requirements in on-line applications such as banking and airlines. Clearly, if copies of a file reside on several computers with independent failure modes, then the file system would be more reliable. The disadvantage, however, is that the copies must be kept mutually consistent by synchronizing transactions at different sites so that a global serialization order is ensured. For instance, two independent transactions must not be allowed to simultaneously update different copies of the same file. Hence, the concurrency control algorithm becomes more complex and also more expensive to implement. The additional communications and processing cost arises because several rounds of messages must be exchanged with other sites during the execution of the algorithm.

Several popular methods for replicated data concurrency control are based on the formation of quorums [BERN87, DAVI85]. We refer to such methods as "voting-type" or quorum consensus (QC) class of algorithms [GIFF79, THOM79].

The common feature in voting-type algorithms is that each site, i is assigned a vote, v_i, and in order to perform various operations quorums must be formed by assembling votes. To perform a read (or write) operation, a transaction must assemble a read (or write) quorum of sites such that the votes of all the sites in the quorum add up to a predefined threshold, Q_r (or Q_w). The basic principle behind the algorithm is that the sum of these two thresholds must exceed the total sum of all votes, i.e.,

$$\text{Invariant 1:} \quad \sum_i v_i < Q_r + Q_w$$

Hence a read and a write operation cannot proceed simultaneously. Moreover, the write threshold is larger than half the sum of all votes, i.e.,

$$\text{Invariant 2:} \quad \sum_i v_i < 2 \times Q_w$$

Thus, two write operations are prevented from proceeding simultaneously. It is important to note that the above two invariants do not enforce unique values upon Q_r and Q_w. Furthermore, the v_i's do not have to assume unique values. Hence, several alternative sets of solutions for these variables exist. The read-one write-all method is a special case of the quorum consensus method with each v_i and Q_r equal to 1, and Q_w equal to n (assuming there are n copies of the file). This leads to better performance for queries at the expense of poorer performance for updates and works well in an environment where a very large fraction of the transactions are queries.

The basic QC algorithm described above is a static algorithm because the votes and quorum sizes are fixed a priori. This restriction does not apply to dynamic algorithms, and in these the votes and quorum sizes are adjusted suitably as sites fail and recover. Examples of such algorithms are: the **Missing Writes method** [EAGE81, EAGE83], and the **Virtual Partition method** [ELAB85]. In the Missing Writes method, the read-one write-all method is implemented when all the sites in the network are up; however, if any site goes down, the size of the read quorum is increased while that of the write quorum is reduced. After the failure is repaired it reverts to the old scheme. In the virtual partition algorithm, each site maintains a view consisting of all the sites that it can communicate with, and within this view, it implements the read-one write-all method.

Other quorum based methods of the dynamic type are: the **vote reassignment method** [BARB86], the **quorum adjustment method** [HERL87], and the **dynamic voting algorithm** [JAJO87]. In the vote reassignment and the quorum adjustment methods, sites that are up can change their votes on detecting failures of other sites by following a certain protocol. In the dynamic voting method, additional information regarding the number of sites at which the most recent update was performed is stored. This makes it possible to shrink the size of a quorum dynamically if site failures or partitions occur. For instance, consider an object replicated at 5 sites designated as A, B, C, D and E, and further assume that the most recent update to this object

was performed at sites A, B, and C. If a subsequent link failure isolates site A from B and C, the latter two sites can still continue to perform updates because they contain a majority among the sites at which the most recent update was performed. Therefore, this method allows updates to take place even with fewer than a majority of the sites available.

Another dynamic algorithm is the **available copies** method [BERN84]. In this method, query transactions can read from any single site while update transactions must write to all the sites that are up. Moreover, in order for the algorithm to work correctly, each transaction must perform two additional steps called missing writes validation and access validation. In the first, a transaction must ensure that all copies it tried to, but could not, write are still unavailable. In the second step, a transaction must ensure that all copies it read or wrote are still available. The **primary copy** algorithm [STON79] is based on designating one copy as a primary copy, and each transaction must update it before committing. Later, updates are spooled to the other copies also. If the primary copy fails, then a new primary copy is designated.

In [GARC84, GARC85], the concept of coteries is introduced as an alternative to quorums. A coterie is defined as a collection of intersecting minimal subsets of sites. It is shown that there exist several coteries which do not have a corresponding vote assignment, and therefore, the solution space of vote assignments is a subset of the coterie solution space. It is also shown that the problems of finding an optimal vote assignment and an optimal coterie assignment in order to maximize availability have exponential complexity.

Our objective in this paper is to develop techniques for optimizing the assignment of votes in a static algorithm. Two models for minimizing communications cost subject to a given reliability constraint are discussed. In the first model, the reliability criterion is failure tolerance, while in the second one it is availability. Techniques for solving these models are described, and computational results to evaluate these techniques are presented. Finally, we also compare the relative performance of an optimized static algorithm against a dynamic algorithm and identify the condi-

tions under which each type of algorithm performs better. Static algorithms have the advantage of simplicity and ease of implementation, and consequently, if the performance of the two types of algorithms is comparable, then static algorithms would be preferred.

The organization of this paper is as follows. In Section 2, we define failure tolerance and availability, discuss our optimization models and present a general framework for understanding vote assignment problems. In Section 3, we define basic concepts which are used later in this paper. Then, in Section 4, we analyze the complexity of the vote assignment problem. Sections 5 and 6 deal with the special case of constant unit inter-site communications cost. In Section 5, we examine the behavior of the total communications cost for different levels of failure tolerance in the case of equal vote assignment (i.e., $v_i = 1$, for all sites) and show that lower communications cost and higher failure tolerance may be conflicting objectives. We also determine the cheapest algorithm for different read-write volume ratios in the absence of reliability considerations. Section 6 illustrates that by assigning unequal votes, substantial savings in communications cost may be achieved while maintaining a desired level of failure tolerance. The algorithm for such a vote assignment and the analysis of its cost behavior are presented; a proof of optimality for our algorithm and a qualitative discussion of its pros and cons are also given. Sections 7 through 10 address the general case of unequal votes and also unequal unit inter-site communications cost. In Section 7, we devise an efficient algorithm for computing the availability (to be used by subsequent vote assignment algorithms) of a given vote assignment. Section 8 turns to a semi-exhaustive algorithm for solving the vote assignment problem. This algorithm is computationally very expensive, but gives a near-optimal solution. Then, in Section 9, we develop greedy heuristics[†] and in Section 10, we present results from computational experiments to evaluate their performance against the semi-exhaustive algorithm. In Section 11, the optimized static algorithms are compared against the available copies algorithm, a dynamic technique. Section 12 concludes

[†] Strictly speaking, the semi-exhaustive algorithm is also a heuristic. We use the term "semi-exhaustive" to indicate the more extensive search of the solution space.

the paper with a summary and some ideas for future research.

2. Models and Framework

In this section we shall define and illustrate concepts pertaining to our two measures of reliability, failure tolerance and availability. We shall also discuss two optimization models and present a general framework which is useful for understanding vote assignment problems.

2.1. Failure Tolerance

Failure tolerance of a vote assignment is defined as the maximum number of site failures that the assignment can tolerate.[‡] An assignment can tolerate k site failures if a read and a write quorum can be formed in spite of any k-site failure. This implies that the most recent updates are available at one or more sites even if any set of k sites is down.

For instance, it is obvious that the read-one write-all algorithm does not permit write operations if even one site is down because all sites must participate in a write quorum.[†] Hence, the failure tolerance is 0. On the other hand, if all votes are equal, Q_r is 2 and Q_w is $n-1$, it means that $n-1$ sites must participate in a write quorum, and hence both read and write operations can continue despite one failure. Similarly, a Q_r value of 3 and a Q_w value of $n-2$ make the system resilient to up to 2 failures. It is a simple extension to show that the failure tolerance for the case of each $v_i=1$ is maximized when both quorums are equal to $(n+1)/2$. Hence, resiliency may be increased by modifying quorum sizes, although this may mean a higher communications cost as we will show in Section 5.

[‡] Note that the terms resiliency and failure tolerance will be used interchangeably in this paper.

[†] The available copies algorithm, described earlier, is a dynamic algorithm which overcomes this problem. It shall be examined further in Section 11.

2.2. Availability

Availability is defined in probabilistic terms as follows: Given n copies of a file such that each is up with probability p_i, the **availability,** AV, is the probability that both a read and a write quorum can be formed. Consider an example where 3 copies of a file reside at different sites. If each p_i is 0.9, and the size of a quorum is 2, then the overall availability of the file is computed as:

$$AV = 3\times(0.9)^2\times0.1 + (0.9)^3=0.97.$$

This means that by replicating a file at 3 sites instead of 1, the availability can be increased from 0.9 to 0.97. One can similarly show that if n is 5, and the quorum size is 3, then AV increases to 0.991.

2.3. Optimization Models

The two main models of interest here are:

Model 1: Find a Vote Assignment to Minimize Communications Cost

such that:

failure tolerance ≥ cut-off

Model 2: Find a Vote Assignment to Minimize Communications Cost

such that:

availability ≥ cut-off

Next we describe a framework in which special cases of these 2 models are also included.

2.4. A Framework

Table 1 gives a general framework for understanding the vote assignment problem, and for integrating vote assignments with reliability. In the two models above, no assumptions are made regarding the unit inter-site communications costs or the vote values. Making such assumptions

leads to special cases of the basic models. The objective of drawing up such a framework is to identify all special cases, and, if possible, look for simpler techniques for solving them rather than solving the more general problem in all cases.

The 3 columns of Table 1 correspond to the unconstrained† and constrained cases, while the rows of the table correspond to the 4 possible combinations of equal or unequal votes together with equal or unequal unit inter-site communications cost. Therefore, there are 12 different cases of interest altogether as represented by the 12 boxes in Table 1. We shall refer to the 12 corresponding problems as assignment problems 1 through 12. The main objective in all cases is

	Minimize Cost (unconstrained)	Model 1: Minimize Cost s.t. FT > FT_{min}	Model 2: Minimize Cost s.t. AV > AV_{min}
EQUAL VOTES c_{ij}=c	1. analytical solution (Section 5)	2. by enumeration find q_r, q_w	3. by enumeration find q_r, q_w
$c_{ij} \neq$c	4. by enumeration find q_r, q_w	5. by enumeration find q_r, q_w	6. by enumeration find q_r, q_w
UNEQUAL VOTES c_{ij}=c	7. analytical solution (Section 5)	8. Algorithm OPT (Section 6)	9. same as 12.
$c_{ij} \neq$c	10. same as 11.	11. IP Formulation [KUMA88]. Algorithms SE-FT, PE-FT1, PE-FT2. (Sections 7,8)	12. Algorithms SE-AV, PE-AV1, PE-AV2. (Sections 7,8)

Table 1: A Framework for vote assignment problems

† The unconstrained problem is a special case resulting when the constraint on failure tolerance (or availability) is removed.

cost minimization; however, this objective may be either unconstrained or it may be constrained by a certain reliability criterion such as availability or failure tolerance. It is evident that the last row of the table represents three general problem formulations, and all the other rows are special cases of these general ones.

In assignment problems 1 through 6, all sites are assumed to have equal votes. Problem 1 corresponds to unconstrained minimization of cost given equal votes and uniform inter-site communications cost between all pairs of sites. This is analyzed in Section 5. The assignment problems 2 through 6 can be solved as follows. Since all votes must be equal, without loss of generality, each vote may be assumed to be 1; the only unknowns, therefore, are q_r and q_w. Moreover, the sum of the two quorums is $n+1$, and q_w must be at least equal to $\left\lceil \frac{n+1}{2} \right\rceil$. Hence, one may determine the optimal value of q_r and q_w by enumerating the communications cost for all values of q_w from $\left\lceil \frac{n+1}{2} \right\rceil$ to n, and selecting q_w and q_r corresponding to the minimum cost alternative. Since the complexity of enumeration here is O(n), it is a simple and efficient solution method.

In assignment problems 7 through 12, the votes assigned to different sites may be non-equal. The solution method for problem 7 is the same as that for problem 1, described in Section 5. Algorithm OPT, discussed in Section 6, gives an optimal solution for assignment problem 8. In a previous paper [KUMA88], we have formulated the assignment problem 11 as a mixed Integer Programming problem. Since the known solution methods for such problems are not very efficient we found that to solve this formulation for values of n greater than 5 consumes very large amounts of computational resources. Therefore, here we have developed heuristic solution methods for this problem. In Sections 7 and 8, we discuss heuristic algorithms to solve assignment problems 11 and 12.

In the next section, we shall define some basic vote assignment concepts and use them later in the design of our algorithms.

3. Vote Assignment Definitions and Concepts

In this section, we shall define some basic concepts in the context of the vote assignment problem and shall use them through the remainder of the paper.

3.1. Vote Assignment versus Vote Combination

At the outset, a distinction must be drawn between a **vote assignment** and a **vote combination.** A vote combination for n sites is an n-tuple consisting of n elements. It becomes a **vote assignment** when each element in this combination is assigned to a **specific** site. Therefore, if each element in a vote combination is unique, then there are n! specific assignments for that combination and they may be produced by enumerating all permutations of the vote combination. On the other hand, if duplicates are present in a vote combination, then the number of corresponding assignments is less than n!. In the extreme case if all elements in the combination are equal, then only one vote assignment exists. For example, in a 5-site problem, the vote combination (1,1,1,1,1) is also a vote assignment because only one permutation exists for this combination of votes. On the other hand, the combination (5,4,3,2,1) has 5! (or 120) different assignments.

Without loss of generality, we shall represent a vote combination as a non-increasing sequence of votes, $V = (v_i, \cdots, v_n)$ where:

$$v_1 \geq v_2 \geq \cdots \geq v_j \geq \cdots \geq v_n$$

Notationally, we shall represent a vote assignment, $\bar{V}$, in a similar way; however, in an assignment $\bar{V}$, the ith element of the vector specifically represents the vote given to site i and therefore, the elements occur in site-number order.

3.2. Quorum Size

The size of a quorum, q is the minimum number of votes needed to make a majority among all votes. For simplicity, in the case of Model 2, a read and a write quorum are required to be equal since the system is considered unavailable if either quorum cannot be formed. (Note, however, that we allow non-equal votes).

3.3. Corresponding Assignment for a Vote Combination

A vote combination can also be viewed as an assignment in which the ith element is assigned to site i. Such an assignment will be called the **corresponding assignment** for a vote combination. For example, $\bar{V}$=(5,4,2,2,1) is the corresponding assignment for the combination, V=(5,4,2,2,1). For brevity, the term "availability of a vote combination" will refer to the availability of the corresponding assignment for the vote combination.

3.4. Equivalent Vote Assignments and Combinations

Two vote assignments, $\bar{V}_1$ and $\bar{V}_2$ are **equivalent** if for every group of sites in $\bar{V}_1$ that can form a quorum, the same group of sites in $\bar{V}_2$ can also form a quorum and vice versa. For example, it is easy to verify that (1,1,1,1,1) and (2,2,2,2,1) are equivalent vote assignments. In the former case, q is 3, while in the latter it is 5. We define two vote combinations, V_1 and V_2 to be equivalent if their corresponding assignments are equivalent.

The following result is derived from our definition of equivalence.

Theorem 1: The availability of two equivalent vote assignments, $\bar{V}_1$ and $\bar{V}_2$ is always equal for all sets of p_i's, where p_i is the reliability of site i.

Proof : Given n sites there are exactly 2^n alternative states representing all combinations of up and down sites. We define X_i, $i=1, \cdots, 2^n$, for any vote assignment as follows:

$$X_i = \begin{cases} 1, & \textit{if a quorum can be formed in state } i \\ 0, & \textit{otherwise} \end{cases}$$

Moreover, the Availability, AV is expressed as:

$$AV = \sum_{i=1}^{2^n} X_i R_i$$

where

R_i: probability that the system is in state i.

Clearly, R_i, for all states i, is independent of a specific vote assignment and depends only upon the individual site reliabilities. Moreover, from the definition of equivalence, if any two assignments $\bar{V}_1$ and $\bar{V}_2$ are equivalent, then:

$$(X_i)_{\bar{V}_1} = (X_i)_{\bar{V}_2} \quad for\ all\ i$$

where

$(X_i)_{\bar{V}_1}, (X_i)_{\bar{V}_2}$: X_i values for assignments $\bar{V}_1$ and $\bar{V}_2$, respectively

Since each X_i and R_i are equal for 2 equivalent assignments for all values of i, it follows that the availability for two equivalent assignments must also be equal. □

Corollary 1: The availability of two equivalent vote combinations, V_1 and V_2 is always equal for all sets of p_i's where p_i is the reliability of site i.

The corollary follows directly from Theorem 1 and Section 3.3. These results are used in Procedure SE, described in Section 8.1.

3.5. Dominating Vote Assignments

An assignment, $\bar{V}_1$ dominates another assignment, $\bar{V}_2$ if for each set of sites in $\bar{V}_2$ that can form a quorum, a quorum can always be formed by the same or a smaller set of sites in $\bar{V}_1$; however, the converse is not true. This means that $\bar{V}_1$ is superior to $\bar{V}_2$ because it would have a greater availability.

It has been shown in [GARC85] that an assignment in which the total number of votes is even is always dominated by a vote assignment in which the total number of votes is made odd

by assigning 1 extra vote. For example, if n is 4, an "odd" assignment represented by (2,1,1,1) is superior to the "even" assignment (1,1,1,1). We shall, therefore, consider only those assignments (or combinations) in which the total number of votes assigned is an odd number.

3.6. Active Votes

A vote, v_j assigned to site j in an assignment, $\bar{V}$ is said to be an **active** vote if there exists at least one quorum of sites which includes site j such that if site j is withdrawn, the remaining sites do not form a quorum.

In the assignment (4,2,2,2,1), v_5 does not contribute to the formation of any quorum, because all quorums must include at least two of the 4 other sites. Therefore, v_5 is an inactive vote and this assignment is treated as an invalid 5-site assignment. For an n-site problem, we shall restrict ourselves to those assignments in which each vote is active.

4. Complexity of the Vote Assignment Problem

In [GARC85] it has been shown by heuristic arguments based on visualizing a hypercube that an upper bound on the total number of vote assignments for n sites is 2^{n^2}. Here we give another derivation which is intuitively simpler.

For n sites, there are 2^n possible groups corresponding to all possible combinations involving the inclusion and exclusion of each site. From these 2^n groups, we may arbitrarily choose any n groups that should form a quorum, and write n simultaneous equations with n variables. Each system of n equations may then be solved for the v_i's. Of course, some of the systems of equations may not have any solution. However, this allows us to place an upper bound on the number of vote assignments as $\left[\begin{matrix} 2^n \\ n \end{matrix}\right]$. This expression represents the number of ways in which n elements may be chosen from a universe of 2^n elements. The complexity of the above expression is $O(2^{n^2})$.

Not only does this number include several systems of equations which have no solution, but it also involves a large number of permutations of a given vote combination. For instance, (5,3,3,1,1), (3,1,1,5,5), (1,1,3,3,5), etc., are examples of permutations of the vote combination, (5,3,3,1,1), and each such permutation is included in the upper bound above. For these two reasons, it is possible that a tighter upper bound may exist for the total number of unique combinations, but has not yet been found. In the next section, we develop an algorithm that enumerates a very large number of vote combinations, and also implement this algorithm to show the number of assignments that exist for $n \leq 9$.

5. Analysis of QC Algorithms (constant c_{ij}'s and v_i's)

In this section, we restrict our attention to voting-type or QC algorithms with equal vote assignments (v_i's) and constant inter-site communications costs (c_{ij}'s). We will assume, without loss of generality, that all files are completely replicated. First, in Section 5.1, we analyze the minimum cost of QC at different values of read-write traffic volumes, disregarding resiliency considerations. Next, the resiliency versus communications cost trade-off is investigated for different values of update-to-query ratio in Section 5.2.

The main parameters of interest are:

n : number of sites where copies of the file exist

r_i : volume of read (or query) traffic from site i

w_i : volume of write (or update) traffic from site i

vol_i : total volume of read and write traffic from site i (r_i+w_i)

V : total read and write traffic volume of all sites ($\sum_i vol_i$)

Q_r : size of the read quorum

Q_w : size of the write quorum

The write ratio, ρ_w is defined as:

$$\rho_w = \sum_i w_i / (\sum_i w_i + \sum_i r_i)$$

If blind writes are not allowed, and each write operation is preceded by an independent read operation, then the maximum value ρ_w would take is 0.5. We do not consider lock escalation and rather assume that a write-lock would be obtained in the first instance when a transaction reads an object for updating. In our model, therefore, ρ_w can also assume values higher than 0.5.

5.1. Minimum Cost Voting

The communications cost, CC_{qc} for the QC algorithm is the sum of the total traffic resulting from read and write transactions, and is expressed as:

$$CC_{qc} = (Q_w - 1) \times \sum_i w_i + (Q_r - 1) \times \sum_i r_i$$

We will denote the minimum cost by CC^*_{qc}. Notice it is assumed that each site will include its local copy while assembling a quorum and hence it communicates with $(Q_r - 1)$ and $(Q_w - 1)$ additional sites to form read and write quorums respectively. There are two important cases of interest: $\rho_w \leq 0.5$, and $\rho_w > 0.5$. We shall analyze each one separately.

Case 1: $\rho_w \leq 0.5$

Now, since $Q_r + Q_w$ must always be greater than $n+1$ (see invariant 1 above), and $\sum_i w_i \leq \sum_i r_i$ when $\rho_w \leq 0.5$ (by definition), it follows that CC_{qc} is minimized when Q_r is 1 and Q_w is n. This is exactly the read-one write-all algorithm. Hence, we get the following result:

Result 1: The read-one write-all algorithm incurs the lowest cost when $\rho_w \leq 0.5$.

The minimum communications cost in this case is derived from the general expression by substituting the quorum sizes and is:

$$\begin{aligned} CC^*_{qc} &= (n-1) \times \sum_i w_i \\ &= (n-1) \times \sum_i vol_i \times \rho_w \end{aligned}$$

$$= (n-1)\,\rho_w V$$

For large n, this cost may be approximated as $n\,\rho_w V$.

Case 2: $\rho_w \geq 0.5$

Reasoning along the same lines as above, it might appear that the communications cost is minimized when Q_r is n and Q_w is 1. However, this is incorrect because invariant 2 is violated. Clearly, Q_w should assume the smallest possible value without violating invariant 2 and this occurs when:[†]

$$Q_w = int(n/2) + 1$$

and, therefore,

$$Q_r = n + 1 - Q_w.$$

Result 2: Hence, when $\rho_w \geq 0.5$, CC_{qc} is minimized at the above quorum values.

For relatively large values of n, we may approximate the quorum sizes as:

$$Q_w = Q_r = n/2$$

Using this approximation, CC^*_{qc} is expressed as:

$$\begin{aligned} CC^*_{qc} &= (n/2) \sum_i (r_i + w_i) \\ &= (n/2) \sum_i vol_i \\ &= (n/2)\, V \end{aligned}$$

Note that in this case the quorums are either equal or differ by 1. For example, when n=11, Q_r=Q_w=6. On the other hand, when n = 12, Q_w=7 and Q_r=6. The minimum cost is plotted against ρ_w in Figure 1. It should be observed that this cost increases linearly as ρ_w goes from 0 to 0.5 but becomes constant for $\rho_w \geq 0.5$

[†] int(x) denotes the integer part of x.

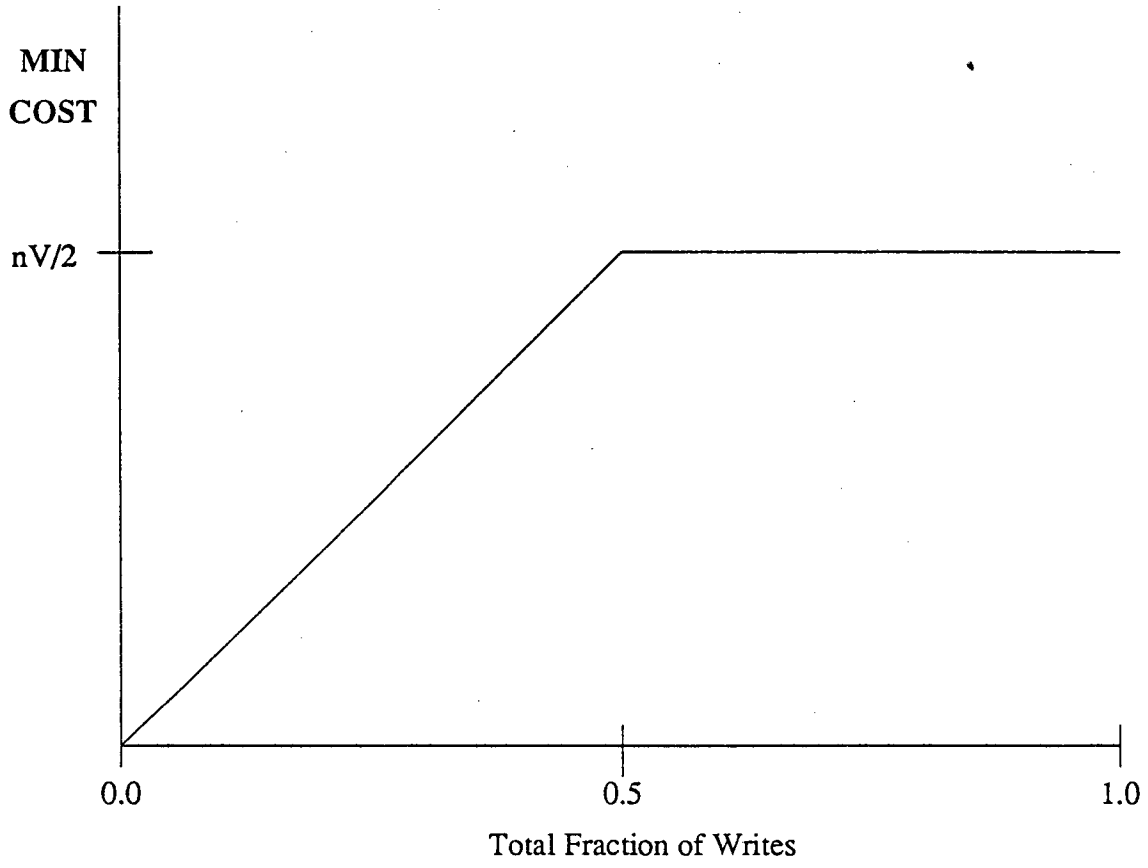


Figure 1: Plot of Min. Cost versus total fraction of writes (ρ_w)

5.2. Failure Tolerance versus Communications Cost

So far in our analysis, minimum communications cost has been determined without regard to resiliency . Here we will study the implications of increased resiliency on communications cost. Again, there are two cases: $\rho_w \leq 0.5$, and $\rho_w \geq 0.5$.

In the second case, as indicated by Result 2 and the subsequent example above, the two quorum sizes are either equal or differ by 1. This means that the availability is maximum as well. Hence, we may conclude the following:

Result 3: When $\rho_w \geq 0.5$, the failure tolerance is maximized at the least-cost quorum sizes.

Now we turn to the case of $\rho_w \leq 0.5$. In the least-cost alternative, $Q_r=1$, $Q_w=n$, and the failure tolerance is 0. Therefore, in order to assure a failure tolerance level of k, Q_w must be reduced by k and Q_r must be correspondingly increased. The resulting incremental cost (IC) is:

$$\mathrm{IC} = (\sum_i r_i - \sum_i w_i) \times k$$

This increase is translated in terms of the minimum cost, CC^*_{qc}, and the increase percentage (IP) is computed as:

$$\mathrm{IP} = \mathrm{IC} \times 100 / CC^*_{qc}$$

After substituting for IC and CC^*_{qc}, and further simplifying, IP may be expressed as:

$$\mathrm{IP} = (1-2\rho_w) \times k \times 100 / (\rho_w (n-1))$$

In Table 2, IP has been computed for a range of values of ρ_w, with n alternately set at 5 and 10. Note that k is the failure tolerance level. It is evident from this table that for small values of ρ_w, higher resiliency is achieved only by incurring a considerably higher communications cost. For instance, when ρ_w is 0.05 and k is 1, IP is 450%. This means that in order to make the algorithm resilient to 1 site failure, the communications cost is 4.5 times larger than its value for 0 failure tolerance. However, as ρ_w approaches 0.5, IP also shrinks and tends towards 0.

In summary, the interesting conclusion to be drawn from this section is that the additional cost of providing failure tolerance is highly dependent on ρ_w. For small values of ρ_w, this cost is very high; however, for $\rho_w \geq 0.5$, it becomes 0.

ρ_w	$n = 5$		$n = 10$			
	k=0	k=1	k=0	k=1	k=2	k=3
.05	0%	450%	0%	200%	400%	600%
.10	0%	200%	0%	89%	178%	267%
.20	0%	75%	0%	33%	67%	100%
.30	0%	33%	0%	15%	30%	45%
.40	0%	12.5%	0%	6%	11%	17%
.50	0%	0%	0%	0%	0%	0%

Table 2: Percentage Increase in communications cost for higher resiliency as compared with the minimum cost (for different ρ_w values)

6. Optimal Vote Assignment (constant c_{ij}'s)

In the previous section it was shown that availability may be improved at the expense of increased communications cost by changing the quorum sizes. The consequent effect on communications cost was expressed analytically. Here we discuss an algorithm for optimally assigning votes to sites so as to minimize communications cost subject to a desired failure tolerance level. As before, failure tolerance, is defined as the number of site failures that can be tolerated. Further, it will be assumed that the inter-site unit communications cost is a constant for all pairs of sites. We first describe the algorithm, then illustrate it with an example and finally present a cost comparison between the communications cost resulting from our method and the conventional method in which equal votes are assigned to each site.

6.1. A Vote Assignment Algorithm

This algorithm assumes that there are n copies of the file and the desired fault tolerance level is k. The main steps in the algorithm are listed below and then a brief explanation follows.

Algorithm OPT

1. Number the sites as $s_1, s_2, \cdots, s_n$

2. Denote the votes to be assigned as $v_1, v_2, \cdots, v_n$

3. Assign votes to sites $s_{k+1}, s_{k+2}, \cdots, s_n$ such that:

   $v_{k+1} = v_{k+2} = \cdots = v_n = 1$

   $k \leq n/2$

4. Assign $Q_r = Q_w = n-k$

5. Assign $v_1 = v_2 = \cdots = v_k = \text{int}((n-k-1)/k)$

6. Let $m = n - k - 1 - \sum_{i=1}^{i=k} v_i$

   If $m > 0$, then assign 1 extra vote to the first m sites, $s_1, s_2, \cdots, s_m$.

Note that in step 1, the sites are numbered arbitrarily in any random order. In step 3, votes of 1 unit are assigned to $n-k$ out of the n sites. These sites are designated as **minor sites.** The basic idea of the algorithm is to assign higher votes to the remaining k sites, which we term as **major sites.** Notice that k can be at most $n/2$. This means that the maximum failure tolerance level is equal to half the total number of sites. The vote assignment scheme ensures that the set of k major sites along with any other minor site can form both a read and a write quorum. Alternatively, if all k major sites are down, then it should still be possible to form both quorums by assembling the votes of all minor sites. This guarantees that no matter which k sites are down, the quorums can still be formed. Consequently, in step 4, Q_r and Q_w are both set to $n-k$, which is the sum of the votes of all the minor sites. In step 5, an amount $n-k-1$ is distributed among the k major sites, thereby ensuring that if the votes of the major sites are added together along with one additional vote, the resulting sum is $n-k$, and hence, both quorums can be formed. The following example illustrates the main steps of Algorithm OPT:

Example

Say, $n = 8$ and $k = 2$.

Assign $v_3 = v_4 = v_5 = v_6 = v_7 = v_8 = 1$

Therefore, $Q_r = Q_w = n - k = 6$

and $v_1 = v_2 = int(5/2) = 2$

$m = 1$; hence, $v_1 = 3$.

Hence, the final vote assignment is:

$v_1 = 3$; $v_2 = 2$

$v_3 = v_4 = v_5 = v_6 = v_7 = v_8 = 1$

$Q_r = Q_w = 6$

For the above example, the conventional QC method assigns a vote of 1 to each site and sets Q_r and Q_w to 3 and 6 respectively; these quorum values ensure that the system can tolerate up to 2 site failures.

6.2. Discussion

In this section, we prove that Algorithm OPT chooses an optimal vote assignment. Further, since the algorithm does not distinguish between sites, we discuss heuristic ways of assigning the higher votes to specific sites.

Algorithm OPT is based on the following lemma:

Lemma 1: When k-site resiliency is desired, a transaction must communicate with at least k other sites in order to perform a read or write operation.

Proof : (by contradiction) Consider an update transaction, T which performs an update operation in which a certain set of $k-1$ other sites are involved. Furthermore, consider a subsequent k-site failure involving the site at which transaction T executed, and the same set of $k-1$ other sites which participated in transaction T. Clearly, in this situation the update performed by T will not

be available at any working site, and hence, k-site resiliency is not assured. Therefore, a transaction must communicate with at least k other sites in order to guarantee k-site resiliency. This is necessary in order to guarantee that even a k-site failure will leave at least one working site with the most recent updates. □

This lemma is used in the following theorem:

Theorem 2: Algorithm OPT is optimal.

Proof : From Lemma 1, each site must communicate with at least k other sites to achieve k-site resiliency. Since the inter-site communications cost is constant for all pairs of sites, a minimum cost, k-site resilient solution is obviously one in which each site communicates with **exactly** k other sites. Our algorithm ensures that if all sites are working, any site can form a read or a write quorum consisting of exactly k other sites. Hence, the communications cost is minimized. □

The intuitive rationale behind our algorithm is as follows. If the desired failure tolerance is k, then higher votes are assigned to k of the n sites, and it is guaranteed that a quorum can be formed if these k major sites and any one other site are included -- a total of $k+1$ sites. Therefore, our vote assignment scheme leads to a least-cost solution if the k major sites are included in all quorums. However, if any of these k sites is down then the transaction must communicate with many more sites in order to assemble a quorum and continue operation. Consequently, with our scheme the total communications cost increases if even one of the major sites is down. In contrast, with an "equal-vote" method this cost is not affected by site failures since all sites have an equal vote. Thus, our method achieves a lower cost under normal operation at the expense of a higher cost when any of the major sites is down. Since the latter situation will occur infrequently, our algorithm is superior to the conventional QC methods.

According to the algorithm as described so far, the assignment of the n votes to specific sites may be done entirely at random. Here we discuss some factors that could influence the choice of the k major sites. If a network is fully connected and site reliability is equal for all sites,

then these k sites may be chosen at random. However, this is not usually the case. Two important factors to be considered are site reliability and connectivity. The k larger votes should be assigned to those sites which have greater reliability and are most well-connected with the other sites. (A simple measure of the connectivity of a site is the number of other sites it has direct links with). In a star network, for example, the central site is a good candidate for a higher vote. In such a network, the connectivity of the central site is $n-1$ and that of the others is 1.

One disadvantage with our method is that the communications traffic to the major sites will be very high as compared to the traffic to the minor sites and our analysis does not consider any queuing delays and network congestion. This issue has to be addressed as a part of network design. One way to handle this is to provide high bandwidth lines between all major sites and also ensure fast access to at least one major site from each minor site. With proper network design, this handicap can easily be overcome.

In the next section we present a cost comparison between Algorithm OPT and a QC algorithm which assigns equal votes to all sites.

6.3. Cost Comparison

The cost comparison consists of two cases. First we discuss the case of $\rho_w \leq 0.5$ which is of greater interest. The general expression for the communications cost incurred by the conventional QC method was derived in the previous section. Here we modify it slightly to express this cost as a function of k, the number of site failures which must be tolerated. Therefore, the minimum communications cost in the conventional QC method when k-site resiliency is provided is as follows:

$$CC^*_{qc_k} = k \times \sum_i r_i + (n-k-1) \sum_i w_i$$

The equivalent communications cost incurred in Algorithm OPT is given by the following expression:

$$CC_{opt_k} = k \times \sum_i (r_i + w_i)$$

Therefore, the savings realized from our vote assignment is:

$$CC^*_{qc_k} - CC_{opt_k} = (n-2k-1)\sum_i w_i$$

This savings may be converted into a percentage as follows:

$$\text{Savings } \% \quad = (n-2k-1)(\sum_i w_i)\times 100/(k\times\sum_i (r_i+w_i))$$

$$= (n-2k-1)\times \rho_w \times 100/k$$

Sample calculations of the savings % with n alternately set to 5 and 10 are given in Table 3. In each case, ρ_w is varied from 0.05 to 0.5, while k is also varied over a range of values. When the failure tolerance, k is set to 0, it corresponds to the unconstrained case. It is evident from this table that large savings are achievable by using our algorithm.

In the case where $\rho_w \geq 0.5$, the expression for $CC^*_{qc_k}$ is rewritten as:

$$CC^*_{qc_k} = (n/2)\times\sum_i (r_i+w_i)$$

The expression for CC_{opt_k} remains the same as before and hence the savings % in this case is given by:

$$\text{Savings } \% = (n/2-k)\times 100/k$$

ρ_w	$n = 5$		$n = 10$			
	k=0	k=1	k=0	k=1	k=2	k=3
.05	0%	10%	0%	35%	12.5%	5%
.10	0%	20%	0%	70%	25%	10%
.20	0%	40%	0%	140%	50%	20%
.30	0%	60%	0%	210%	75%	30%
.40	0%	80%	0%	280%	100%	40%
.50	0%	100%	0%	350%	125%	50%

Table 3: Percentage savings achieved from our vote assignment method

When n is 10 and k is 1 this figure is 400%. Again the potential for savings is large.

7. Computing Availability

In this section, we devise an efficient algorithm for computing availability when several copies of a file exist, and the vote assigned to and the reliability of each site are known. One simple method for computing availability is by enumerating all possible combinations of up and down sites, identifying those combinations in which a quorum can be formed, and computing the aggregate probability of all such combinations. This is clearly an inefficient scheme. Here we describe a more efficient algorithm and show that it is considerably less expensive than complete enumeration. Reducing the computational complexity is important because the availability computation is central to all subsequent algorithms. A formal problem definition and the details of our algorithm are given in the following section. Subsequently, an example will be given to demonstrate the algorithm.

7.1. Algorithm Description

Problem Definition: Compute the availability for a vote assignment represented by the vector $\bar{V} = (v_1, v_2, \cdots, v_n)$, and a quorum of size q. The reliability, or the probability that sites i are up, is denoted by the vector $\bar{P} = (p_1, p_2, \cdots, p_n)$. Without loss of generality, it is assumed that:

$$v_1 \geq v_2 \geq \cdots \geq v_j \geq \cdots \geq v_n$$

Our Algorithm **AVAIL** for computing availability is based on first constructing a tree which we call the quorum subset tree. Each branch in this tree corresponds to the inclusion and exclusion of a certain site in a quorum and by following the path from a leaf node to the root one can generate alternative subsets. The procedure for constructing this tree is called **BUILD-TREE.** We shall describe this procedure first, and then use it as a subroutine in Algorithm AVAIL.

The root of the tree is labeled as level 1, and lower levels are numbered[†] successively. An information triplet is maintained at each node as follows:

(votes included, votes excluded, votes remaining)

where

votes included (VI): total votes of sites included so far
votes excluded (VE): total votes of sites excluded so far
votes remaining (VR): total votes still to be assigned

Based on this (VI, VE, VR) triple, a decision is made as to whether a particular node is "fathomed". If a node is fathomed, then no further branching is done from there. Otherwise, the branching process is repeated. At an unfathomed level i node, we consider the effect of including or excluding site i by constructing two branches: one corresponding to including site i and the other corresponding to excluding site i. The main steps of procedure **BUILD_TREE** are described below. Figure 2 shows a tree that has been constructed from this algorithm for $\bar{V}$ = (5,3,3,1,1).

BUILD_TREE

1. (initialization)

   $i=1$

   The root is marked by the triple (0,0,W), where W is the sum of all votes. Next, two branches are constructed from the root: one corresponding to the inclusion of site 1 (the "include 1" branch) and the other corresponding to excluding site 1 (the "exclude 1" branch). The nodes at the end of these two branches represent level 2 of the tree.

2. At each level $i+1$ node, the triple at the node is computed from the ith level parent node in the following manner. If the branch leading to the node is an inclusion branch, then:

   $\mathrm{VI} = (VI)_p + v_i$

[†] Note that in order to simplify the description the numbering is for levels (where each level number corresponds to a site number) and **not** nodes. Nodes of the tree are referred to as 'level i nodes' generically.

$VE = (VE)_p$

$VR = (VR)_p - v_i$

(The subscript p denotes the parent node values while the unsubscripted VI, VR, and VE represent the child node values).

On the other hand, if the branch is an exclusion branch, then the new values are computed as:

$VI = (VI)_p$

$VE = (VE)_p + v_i$

$VR = (VR)_p - v_i$

3. (Fathoming Step) This step is repeated for all nodes at level $i+1$.

   We consider 4 cases:

   CASE 1: $VI \geq q$

   If VI for a new node is greater than or equal to q, then this node is marked as "fathomed - type 1".

   CASE 2: $VI < q$ and $VI+VR=q$

   If $VI+VR$ for the new node is equal to q, then this means that all the remaining sites must be included in order to create a quorum. In this case, we put a note on the node to indicate that sites $i+1, i+2, \cdots, n$ must be included, and mark the node as "fathomed - type 1".

   CASE 3: $VE > W-q$

   This means that enough sites have been excluded already to preclude the formation of a quorum from the remaining sites. Hence, this node is marked as "fathomed - type 2".

   CASE 4: $VI < q$ and $VI+VR > q$

   This node cannot be fathomed.

4. If all nodes at level $i+1$ have been fathomed, then the tree construction is complete, and the algorithm stops, else

```
    {
i=i+1.
At each unfathomed level i node, construct
    an "include i" and an "exclude i" branch
go to step 2.
    }
```

Now we list the steps of Algorithm AVAIL and a detailed description follows.

Algorithm AVAIL

1. First construct a tree from procedure **BUILD_TREE** described above.

2. Set Availability, A = 0.

3. Next, for each "fathomed - type 1" node, do {

```
P = 1.
Traverse tree upwards from the node and
  Do Until root is reached {
    If there is an "include i" in the path,
      P = P × p_i,
    else
      If there is an "exclude i" in the path,
        P = P(1 − p_i).
    If current node is not the root, traverse the next higher path.
                    }
A = A + P
                    }
```

4. The Availability of the file system is given by A.

The Availability is computed by following the path from each "fathomed -- type 1" node to the root backwards. Each "include i" along a path corresponds to p_i and each "exclude i" corresponds to $(1-p_i)$. The product of the p_i's and $(1-p_i)$'s is computed along each such path, and aggregating the individual products gives the availability. The following example will demonstrate this algorithm.

7.2. An Example

Here we describe an example to illustrate algorithm AVAIL.

Example: Compute the availability for the following $\bar{V}$ vector:

$$\bar{V} = (5,3,3,1,1)$$

The first step in implementing Algorithm AVAIL is to construct the quorum subset tree. This is shown in Figure 2. Using this tree, the availability for this example is computed as:

$$A = p_1 p_2 + p_1(1-p_2)p_3 + p_1(1-p_2)(1-p_3)p_4 p_5 + (1-p_1)p_2 p_3 p_4 + (1-p_1)p_2 p_3(1-p_4)p_5.$$

8. A Semi-Exhaustive (SE) Algorithm

In Sections 5 and 6, we examined techniques to solve the special cases that arose from models 1 and 2. These solutions became easier as a result of certain simplifying assumptions that were made about the relative vote sizes or the inter-site communications costs. In the remainder of this paper we shall focus on solving the two general models, also represented by problems 11 and 12 in the framework of Table 1.

In this section we shall describe a procedure (Procedure SE) to generate non-equivalent vote combinations, and then use it to develop an algorithm Algorithms SE-AV and SE-FT to solve models 1 and 2. These algorithms produce alternative assignments for each vote combination generated by Procedure SE. For each assignment, the algorithm evaluates the cost and availability, and selects the minimum cost assignment which meets the reliability criterion.

8.1. Procedure SE

This procedure starts with an initial vote combination, and further new combinations are generated by incrementing the votes in the current combination. To make the procedure more efficient, we devise a method to identify and discard a combination which is equivalent to one already produced. Thus, only non-equivalent vote combinations are generated.

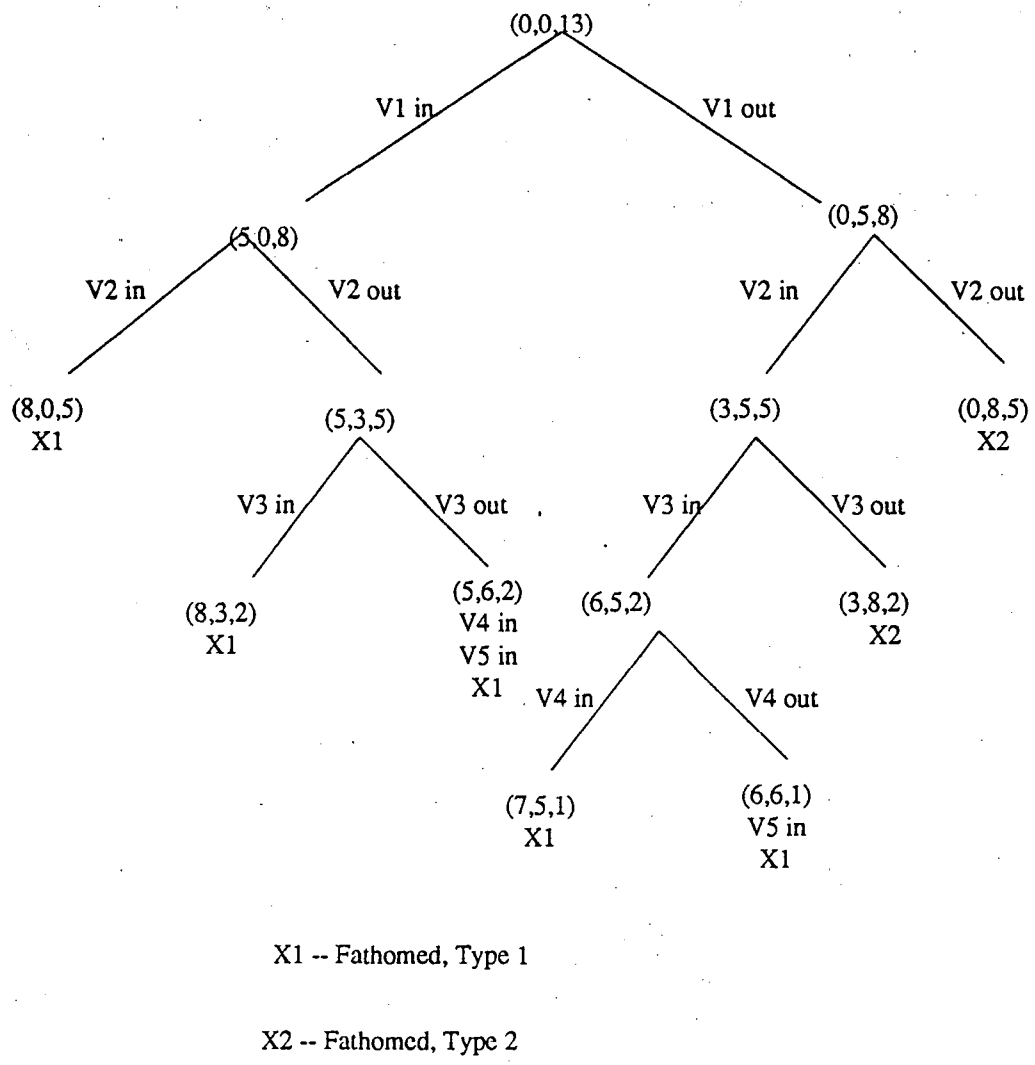


Figure 2: Quorum subset tree for the example to illustrate the computation of Availability

The initial combination is:

For odd n : v_i 's = 1 for all i ,
For even n : $v_1 = 2$, v_i 's = 1 for $i = 2, \cdots, n$

Since each v_i must be a positive integer strictly greater than 0, this is obviously the combination with the smallest odd sum of votes.

Next, we generate new combinations by considering all ways in which 2 votes may be added to the current combination. (A combination is **new** if it is not equivalent to a combination which has already been produced). This will give us a set of v_i's which add up to $n+2$ or $n+3$ depending on whether n is odd or even, respectively. At each subsequent stage, we consider only the new combinations produced in the previous stage and use them to produce further combinations. For example, say n is 5, and the starting combination is (1,1,1,1,1). At the next step, the total number of votes must be increased by 2, and the alternative combinations that result are (3,1,1,1,1), and (2,2,1,1,1). Each of these combinations will have to be checked for equivalence against the combination (1,1,1,1,1). If found to be new, then it is added to the existing list of combinations.

Each successive iteration of the while-loop in the main section of Procedure SE (described below) corresponds to finding new combinations by adding r more votes to each current N vote combination. Ordinarily N is incremented by 2 in each successive iteration and r remains at 2; however, if at any iteration no new combination is found, then in subsequent iterations r is incremented by 2 and N is kept fixed until a new combination is realized. At this point, N is incremented by the current value of r. There are two stopping rules. The first rule is that the total sum of votes becomes greater than a pre-specified maximum, while the second rule is that r becomes larger than 25. The second rule corresponds to a situation in which no new combination is found for a gap of 25 votes, a reasonably large interval. Such a gap would arise if no new combination is found even after 12 successive iterations.

The availability of each prospective new combination is computed by applying Algorithm AVAIL (described earlier in Section 7) to its corresponding assignment. The value of availability so computed is called the **signature** for the combination. Since, as shown in Section 3, two equivalent combinations will always have the same availability, this gives us a convenient way of eliminating redundant vote combinations. Each new signature is checked against an array of signatures, and if not found in it, it is added to this array. A collision occurs if the signature already

exists in the array, and in this case the combination is discarded. Since Procedure SE is used only to generate non-equivalent combinations, an arbitrary set of p_i's can be used for this purpose (the actual p_i's are, however, required to compute the real availability in Algorithm SE-AV, discussed in Section 8.3). Although the converse of Theorem 1, i.e., if two assignments have the same availability then they must be equivalent has not been proven, it has been verified experimentally that false collisions are prevented by choosing site reliabilities such that no two p_i's are equal. Therefore, this rule should be observed while selecting p_i's for use in Procedure SE.

The main steps are as follows:

Procedure SE

1. (Initialization). The starting combination is $V = (v_1, \cdots, v_n)$ such that:

   For odd n: v_i's = 1 for all i; Sum of votes, N = n.
   For even n: $v_1 = 2$, v_i's = 1 for $i = 2, \cdots, n$; N = n+1.

   Also, initialize:

   the final solution set, S = ∅,
   the solution set for a total of N votes, $S_N = \{V\}$
   the p_i values from input data,
   the signature array, SA=0.

2. (Main section)

```
r = 2.
while (N+r < N_max and r < 25){
   for each combination, V in S_N {
      add r more votes to V to create combination, V_temp.
      compute availability, AV_temp for V_temp.
      If (AV_temp ∉ SA array){
            add V_temp to S_N+r.
            add AV_temp to SA array.
                                  }
                                 }
   If (S_N+r ≠ ∅)
            r = 2.  N = N + 2.
   else
            r = r + 2.

            }
```

3. The final solution set, S is obtained by concatenating all non-empty sets, S_i.

8.2. Implementing Procedure SE

We implemented this algorithm and Table 4 shows the number of unique vote combinations that were found for various values of n. For n equal to 7, no vote combinations were found for N_{max} between 44 and 69, at which point the program terminated. When n was 8 and 9, the program was interrupted at N_{max} equal to 50 and 45 respectively. We repeated this experiment for two different sets of probability values and found the same result in each case. This was done to eliminate the possibility of two non-equivalent combinations having the same signature.

It should be reemphasized that the figures in Table 4 represent vote combinations and not specific vote assignments. For each combination, there are potentially n! assignments if each vote is unique. An exhaustive algorithm to solve models 1 and 2 must consider all the specific assignments corresponding to each vote combination, and therefore, multiplying the numbers in Table 4 by n! gives a better appreciation for the true complexity of the problem.

# of Sites(n)	Maximum Sum of Votes	# of Combinations
1	-	1
2	-	1
3	-	1
4	-	1
5	-	4
6	-	19
7	-	133
8	50	>2071
9	45	>7603

Table 4: Number of unique vote combinations for various n

8.3. Vote Assignment Algorithms

Here we shall describe Algorithms SE-FT and SE-AV to solve models 1 and 2, respectively. In both algorithms Procedure SE is used to generate a set of vote combinations, and then all possible assignments of each combination are evaluated to find the best solution.

The main objective in using Procedure SE is to eliminate non-equivalent combinations. Although Procedure SE is an availability-based procedure, it is equally applicable in solving both models because it enables us to eliminate equivalent assignments in either case. Note that two equivalent vote combinations will have identical failure tolerance and communications cost. This can be proven along the lines of Theorem 1; however, for brevity, we shall omit the details here. Intuitively, the proof again follows from the fact that if $\bar{V}_1$ and $\bar{V}_2$ are two equivalent assignments, then any group of sites in assignment $\bar{V}_1$ that form a quorum will also form a quorum in assignment $\bar{V}_2$, and vice versa.

Algorithm SE-FT

1. Produce Set S from Procedure SE.

2. (Main section)

```
min-cost = ∞
For each combination, V̄ in set S {
  if (FAILURE-TOLERANCE(V̄) ≥ cut-off)
     for each permutation, V̄_p of V̄ {
         if (cost < min-cost){
             min-cost = cost.
             min-assignment = V̄_p.
                        }
                                 }
                        }
```

3. The best solution is given by *min-assignment*.

FAILURE-TOLERANCE($\bar{V}$)

1. Find smallest ft such that:

$$\sum_{i=1}^{ft} v_i \geq q$$

Algorithm SE-AV

1. Produce Set S from Procedure SE.

2. Produce set P, the permutation set of all combinations in S.

3. (Main section)

```
min-cost = ∞
For each assignment in set P {
  compute cost, and availability of assignment, V̄.
  if (availability ≥ cut-off and cost < min-cost){
    min-cost = cost.
    min-assignment = V̄.
                                                    }
                        }
```

4. The best solution is given by *min-assignment*.

Both the above algorithms are conceptually straightforward but computationally very intensive. They utilize procedures to compute cost, failure tolerance, and availability. The procedure for computing failure tolerance, FAILURE_TOLERANCE, is listed above, while the procedure for computing availability, AVAIL, was described in Section 7 and the procedure for computing cost, COMPUTE_COST, is discussed in the next section.

9. Vote Assignment Heuristics

In Section 4 we showed that finding all vote assignments is a very complex problem because an upper bound on the number of vote assignments is $O(2^{n^2})$. In Section 8 we described algorithms SE-FT and SE-AV to solve models 1 and 2 by a semi-exhaustive method. This method was computationally very intensive even for a small number of sites like 7 or 8. Hence it is necessary to develop heuristics that examine a smaller search space and yet give a good solution.

In this section we shall describe such heuristic solutions to models 1 and 2, that incur a drastically lower computational effort as compared to the semi-exhaustive algorithm. These

heuristics are based on generating a small subset of vote combinations and then allocating them to sites in a "greedy" manner. A short outline of these algorithms is as follows:

generate a partial set of vote combinations
assign votes to sites in a greedy manner
evaluate the cost for each assignment
choose the least cost solution

We shall describe two procedures, PE1 and PE2, for generating a partial subset of vote combinations and then give the full algorithm. In the first procedure, **PE1** we apply two simple rules to generate a small number (O(n)) of vote combinations. These two rules allow us to produce unique vote assignments in which varying weights are assigned to different sites. The first rule is named the **failure tolerance is k** rule and is used to generate a unique vote combination corresponding to each value of k over the range from 1 to $FT_{\max}(n)$, the maximum value FT can assume for an n-site solution. The second rule is called the **k-sites make a majority** rule and consists of determining a vote combination in which the first k sites receive equal votes and make a quorum of value, say q, while the votes of the remaining sites add up $q-1$. The detailed steps in Procedure PE1 are:

Procedure PE1

1. Solution set, S = ∅.

2. **(FT is k rule)**

For k= $1, \cdots, \left\lfloor \frac{M+1}{2} \right\rfloor - 1$ {
$v_{k+1} = v_{k+2} = \cdots = v_n = 1$
$v_1 = v_2 = \cdots = v_k = \text{int}((n-k-1)/k)$
$m = n - k - 1 - k \times v_1$
i = 0.
while (i < m){
i=i+1.
$v_i = v_i + 1$.
}
S = S + $(v_1, \cdots, v_n)$
}

3. (k-sites form majority rule)

```
For k= 2, ···, ⌊(M+1)/2⌋ −1{
    v_{k+1} = v_{k+2} = ··· = v_n = 1
    v_1 = v_2 = ··· = v_k = int ((n−k+1)/k)
      if (n−k+1 > k×v_1){
        for i=1, ···,k
            v_i=v_i+1;
    m = k×v_1 − n + k − 1
    i = 0.
    while (i < m){
        i=i+1.
        v_{k+i} = v_{k+i} + 1.
            }
                    }
S = S + (v_1, ···, v_n)
                    }
```

4. The set, S contains all the vote combinations produced by PE1.

In procedure **PE2**, all vote combinations with a sum less than or equal to $2n+1$ are produced from Procedure SE of Section 8.1 and included in set S.

Procedure PE2

1. Using Procedure SE, generate all vote combinations such that $N_{\max} = 2n+1$

Now we turn to describe the complete algorithms to solve models 1 and 2. Algorithms PE-FT1 and PE-FT2 apply to model 1, while algorithms PE-AV1 and PE-AV2 to model 2. The main steps of algorithm PE-FT1 are listed first and, then, a detailed description of each step follows.

Algorithm PE-FT1

1. Renumber sites such that:

$$C_1 \le C_2 \le \cdots \le C_j \le \cdots \le C_n$$

where

$$C_j = \sum_i c_{ij}(q_i + u_i)$$

2. Produce Set S from Procedure PE1.

3. Set Min-Cost = ∞

4. (Main section)

```
For each vote combination, V̄ within S {
    if (FAILURE-TOLERANCE(V̄) ≥ cut-off) {
        COMPUTE_COST(V̄)
        If (cost < Min-cost) {
        Min-Cost = Cost
        Min-Assignment = V̄(M, FT)
                }
                                    }
                                        }
```

5. The best solution is given by Min-assignment, and the minimum cost is contained in Min-cost.

FAILURE-TOLERANCE($\bar{V}$)

1. Find smallest ft such that:

$$\sum_{i=1}^{ft} v_i \geq q$$

COMPUTE_COST($\bar{V}$)

1. For each site i, determine the set of other sites to be included in i's quorum, Q_i by:

   (a) first ordering all other sites j in ascending order of $\frac{c_{ij}}{v_j}$, and

   (b) then choosing the first k sites from this sequence such that:

$$v_i + \sum_k v_k \geq q$$

2. The cost of assignment, $\bar{V}$ is:

$$\sum_i \sum_{k \in Q_i} (q_i + u_i) \times c_{ik}$$

Algorithm PE-FT2

This algorithm is identical to PE-FT1 except that in step 2, procedure PE2 is used to produce set S.

Now we explain the steps in Algorithms PE-FT1 and PE-FT2. In step 1, we determine a "greedy" ordering of sites based on C_i, the total communications cost if there was only one file copy and this was assigned to site i. Therefore, the site with the smallest C_i is renumbered as site 1, etc. Since the votes in any vote combination occur in a non-increasing sequence, this "greedy" reordering of sites ensures that the site with the lowest C_i gets the highest vote. In step 2, the initial minimum cost is set to ∞. In step 3, we produce a set of vote combinations to be evaluated using Procedure PE1 or PE2 in algorithms PE-FT1 and PE-FT2 respectively. Then, Procedure **COMPUTE_COST** is used to compute the cost of those assignments which satisfy the failure tolerance cut-off. Finally, from all the alternatives examined, the minimum cost solution is chosen as the best solution.

The **COMPUTE_COST** procedure is required to determine the total communications cost for assignment, $\bar{V}$ given the query and update volumes (q_i and u_i respectively) of each site in addition to the c_{ij}'s. The major step is to determine the set of sites, Q_i which site i must communicate with in order to form a quorum. This sub-problem can be formulated as a knapsack problem [BUDN77] and the technique described in step 1 is the solution method for it. Once the Q_i's are known, the total cost is computed as in step 2. Now we shall turn to describe algorithms PE-AV1 and PE-AV2 which minimize communications cost subject to an availability cut-off.

The steps in algorithm PE-AV1 are:

Algorithm PE-AV1

1. Renumber sites such that:

$$C_1 \leq C_2 \leq \cdots \leq C_j \leq \cdots \leq C_n ,$$

where

$$C_j = \sum_i c_{ij}(q_i + u_i)$$

2. Produce Set S from Procedure PE1.

3. Set Min-Cost = ∞

4. (Main section)

```
For each combination, V̄ in set S {
  if (MAX_AVAIL(V̄) >= cut-off){
    While (AVAIL(V̄) < cut-off)
      REASSIGN(V̄).
    COMPUTE_COST(V̄).
    if (Cost < Min-cost){
       Min-Cost = Cost
       Min-Assignment = V̄
          }
                 }
               }
```

5. The best assignment is given by Min-assignment, and the minimum cost is contained in Min-cost.

MAX_AVAIL($\bar{V}$)

1. Arrange the sites in descending order of p_i.

2. Assign votes to sites in the same order as they occur in $\bar{V}$.

3. Use algorithm AVAIL to compute availability for this assignment.

REASSIGN($\bar{V}$)

1. Rearrange $\bar{V}$ by swapping v_i and v_j where i and j are two sites ($i < j$) such that:

   1. $(v_i - v_j) \times (P_i - P_j) < 0$.
   2. $C_i - C_j$ minimum over all such pairs of sites, (i, j).

COMPUTE_COST($\bar{V}$)

1. For each site i, determine the set of other sites to be included in i's quorum, Q_i by:

   (a) first ordering all other sites j in ascending order of $\frac{c_{ij}}{v_j}$, and
   (b) then choosing the first k sites from this sequence such that:

$$v_i + \sum_k v_k \geq q$$

2. The cost of the assignment is computed as:

$$\sum_{i} \sum_{k \in Q_i} (q_i + u_i) \times c_{ik}$$

Algorithm PE-AV2

This is similar to algorithm PE-AV1 with the exception that in step 2, Procedure PE2 is used instead of Procedure PE1 in order to generate set S.

The major difference between algorithms PE-AV1 and PE-FT1 lies in step 4. While for a given vote combination, the failure tolerance is independent of a specific assignment, the availability does depend upon the specific assignment of votes to sites. For instance, by assigning higher votes to sites with higher probabilities, it is possible to increase the availability. Therefore, for each vote combination, we initially assign votes to sites in a greedy manner, and then iteratively reassign the same set of votes (again, greedily) to improve availability until the desired cut-off is reached. The minimum cost assignment which satisfies the availability requirement is chosen as the best solution.

The steps in Procedure MAX-AVAIL and REASSIGN are listed above. Procedure MAX-AVAIL determines the maximum availability that can be achieved from a given vote combination. This is done by first assigning votes to sites in descending order of p_i, i.e., the site with the highest p_i is given the largest vote, etc. Then, the availability for this assignment is computed.

Procedure REASSIGN is used to reassign votes so as to increase the overall availability for a given combination of votes. The reassignment is done by first identifying all pairs of sites (i, j) such that on swapping their respective votes, the site with the higher reliability would get a larger vote in the pair. Among all such pairs, the pair with the smallest $C_i - C_j$ is chosen, where C_i and C_j are the total communications costs if there were just one copy at sites i and j respectively. This is a greedy way of minimizing the additional cost that would be incurred as a result of this reassignment. Other greedy strategies for doing this reassignment were also considered but discarded because they were not as effective as the current one.

10. Computational Results

In order to evaluate our algorithms, we implemented algorithms SE-AV, PE-AV1, and PE-AV2 for 7 and 9-site networks. For the 7-site network, 3 experiments were conducted with different sets of inter-site communications costs in each case. These costs were generated from 3 uniform distributions: U(1,1), U(1,5), U(1,10), and the results of the corresponding experiments are shown in Tables 5, 6, and 7 respectively. The columns of each table show the minimum total communications cost and the corresponding assignment chosen by each algorithm. The figures in parentheses next to the cost figures indicate the percentage by which Algorithms PE-AV1 and PE-AV2 perform poorly as compared to Algorithm SE-AV. Each row of these tables corresponds to a different availability cut-off value, and for each cut-off, the best cost and the vote assignment computed by the three algorithms has been shown. The last row of each table, shows the average percentage over the range of AV values by which Algorithms PE-AV1 and PE-AV2 are inferior to Algorithm SE-AV. The reliability vector and the traffic volumes at each site are also indicated in each table.

In the 7-site implementation of Algorithm SE-AV, all 133 combinations listed in Table 4 were considered, and all permutations of each combination were evaluated. It is evident from Table 5 that when the inter-site communications cost, c_{ij} is constant for all pairs of sites, both heuristics perform extremely well (within 11% of the "optimal" on an average). On the other hand, when the c_{ij}'s vary over an order of magnitude range (see Table 7), then Algorithm PE-AV2 on an average comes within 15% of Algorithm SE-AV, while the corresponding metric for Algorithm PE-AV1 is 69%. Clearly, the performance of the heuristics do deteriorate with an increasing range of c_{ij}'s and this deterioration is more in the case of Algorithm PE-AV1.

For the 9-site network, Table 8 shows the results for the case in which c_{ij}'s are drawn from a uniform distribution: U(1,10). This distribution was chosen out of the 3 mentioned above to obtain worst case results. In the implementation of Algorithm SE-AV, only 191 combinations out

of the 7603 listed in Table 4 were considered. These correspond to all combinations where the sum of votes is less than or equal to 23. This was done in order to keep the computational effort within a reasonable bound (few hours on a 3-4 MIPS computer). Table 8 shows that on an average the cost of Algorithm PE-AV2 came within 19% of the "optimal" solution, while the corresponding metric for Algorithm PE-AV1 was 53%.

11. Static versus Dynamic Algorithms

So far in this paper our focus has been on the optimization of static algorithms. As mentioned earlier, the second type of QC algorithms are the dynamic ones. Although a detailed comparison between static and dynamic algorithms is outside the scope of this work, we shall perform a comparison here between an optimized static algorithm for Model 2 and the available copies algorithm [BERN84] which is a dynamic algorithm.

AV	Algo. SE-AV	Algo. PE-AV2	Algo. PE-AV1
0.93	39 (5,1,1,1,1,1,1)	39 (0%) (1,5,1,1,1,1,1)	39 (0%) (1,5,1,1,1,1,1)
0.94	39 (5,1,1,1,1,1,1)	39 (0%) (1,5,1,1,1,1,1)	39 (0%) (1,5,1,1,1,1,1)
0.95	44 (7,2,1,2,1,2,2)	49 (11%) (1,2,1,6,1,2,2)	61 (39%) (1,1,1,3,1,1,3)
0.96	44 (9,3,1,3,1,3,3)	49 (11%) (2,6,1,2,1,1,2)	61 (39%) (1,1,1,3,1,1,3)
0.97	53 (7,3,3,3,1,1,1)	57 (8%) (2,5,1,2,1,1,1)	61 (13%) (1,1,1,3,1,1,3)
0.98	62 (1,3,1,3,1,1,1)	62 (0%) (1,3,1,3,1,1,1)	62 (0%) (1,3,1,3,1,1,1)
0.99	78 (2,2,1,1,1,1,1)	78 (0%) (2,1,1,2,1,1,1)	78 (0%) (2,1,1,2,1,1,1)
avg%	0%	4%	11%

Table 5: Communications Cost and vote assignment
for various desired values of AV (7 sites, c_{ij}'s from Uniform(1,1)).
$\overline{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84)
Traffic Volumes = (5,7,4,9,1,5,8)

AV	Algo. SE-AV	Algo. PE-AV2	Algo. PE-AV1
0.93	95 (1,1,5,1,1,1,1)	95 (0%) (1,1,5,1,1,1,1)	95 (0%) (1,1,5,1,1,1,1)
0.94	96 (1,2,1,7,2,2,2)	115 (20%) (1,5,1,1,1,1,1)	115 (20%) (1,5,1,1,1,1,1)
0.95	101 (1,2,7,2,1,2,2)	119 (18%) (1,2,2,6,2,1,1)	179 (77%) (1,1,3,3,1,1,1)
0.96	111 (2,4,10,4,2,1,4)	122 (10%) (1,1,2,5,2,1,1)	179 (61%) (1,1,3,3,1,1,1)
0.97	119 (3,3,2,6,1,1,1)	140 (18%) (1,1,1,3,1,1,1)	179 (50%) (1,1,3,3,1,1,1)
0.98	128 (2,5,7,4,3,1,1)	145 (13%) (1,2,2,4,2,1,1)	179 (40%) (1,1,3,3,1,1,1)
0.99	167 (4,4,3,1,2,2,3)	181 (8%) (2,3,2,2,2,1,1)	214 (28%) (1,2,2,2,2,1,1)
avg%	0%	11%	35%

Table 6: Communications Cost and vote assignment
for various desired values of AV (7 sites, c_{ij}'s from Uniform(1,5)).
$\bar{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84)
Traffic Volumes = (5,7,4,9,1,5,8)

AV	Algo. SE-AV	Algo. PE-AV2	Algo. PE-AV1
0.93	101 (1,2,2,7,1,2,2)	125 (24%) (1,2,2,6,1,2,1)	160 (58%) (1,1,5,1,1,1,1)
0.94	101 (1,2,2,7,1,2,2)	125 (24%) (1,2,2,6,1,2,1)	220 (118%) (1,5,1,1,1,1,1)
0.95	107 (1,2,1,6,1,2,2)	125 (17%) (1,2,2,6,1,2,1)	236 (121%) (1,1,3,3,1,1,1)
0.96	120 (2,4,3,9,2,1,4)	140 (17%) (1,2,2,5,1,1,1)	236 (97%) (1,1,3,3,1,1,1)
0.97	140 (1,3,3,5,1,1,1)	140 (0%) (1,3,3,5,1,1,1)	236 (69%) (1,1,3,3,1,1,1)
0.98	146 (2,4,2,5,1,1,2)	188 (29%) (1,2,4,4,1,2,1)	236 (62%) (1,1,3,3,1,1,1)
0.99	240 (2,2,3,3,2,1,2)	267 (11%) (2,2,3,3,1,2,2)	302 (26%) (1,2,2,2,1,2,1)
avg%	0%	15%	69%

Table 7: Communications Cost and vote assignment
for various desired values of AV (7 sites, c_{ij}'s from Uniform(1,10)).
$\overline{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84)
Traffic Volumes = (5,7,4,9,1,5,8)

AV	Algo. SE-AV	Algo. PE-AV2	Algo. PE-AV1
0.93	179 (1,2,2,2,1,9,2,1,1)	225 (26%) (1,2,1,2,1,7,1,1,1)	259 (45%) (1,7,1,1,1,1,1,1,1)
0.95	215 (2,3,1,1,2,9,3,1,1)	241 (12%) (1,1,1,2,1,6,1,1,1)	386 (80%) (1,1,1,4,1,4,1,1,1)
0.97	252 (3,4,1,1,2,8,1,2,1)	310 (23%) (1,3,1,2,1,6,1,1,1)	386 (53%) (1,1,1,4,1,4,1,1,1)
0.99	366 (3,4,2,2,1,5,2,1,1)	418 (14%) (2,3,2,5,1,2,1,2,1)	484 (32%) (1,3,2,3,1,3,1,2,1)
avg%	0%	19%	53%

Table 8: Communications cost and vote assignment
for various desired values of AV (9 sites, c_{ij}'s from Uniform(1,10)).
$\overline{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84,0.88,0.83)
Traffic Volumes = (5,7,4,9,1,5,8,6,3)

Algorithm SE was used to optimize the vote assignment for the 7-site network discussed in Section 10, and the same 3 uniform distributions were used as before to generate the unit inter-site communications cost. Again, the availability cut-off was varied from 0.93 to 0.99 in intervals of 0.01. At each cut-off value, the corresponding communications cost was computed using Algorithm SE-AV. The results of these computations are presented in Table 9. The total traffic volumes and the reliability of each site, p_i are also indicated. (Note that Table 9 was actually constructed by extracting the first column from Tables 5, 6, and 7, and combining the 3 extracted columns together).

To make a comparison against the available copies algorithm we need to compute the communications cost and availability again. In the static case, the read and write quorums were required to be equal as discussed in Section 3.2; therefore, the total communications cost does not depend on the write ratio, ρ_w. In the dynamic case, however, this is not true, and the communications cost would vary for different values of ρ_w. In Table 10, we have computed the communications cost for different values of ρ_w. The 3 columns correspond to the 3 different distributions for the unit inter-site communications cost as in Table 9. The rows of Table 10 correspond to dif-

AV	U(1,1)	U(1,5)	U(1,10)
0.93	39	95	101
0.94	39	96	101
0.95	44	101	107
0.96	44	111	120
0.97	53	119	140
0.98	62	128	146
0.99	78	167	240

Table 9: Static Algorithm: Minimum Communications Cost for 3 cost distributions (7 sites)
$\bar{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84)
Total Traffic Volumes = (5,7,4,9,1,5,8)

ferent values of ρ_w. The total traffic volumes were the same as in Table 9; however, the read and write traffic components were varied depending upon ρ_w.

The availability in the dynamic case is computed differently than in the static case. We now turn to describe our method for performing this computation. In the available copies algorithm, transactions running at the time a site fails or recovers may have to be aborted, and restarted. This will result in a time delay during which the system will be unavailable. We call this time interval a reconfiguration interval, t_{recon}. It is assumed that each site has a mean time to failure (MTTF) and a mean time to repair (MTTR). For simplicity, all sites are assigned the same value of MTTF and also of MTTR. To make the analysis tractable, it is reasonable to further assume that each site will fail and recover once on the average within a MTTF + MTTR cycle, and at each failure and recovery point, the system will be unavailable for t_{recon} length of time. Thus, if availability is defined as the fraction of time for which the system is available during one cycle, it may be estimated as:

$$1-\frac{2\times t_{recon}\times n}{MTTF+MTTR}$$

ρ_w	U(1,1)	U(1,5)	U(1,10)
0.05	12	34	73
0.1	24	75	141
0.15	36	108	199
0.2	48	139	256
0.3	72	213	422
0.4	96	282	502
0.5	120	364	675

Table 10: Dynamic Algorithm: Communications Cost versus ρ_w for 3 cost distributions (7 sites)
$\overline{P}$ = (0.91,0.90,0.89,0.87,0.86,0.85,0.84)
Total Traffic Volumes = (5,7,4,9,1,5,8)
AV = 0.98

In order to make the comparison against the optimized static algorithm, MTTF was set at 6 hours, while MTTR was set to 44 minutes for each site. This translates to a reliability of 0.874 in probabilistic terms which is the average p_i for the 7 sites in Table 9. Using a t_{recon} value of 30 seconds, and setting n to 7, the availability is 0.98.

Several interesting conclusions may be drawn from the results in Tables 9 and 10. First, if cost minimization is the main objective, then the dynamic algorithm is superior when ρ_w is below a cut-off. However, this cut-off becomes smaller as the range of variation of c_{ij} increases. For instance, if the distribution chosen is U(1,1), then the cut-off is 0.15, while for distributions U(1,5) and U(1,10), the cut-off reduces to 0.10 and 0.05, respectively. This means that as the variation in c_{ij} increases, the dynamic algorithm becomes less attractive if cost minimization is the main objective.

On the other hand, if the cost of the dynamic algorithm is compared against its static counterpart which gives the same availability, then the above ρ_w cut-offs increase to 0.25, 0.2, and 0.1 respectively. Therefore, the ρ_w space within which the dynamic algorithm does better becomes larger. Clearly, ρ_w is a critical factor in choosing between the static and dynamic algorithms.

Secondly, while different static vote assignments lead to various values of availability and communications cost, in the dynamic case there is one availability value, 0.98. Table 9 shows that the static method can give a higher availability of 0.99. Therefore, if availability maximization is the main goal then the static technique seems to outperform the dynamic one. Of course, t_{recon} and n are critical parameters in computing the availability from the formula above, and if these can be lowered, then availability would naturally increase.

12. Summary and Future Research

The vote assignment problem is important because the assignment of votes to sites greatly affects the transaction overhead and availability. In this paper, a general framework for understanding and analyzing this problem was introduced. The main objective of this framework is

communications cost minimization either in an unconstrained manner or subject to a certain reliability constraint. Two basic optimization models were developed. In the first model, the reliability constraint was failure tolerance while in the second one it was availability. In the general models, no restriction is imposed on the votes or the unit inter-site communications costs. However, if such conditions are added then some of the special cases that arise are easier to solve than the general model. Such special cases were also included in our framework and techniques for solving them described.

It was shown that the general vote assignment problem has exponential complexity. Moreover, no efficient solution procedure that would run in reasonable time is known. (Complete enumeration is an obvious solution method, though clearly not a feasible one). A semi-exhaustive algorithm and two partial enumeration algorithms to solve these problems were discussed in detail and also implemented. These algorithms employ efficient techniques for computing the availability of a vote assignment. The signature concept was used to prune the exponential space of vote combinations and to generate only non-equivalent combinations.

The computational results in Section 10 validate greedy algorithms as an effective technique for vote assignment. Although these algorithms deteriorate as the range over which c_{ij}'s are varied increases, they compare favorably with semi-exhaustive algorithms overall, and lead to several orders of magnitude savings in computational effort.

Lastly, the optimized static algorithm was compared against the available copies method, a dynamic algorithm. It was found that no one type of algorithm uniformly dominates the other. Ranges over which each type of algorithm does better were determined. Dynamic algorithms were better for a small value of the write ratio, ρ_w, and low variability in the inter-site communications cost. On the other hand, static algorithms were better if the goal was to maximize availability. Also, if the goal was to minimize cost, then the range of ρ_w values over which the dynamic algorithm does better is very small. Another factor that should be considered is that

static algorithms are simpler and much easier to implement than dynamic algorithms.

This study also shows that a more detailed comparison of static algorithms against dynamic algorithms would be a very useful exercise. In this paper, the treatment of dynamic algorithms has been restricted to just one type, the available copies method. Future work is anticipated to evaluate other dynamic algorithms, perhaps using more elaborate models. For instance, the reconfiguration time should also include the time to reoptimize the vote assignment.

Finally, yet another area for future research is to establish bounds on the sub-optimality of the semi-exhaustive algorithms. Basically, our algorithm is semi-exhaustive, not completely exhaustive, for two reasons. First, we generate new vote combinations by adding an incremental number of votes to the set of combinations obtained at the most recent stage, not to all the combinations obtained so far. This could lead to some combinations being missed. Secondly, we choose an arbitrarily high number for a stopping rule, or stop if no new combinations are found for a gap of 25 votes. It would naturally be desirable if it could be proven rigorously that there is an upper limit for the total number of votes beyond which no new combinations will be found. A method for overcoming these two drawbacks could conceivably lead to a smart, completely exhaustive algorithm. A different approach to establishing sub-optimality bounds on the semi-exhaustive algorithms would be to use some kind of an upper bound analysis technique.

Acknowledgements.

We thank Prof. Michael Stonebraker for his suggestions on comparing static algorithms against dynamic ones.

References

[BARB86] Barabara, D., Garcia-Molina, H., and Spauster, A., "Protocols for Dynamic Vote Reassignment", Technical Report, Department of Computer Science, Princeton University, May 1986.

[BERN84] Bernstein, P.A., and Goodman, N., "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases", ACM Transactions on Database Systems 9(4), pp 596-615, December 1984.

[BERN87] Bernstein, P., Hadzilacos, V., and Goodman, N., Concurrency Control and Recovery in Database Systems, Addison Wesley Publishing Co., 1987.

[BUDN77] Budnick, Frank S., Mojena, R., and Vollmann, T.E., "Principles of Operations Research for Management", Richard D. Irwin, Inc., 1977.

[DAVI85] Davidson, S. B., Garcia-Molina, H., and Skeen, D., "Consistency in Partitioned Networks", ACM Computing Surveys 17(3), pp 341-370, September 1985.

[EAGE81] Eager, D.L., "Robust Concurrency Control in Distributed Databases", Technical Report CSRG #135, Computer Systems Research Group, University of Toronto, October 1981.

[EAGE83] Eager, D.L., and Sevcik, K.C., "Achieving Robustness in Distributed Database Systems", ACM Trans. Database Syst. 8(3), pp 354 - 381, September 1983.

[ELAB85] El Abbadi, A., Skeen, D., and Cristian, F., "An Efficient, Fault-Tolerant Protocol for Replicated Data Management", Proc. 4th ACM SIGACT-SIGMOD Symp, on Principles of Database Systems, pp 215 - 228, Portland, Oregon, March 1985.

[GARC84] Garcia-Molina, H., and Barbara, D., "Optimizing the Reliability Provided by Voting Mechanisms", Proc. 4th International Conference on Distributed Computing Systems, pp 340-346, May 1984.

[GARC85] Garcia-Molina, H., and Barbara, D., "How to Assign Votes in a Distributed System", Journal of ACM, Vol. 32, No. 4, pp 841-860, October 1985.

[GIFF79] Gifford, D.K., "Weighted Voting for Replicated Data", Proc. 7th ACM SIGOPS Symp. on operating Systems Principles, pp 150 - 159, Pacific Grove, CA, December 1979.

[HERL87] Herlihy, M., "Dynamic Quorum Adjustment for Partitional Data", ACM Trans. on Database Systems, Vol 12, No 2, pp 170-194, June 1987.

[JAJO87] Jajodia, S. and Mutchler, D., "Dynamic Voting", Proc. 1987 ACM SIGMOD, pp 227-238, San Francisco, CA, May 1987.

[KUMA88] Kumar, A., and Segev, A., "Optimizing Voting-Type Algorithms for Replicated Data", Lecture Notes in Computer Science, Vol 303, J.W. Schmidt, S. Ceri and M. Missekoff (eds.), pp 428-442, Springer-Verlag, March 1988.

[STON79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed Ingres", IEEE Transactions on Software Engineering 3(3), pp 188-194, May 1979.

[THOM79] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control", ACM Trans. on Database Systems 4(2), pp 180-209, June 1979.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720