A Static Pessimistic Scheme for Handling Replicated Databases

Jian Tang

Department of Computer Science
Memorial University of Newfoundland
St.John's, Newfoundland A1C 5S7
Canada

N. Natarajan

Department of Computer Science
Pennsylvania State University
University Park, PA 16802

Abstract

A replicated database system may partition into isolated groups in the presence of node and link failures. When the system has partitioned, a *pessimistic scheme* maintains availability and consistency of replicated data by ensuring that updates occur in at most one group. A pessimistic scheme is called a *static scheme* if these *distinguished* groups are determined only by the membership of different groups in the partitioned system. In this paper, we present a new static scheme that is more powerful than voting. In this scheme, the set of distinguished groups, called an *acceptance set*, is chosen at design time. To commit an update, a node checks if its enclosing group is a member of this acceptance set. Using an encoding scheme for groups, this check is implemented very efficiently. Another merit of the proposed scheme is that the problem of determining an *optimal* acceptance set is formulated as a *sparse 0-1 linear programming problem.* Hence, the optimization problem can be handled using the very rich class of existing techniques for solving such problems. Based on our experiments, we feel that this optimization approach is feasible for systems containing up to 10 nodes (copies).

1. Introduction

A replicated database system may sometimes partition into isolated groups due to node and link failures. There are two approaches for maintaining data availability in the presence of such partitions [DGS85]. In one class of *replica control schemes*, called *optimistic schemes*, all partition groups are permitted to update a replicated data. When groups merge, inconsistencies are detected and resolved by undoing the effects of some already completed transactions. Thus, they are based on the premise that *committed* transactions can be undone or compensated for. Whether this assumption is valid or not depends on the application. In the other class of schemes, called *pessimistic schemes*, when the system is in a partitioned state, updates on a replicated data are allowed in only one partition group, thus preventing inconsistencies. The group chosen is called the *distinguished group* for that partition state. Different groups can be distinguished groups for different partition states. Also, in the same state, different data items may have different distinguished groups.

A pessimistic scheme can be classified as *static* or *dynamic* depending on the manner in which the distinguished group for a partition state is chosen. In a static scheme, the partition structure, i.e., the membership of different partition groups, completely determines whether a group is distinguished or not [Gi79]. In a dynamic scheme, some additional information, such as currency of copies, is also used for determining whether a group is distinguished [JM87]. In general, a dynamic scheme provides higher availability than a static scheme. But, a static scheme has several merits: it is conceptually simple; information overhead is less; recovery overhead is minimal and no need to run "catch up" protocols.

No static scheme (or pessimistic scheme) can guarantee that a distinguished group always exists. Data availability is lost when the network partitions such that no group is the distinguished group. Thus, we are faced with the problem of *optimizing* the availability provided by a static scheme. The problem can be stated as follows: given node and link failure probabilities, choose a set of distinguished groups such that the probability that a distinguished group exists when a partition occurs is maximum. This is a computationally hard problem since the number of candidate sets of distinguished groups is double exponential in n, where n is the number of nodes in the system.

In this paper, we propose a new static scheme which has several merits. In our scheme, each node maintains a table, called an *acceptance set*, which is the set of possible distinguished groups. Any two groups in an acceptance set intersect each other. When a transaction arrives at a node, the node determines its enclosing group and checks if that group is a member of the acceptance set. If so, it executes the transaction; otherwise, it rejects it. In general, different acceptance sets may be chosen for different data items, and for each item two different sets can be used: one for read and another for write. To simplify exposition, in the following we assume that there is only one acceptance set for the system. This static scheme is more powerful than voting [Gi79] in the sense that there are acceptance sets that have no equivalent vote assignment. In this respect, it is similar to the *coterie scheme* proposed by Garcia-Molina and Barbara [GB84, GB85]. Compared to the coterie scheme, our scheme is more efficient to implement. We encode groups to numbers and check for distinguished group involves search for a number in a set of numbers. For a 10 node (i.e., copy) system, the acceptance set contains at most 1024 numbers.

Another merit of our static scheme is that we formulate the problem of determining an optimal acceptance set for a system as a *0 -1 sparse linear programming problem.* No such formulation is possible for the coterie scheme. Our experiments reveal that this optimization approach is feasible for up to 10 nodes. For larger systems, we need some heuristics to generate a better scheme from a given scheme. Although we have developed such a transformation procedure, we do not describe it here [TN88].

This paper is organized as follows. In the next section, we describe the related work on static schemes. In section 3, we describe the new static scheme. In section 4, we discuss an efficient implementation of this static scheme and illustrate how this scheme can be used for serializing updates on replicated files. In section 5, we compare this scheme with the coterie scheme. In section 6, we discuss the optimization problem formulation and some experimental results. We then conclude by summarizing the major results.

2. Related work on static schemes

One of the most popular static schemes is *voting* scheme [Gi79, Th79]. In a voting scheme, the set of distinguished groups is not maintained explicitly, but determined implicitly. A node determines that its enclosing group is a distinguished group if the sum of the votes of *reachable* nodes exceeds a threshold. Thresholds for read (RT) and write (WT) operations are chosen such that sum of RT and WT exceeds total votes, and WT is more than half of total votes.

Garcia Molina and Barbara have proposed a static scheme that is more powerful than voting [GB84, GB85]. They introduce the notion of a *coterie* to identify the set of distinguished groups. A coterie is a set of groups in which every pair of groups has a node in common. Further, if a group is in a coterie, then any proper superset of that group must not be in the coterie. The coterie scheme works as follows. A coterie is chosen for the system. This coterie must be a *nondominated coterie (ND coterie).* A coterie A is a ND coterie if there is no other coterie B such that every group in A is a superset of some group in B. A node determines that its enclosing group is the distinguished group if the group is a superset of (or identical to) some group in the chosen coterie. The coterie scheme is more powerful than voting scheme. For systems with more than five nodes, there are coteries that do not have equivalent vote assignments. However, the coterie scheme has the disadvantage that its implementation is not very efficient. If the chosen coterie has m groups, then $m/2$ set operations are needed to determine if a group is distinguished. Since m could be as large as 2^{n-1}, where n is the number of nodes in the system, this overhead is not negligible.

Garcia-Molina and Barbara have considered also the problem of determining an *optimal coterie* for a system, given the failure probabilities of individual nodes. They have considered an enumeration approach; i.e., enumerate all candidate ND coteries, compute the availability of each coterie, and choose the best. It turns out that enumerating all ND coteries of a system is impractical for large systems (i.e., more than five nodes) because of the large number of ND coteries. They show that this number is lower bounded by $2^{2^{cn}}$ for some constant c, where n is the number of nodes in the distributed system. For large systems, they suggest a partial enumeration approach, and have proposed a transformation method for enumerating a subset of candidate ND coteries. However, as they have pointed out, even the partial enumeration method generates too many coteries (still double exponential in n). Thus, their approach seems appropriate only for systems of up to 5 nodes.

More recently, Tong and Kain have proposed algorithms for optimal vote assignment [TK88] for voting systems. For systems with perfect links, they propose an algorithm with linear execution time that generates vote assignments that are close to optimal (within 0.3%). For systems with imperfect links, they have proposed heuristics that generate vote assignments that are close to optimal within 1.1%. Although these are interesting algorithms, they have the limitation that they produce only optimal voting schemes, but not optimal *static schemes*. (Recall that there are acceptance sets and coteries that do not have equivalent vote assignments).

3. A new static scheme

3.1 Network states and acceptance sets

In section 1, we described the basic idea of our new static scheme. In brief, the set of distinguished groups for a system, called the acceptance set, is determined at system design time. A node determines that its enclosing group is a distinguished group if the group is a member of the acceptance set. In this section, we present a more precise description of the scheme. First, we introduce a notion, *network state*, to model the current network state. It contains information about partitions and status of each node. We then define an acceptance set, and then describe some properties of acceptance sets.

Definition: Let U be a set of nodes and W be the power set of U with the empty set excluded. That is, $W = 2^U - \phi$. The *universe of network states* is defined as $P = \{p: p \subseteq W \ \& \ \forall r \forall s(r \in p \ \& \ s \in p \Rightarrow r \cap s = \phi)\}$.

Each element of P is called a *network state* (or *state*), and it is a set of disjoint subsets of U. Each element in a state denotes a *partition group* (or *group*). A failed node is not included in any group. For example, suppose U = {1,2,3}. Then, each of the following is a network state.

{{1,2,3}} -- A state with one group
{{1,2}, {3}} -- A state with two groups
{{1},{2},{3}} -- A state with three groups
{{1},{2}} -- A state in which node 3 has failed

An acceptance set is a set of groups, and each element in the set is a distinguished group. Given that in a pessimistic scheme, at most one group can be a distinguished group in a state, the following definition is natural.

Definition: An acceptance set A is a set of subsets of U such that $\forall p(p \in P \Rightarrow |p \cap A| \leq 1)$.

Although this definition is adequate to characterize an acceptance set, it is not very convenient to use. To check if a set of groups is an acceptance set, we have to consider all possible network states. We give below a theorem that simplifies the check for acceptance sets.

Theorem 1: A set of groups S is an acceptance set if and only if any two groups in S intersect each other.

Proof:
If: Assume that any two groups in S intersect each other. Since every network state contains pairwise disjoint groups, it cannot have more than one group that is in S. Thus S is an acceptance set.
Only if: Assume S is an acceptance set. If there are two groups, say r and s, such that $r \in S$ and $s \in S$ and $r \cap s = \phi$, then there is a state $\{r,s\}$ such that $|\{r,s\} \cap S| = 2$. A contradiction to S being an acceptance set. □

Example 1:

Assume U = {1,2,3} and S_1 = {{1},{1,2},{1,3}}. It is easy to check that S_1 is an acceptance set. Now, consider S_2 = {{1},{2}}. It is not an acceptance set.

3.2 Maximal acceptance sets

If an acceptance set S is a superset of another acceptance set R, we prefer S over R for the following reason. There are at least as many states in which a distinguished group exists when S is used as there are when R is used. Hence, we should always use *maximal acceptance sets*, as defined below.

Definition: An acceptance set S is a *maximal acceptance set* if $\forall r \exists p(r \notin S \ \& \ r \subseteq U \ \& \ p \in P \ \& \ |(S \cup \{r\}) \cap p| = 2)$.

Thus, an acceptance set is not a maximal acceptance set if it has a superset which is also an acceptance set. The following theorem simplifies the check for a maximal acceptance set.

Theorem 2: If S is a maximal acceptance set, then $\forall s \forall t(s \in S \ \& \ s \subseteq t \Rightarrow t \in S)$.

Proof:

Let *s,t* be such that $s \in S \ \& \ s \subseteq t$. Since *s* intersects every group in S, so does *t*. Thus $S \cup \{t\}$ is an acceptance set. Since S is maximal, we have $t \in S$. □

Example 2:

S_1 of Example 1 is not a maximal acceptance set since it does not include the group {1,2,3}.

The following theorem gives the necessary and sufficient condition for a maximal acceptance set.

Theorem 3: An acceptance set S is a maximal acceptance set if and only if $\forall r(r \subseteq U \ \& \ r \notin S \Rightarrow \exists t(t \in S \ \& \ t \cap r = \phi))$.

Proof:

If: Assume that the condition is true. Let $r \subseteq U \ \& \ r \notin S$. Since *r* does not intersect some group in S, by Theorem 1, $S \cup \{r\}$ is not an acceptance set. Thus S is a maximal acceptance set.

Only if: Assume that S is a maximal acceptance set. Let $r \subseteq U \ \& \ r \notin S$. If $\forall t(t \in S \Rightarrow t \cap r \neq \phi)$, then by Theorem 1, $S \cup \{r\}$ is an acceptance set. Thus S is not a maximal acceptance set. A contradiction. □

4. An implementation of this static scheme

In this section, we describe an efficient implementation of the proposed static scheme. We encode groups to integers and use this to determine efficiently if a group is distinguished. We shall see in section 6 that this encoding scheme is useful also in formulating the optimization problem as an integer linear programming problem. To make the implementation ideas concrete, we describe the implementation in the context of serializing updates on replicated files.

4.1. Group encoding

In principle, any encoding scheme that establishes a one to one map between W, the set of groups, and the set of integers {1,2,...,|W|} is adequate. In the following, we introduce an encoding scheme which is conceptually simple and easy to implement.

Suppose the system is composed of *n* nodes. We identify each node using an *unique* integer in the range 1 to *n*. Thus we can denote a system composed of *n* nodes as U = {1,2,...,*n*}. We encode a group *r* to an integer *i* as follows. Let $b_n b_{n-1} \ldots b_1$ be the binary representation of *i*. Now, b_j of *i* is 1 if *j*th node is in group *r*; otherwise, b_j is 0. For example, suppose U = {1,2,3,4}. Then, group {1,3} is encoded as 5 and group {1,2,4} is encoded as 11. Thus, each group is encoded to a number in the range 1 to 2^n-1. (Note that 0 is not included since the empty set is not a valid partition group). With this encoding, an acceptance set is just a list of integers. Note that for a *n* node system, an acceptance set can contain at most 2^{n-1} numbers, since if a group is in an acceptance set, its complement cannot be in that acceptance set.

4.2. An example: Updates on replicated files

In this section, we illustrate, using the group encoding suggested above, how this static scheme can be implemented to ensure that at most one partition group performs updates on replicated files. To simplify description, we assume full replication. When an update of a file is initiated at a node, it is processed in the following manner.

First, the local file copy is copied into a temporary file. During transaction execution, updates are made only to the temporary file. When the transaction attempts to commit, it is checked if the enclosing group of the node is the distinguished group. If yes, the temporary file is propagated to every reachable node; otherwise, the transaction is aborted. Thus distinguishability check is integrated with *two phase commit*. We ignore here concurrency control problems, but they can be handled rather easily.

In the following, we outline the actions of the *coordinator* node (i.e., node at which the transaction is initiated), and *cohort* nodes (other nodes in the reachable set of the coordinator) at commit point. S denotes the encoded acceptance set (a list of numbers) used by the system. *Cohort(j)* denotes the cohort node whose number is *j*.

Coordinator:

1. Send a request to all reachable nodes to indicate the intention to commit.
2. Wait until a reply (containing an integer) is received from every reachable node.
3. Compute the sum M of all integers received.
4. If M belongs to S then commit the update and propagate the update to all reachable nodes; otherwise, reject the update and notify all reachable nodes.

Cohort(j):

1. Upon receiving the request from the coordinator, send 2^{j-1} as reply;
2. Wait until the updated file or abort notification is received.
3. If the updated file is received, install it as the local copy.

It is easy to see that, except for step 4 of the coordinator, the scheme is as efficient as a voting scheme. Step 4 could be implemented efficiently using an efficient search mechanism, e.g., binary search.

5. Relationships with the coterie scheme

The objective of this section is to show that the notions of acceptance set and coterie are equally powerful. Hence our scheme also is more powerful than voting [GB85]. To make the paper self-contained, we introduce some basic concepts of the coterie scheme. For details, see [GB85].

Definition: Let U be the set of nodes that compose the system. A set of groups S is a *coterie* under U iff

i. $G \in S$ implies that $G \neq \phi$, and $G \subseteq U$.
ii. (intersection property) If $G, H \in S$, then G and H must have at least one common node.
iii.(minimality property) There are no $G, H \in S$ such that $G \subset H$.

Definition: Let R, S be coteries under U. R *dominates* S iff $R \neq S$ and, for each $H \in S$, there is a $G \in R$ such that $G \subseteq H$.

Definition: A coterie S under U is *dominated* iff there is another coterie under U which dominates S. If there is no such coterie, then S is *nondominated* (ND).

Theorem 4: (From [GB85]). A coterie, C, is dominated iff there exists a group, *g*, such that

a. *g* is not a superset of any group in C, and
b. *g* intersects with every group in C.

The following two theorems establish a one to one correspondence between the set of acceptance sets and the set of ND coteries.

Theorem 5: Let C be a ND coterie and $J(C) = \{s: s \supseteq r \text{ for some } r \in C\}$. Then J(C) is a maximal acceptance set.

Proof:
The intersection property of J(C) directly follows from that of C. By Theorem 1, J(C) is an acceptance set. To show that J(C) is maximal, consider a group, *g*, $g \notin J(C)$. By construction of J(C), $g \notin C$. Further, *g* is not a superset of any group in C. Since C is a ND coterie, by Theorem 4, it follows that
$\exists t(t \in C \ \& \ g \cap t = \phi)$. Since every member of C is also a member of J(C), it follows that $\exists t(t \in J(C) \ \& \ g \cap t = \phi)$. Since this is true for any $g \notin J(C)$, by Theorem 3, J(C) is maximal. □

Theorem 6: Let M be a maximal acceptance set and $K(M) = \{c: c \in M \ \& \ \forall t(t \subset c \Rightarrow t \notin M)\}$. Then K(M) is a ND coterie.

Proof:
From the way in which K(M) is constructed, it follows that K(M) satisfies both intersection property and minimality property. Hence, it is a coterie. We show that K(M) is a ND coterie by

contradiction. Suppose K(M) is dominated. Then, there exists a group r, $r \notin$ K(M), that satisfies properties (a) and (b) of Theorem 4. We prove that such a group does not exist.

Since r is a group, either $r \in$ M or $r \notin$ M. Suppose, $r \in$ M. Since $r \notin$ K(M), it follows that there exists a group s such that $s \subset r$ and $s \in$ K(M). Thus, r does not satisfy property (a). Suppose $r \notin$ M. Then, there exists a group s such that $s \in$ M and $s \cap r = \phi$. Let u be such that $u \subseteq s$ and $\forall t (t \subset u \Rightarrow t \notin$ M). From the definition of K(M), $u \in$ K(M). Since $s \cap r = \phi$ and $u \subseteq s$, it follows that $u \cap r = \phi$. Thus, r does not satisfy property (b). Hence, K(M) is a ND coterie. □

It is easy to see that for any maximal acceptance set, J(K(M)) = M and for any ND coterie C, K(J(C)) = C.

6. Determining an optimal acceptance set

We now consider the problem of determining an optimal acceptance set for a system given the failure probabilities of its nodes and links. To do this, we first need to define precisely the notion of optimality. For a state $p \in$ P, we define T(p) as the probability that p will occur. This can be computed from node and link probabilities. Now, for a group g, let $F(g) = \sum_{g \in p} T(p)$. That is, F($g$) is the probability that g is a group in some network state. The availability of an acceptance set is the probability that a distinguished group exists in the system (in any arbitrary network state) when that acceptance set is used. Thus, the availability of an acceptance set S is $\sum_{g \in S} F(g)$. An acceptance set is optimal if it has the highest availability among all possible acceptance sets for the system.

We now formulate the problem of determining an optimal acceptance set as a 0-1 integer linear programming problem. As we will discuss later, this optimization approach seems feasible for up to 10 nodes (i.e., copies).

6.1. Formulation as an integer linear programming problem

The constraint part of the formulation ensures that any solution obtained satisfies the constraint on acceptance sets. That is, in any network state, at most one group in the state belongs to the acceptance set. However, it is not necessary to consider every state of the system to check if a set of groups is an acceptance set. It is sufficient to consider only states in which every node belongs to some group. (We formally prove this later). We call such states *full-node* states. Recall that, in a state, a node that has failed does not belong to any group. Thus, a full-node state is a state in which no node has failed. For example, for a 3 node system, {{1,2},{3}} is a full-node state, while {{1},{2}} is not. Thus, a set of groups S is an acceptance set if, in every full-node state, at most one group in the state belongs to S.

We now formulate the optimization problem. Let the system be composed of n nodes numbered 1 to n. Using the group encoding scheme suggested earlier, every group is assigned a unique number in the range 1 to 2^n-1. We represent an acceptance set by a $k \times 1$ vector of 0's and 1's, where $k = 2^n$-1. A 1 in the ith position of this vector means that the ith group is included in the acceptance set, and a 0 means that it is not. Let **A** denote the optimal acceptance set that we seek to determine. A full-node state is represented by a vector of length k, where a 1 in the ith position means that the ith group is a group in the state, and a 0 means that it is not. Let **Q** be a $m \times k$ matrix, each row of which is a 0-1 vector denoting one possible full-node state, and m is the number of full-node states. Let **E** be a $1 \times k$ vector whose jth element denotes the probability that the jth group is a group in some state (not necessarily a full-node state). Let **1** be $m \times 1$ unit vector. Now, the optimization problem is:

maximize **EA** subject to the constraint $\mathbf{QA} \leq \mathbf{1}$.

The meaning of this formulation is fairly obvious. Note that **EA** is the availability of the acceptance set denoted by **A**. **QA** gives, for each state, the number of groups in the state that belong to the acceptance set denoted by **A**.

The following theorem tells us that the solution to the above integer programming problem gives an optimal acceptance set.

Theorem 7: Let **A** be a solution to the formulated integer programming problem. Let M be the acceptance set denoted by **A**, i.e., the corresponding set of distinguished groups. Then, M is an optimal acceptance set.

Proof:
We first prove that M is an acceptance set. Let y be any state. Suppose, y is a full-node state. Let q be the corresponding row for y in Q. Since **A** is a solution, $|y \cap M| \leq 1$. Now, suppose y is not a full-node state. We construct a full-node state y_1 as follows. Let g be the set of nodes not included in y. Let $y_1 = y \cup \{g\}$. Now, $|y_1 \cap M| \leq 1$. Since, y is a subset of y_1, it follows that $|y \cap M| \leq 1$. Hence, M is an acceptance set.

We now prove that M is optimal. Let M' be an acceptance set. Let **A'** be the 0-1 vector representation of M'. Since M' is an acceptance set, we have
$\forall p$(p is a full-node state => $|p \cap M'| \leq 1$). But this is to say, $\mathbf{QA'} \leq \mathbf{1}$. From the optimization problem formulation, it follows that $\mathbf{EA'} \leq \mathbf{EA}$. Thus, the availability of M' is not greater than that of M. □

In general, to calculate the vector **E**, we need to know the topology of the network and the failure probability of each node and link. Also, we have to consider all possible network states. However, some simplifications are possible. For instance, the topology may rule out formation of some groups. Suppose G is the graph that represents the initial network, i.e., the network in which all links and nodes are alive. For any group of nodes, g, the graph of g is defined as the subgraph of G with vertex set g. Let j be the encoded number for g. If the graph of g is not connected, then j the element of **E** is 0. Further simplification is possible if we neglect link failures. (Of course, the acceptance now obtained is not really optimal, but will be close to optimal). If we consider only node failures, then elements of **E** can be calculated using a very simple formula as follows. Let g be a group, and let j be its encoding. For any node i, let p_i be the probability that i is alive. Let $V(g)$ be a set of nodes defined as $V(g) = \{r | r \notin g \ \& \ r$ is a neighbour of some node in $g\}$. If the graph of g is connected, then

$$e_j = \prod_{i \in g} p_i \prod_{i \in V(g)} (1-p_i)$$

Otherwise, e_j is zero.

Example 3:

Consider a ring network of three nodes, numbered 1 to 3. For nodes 1,2 and 3, the probability that each of them does not fail is 0.7,0.8, and 0.9 respectively. Links do not fail. We wish to determine an optimal acceptance set for this system.

For this system, the possible groups are:
{1}, {2}, {3}, {1,2}, {2,3}, {1,3}, {1,2,3}
We encode each group as follows:
{1} -> 1; {2} -> 2; {1,2} -> 3; {3} -> 4;
{1,3} -> 5; {2,3} -> 6; {1,2,3} ->7;
There are five full-node states, and they are ordered as follows:
{{1},{2},{3}}, {{1,2},{3}}, {{1,3},{2}},
{{2,3},{1}}, {{1,2,3}}
Now, matrix **Q** has 5 rows and 7 columns. Consider the first row of **Q**. It corresponds to the first full-node state. This state is composed of groups numbered 1,2, and 4. Thus, the first row of **Q** is [1 1 0 1 0 0 0]. Other rows can be constructed in a similar way. The complete matrix **Q** is:

$$\mathbf{Q} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Vector **E** has seven elements (one for each group) and each element is computed using the formula given earlier in this section.

E = [0.014, 0.024, 0.056, 0.054, 0.126, 0.216, 0.504]

Let **A** a 0-1 variable column vector of length 7. Let **1** be the unit vector of length 5. Now, the optimization problem is:

maximize **EA** subject to the constraint $\mathbf{QA} \leq \mathbf{1}$.

The acceptance set corresponding to the solution vector **A** is an optimal acceptance set for the system.

6.2. The feasibility of this optimization approach

For a n node system, the length of vector **A** is $2^n - 1$, and this number is also the number of columns in matrix **Q**. The number of rows in **Q** is the number of full-node states. For a n node system, the number of full-node states, N(n), is given by the following recurrence relation:

N(1) = 1;
For $n > 1$,

$$N(n) = N(n-1) + \binom{n-1}{1} N(n-2) + \binom{n-1}{2} N(n-3) + \ldots + \binom{n-1}{n-2} N(1) + 1$$

In Table 1, we list, for systems consisting up to 10 nodes, the length of the solution vector **A**, and number of rows in matrix **Q**. (The number of columns in **Q** is the same as the length of **A**.)

The values in the right most column are the maximum percentage of non-zero elements in matrix **Q**. It can be seen from the table that when $n \leq 6$, the length of **A** and the size of matrix **Q** are relatively small. When $n = 7$ or 8, the length of **A** is still moderate, while the number of rows in **Q** increases almost quadratically. However, the percentage of non-zero elements in **Q** becomes very small. All these cases fall into the category of large scale sparse 0 - 1 linear programming problems which can be solved very efficiently by current techniques [CJP83, JKS85, Pa79]. When $n = 9$ or 10, the size of **Q** becomes very large. Since at this time **Q** also becomes extremely sparse, these cases can still be handled, probably less efficiently.

To determine the feasibility of this optimization approach, we performed some experiments using a linear programming package *Lindo* on a VAX 11/780. We formulated the optimization problem for systems containing 5, 6, 7, and 8 nodes. In each case, we tried 5 different sets of node failure probabilities. Links were assumed to be perfect. The results are given in Table 2.

The experimental results show that the optimization approach is quite feasible and attractive for systems up to 8 nodes. Because of the limitations of the programming package, we could not test for systems of 9 and 10 nodes. However, we believe that they can be handled by other packages commercially available. The experimental results do not include the time for creating matrix **Q** and vector **E**. We have already discussed how calculation of **E** can be simplified. Matrix **Q** can be generated in a straight forward way. First, enumerate all full-node states and then create corresponding rows of **Q**. This is rather inefficient. In the appendix, we describe an efficient procedure for constructing **Q**. It takes advantage of the group encoding scheme, and does not explicitly generate full-node states at all. The only operations involved are arithmetic operations.

For $n \geq 11$, the linear programming approach may not be feasible due to the large problem size. In this case, we need to use a heuristic approach to obtain an optimal acceptance set. We have developed some heuristics. However, we will not discuss them here, and they are described elsewhere [TN88].

7. Conclusion

We presented a new static pessimistic scheme for maintaining availability and consistency of replicated data in a distributed system. In this scheme, the set of distinguished groups for a system, called the acceptance set, is determined at design time. Each node maintains this set. When a transaction (commit) is initiated at a node, the node first determines its reachable set of nodes. It commits the transaction only if the enclosing group is a distinguished group contained in the acceptance set. This scheme is more powerful than voting and equally powerful as the coterie scheme. By encoding groups to numbers, the scheme is implemented efficiently. The check for distinguished groups reduces to the problem of searching for a number in a set. The coterie scheme does not have this merit.

Another merit of the proposed static scheme is that the problem of determining an optimal scheme for a system is formulated a 0-1 linear integer programming problem. This optimization approach seems feasible for up to 10 nodes. We have experimental evidence that demonstrates that this approach is very attractive.

Acknowledgment

We thank Prof.P.Pardalos for his comments and suggestions on the linear programming problem formulation.

References

[CJP83]
H.Crowder,E.L.Johnson,M.Padberg, Solving Large-Scale Zero-One Linear Programming Problems, *Operations Research*, Vol.31, No.5, September-October 1983, pp.803-834.

[DGS85]
S.B.Davidson,H.Garcia-Molina,D.Skeen, Consistency in partitioned networks, *ACM Computing Surveys*, Vol.17, No.3, September 1985, pp.341-370.

[GB84]
H.Garcia-Molina, D.Barbara, Optimizing the reliability provided by voting mechanisms, Proc. Fourth International Conference on Distributed Computing Systems, October 1984, pp. 340-346.

[GB85]
H.Garcia-Molina, D.Barbara, How to assign votes in a distributed system, *Journal of the*

ACM, Vol.32, No.4, October 1985, pp.841-860.

[Gi79]
D.K.Gifford, Weighted voting for replicated data, Proc. of Seventh ACM Symposium on Operating Systems Principles, December 1979, pp.150-162.

[JKS85]
E.L. Johnson, M.M. Kostreva, U.H.Suhl, Solving 0 - 1 integer Programming Problems Arising from Large Scale Planning Models, *Operations Research*, Vol.33, No.4, 1985 pp.803-819.

[JM87]
S.Jajodia,D.Mutchler, Dynamic voting, Proc. of SIGMOD 87, pp.227-238.

[Pa79]
M.W.Padberg, Covering, Packing and Knapsack Problems, *Annals of Discrete Mathematics*, 4 (1979), 265-287.

[Th79]
R.H.Thomas, A majority consensus approach to concurrency control for multiple copy databases, *ACM Transactions on Database Systems*, Vol.4, No.2, June 1979, pp.180-209.

[TK88]
Z.Tong,R.Y.Kain, Vote assignments in weighted voting mechanisms, Proc. of Seventh Symposium on Reliable Distributed Systems, October 1988, pp.138-143.

[TN88]
J.Tang,N.Natarajan, A formal model for pessimistic schemes for managing replicated databases, Tech. Report, Dept. of Computer Science, Pennsylvania State University.

Appendix

An Algorithm for constructing matrix Q

The algorithm generates **Q** without explicitly enumerating all full-node states. Using the group encoding scheme discussed in the paper, the algorithm generates **Q** using only arithmetic operations. The algorithm is based on the idea that full-node states of a network can be generated in a recursive manner. To accomplish this, we generalize the notion of full-node states introduced in the paper to any arbitrary group of nodes of a network. For any group, g, a full-node state on g is a set of disjoint subsets of g such that each node in g appears in some subset.

Suppose, g is a singleton set. Then, the set of full-node states on g, called, FN(g), is the set $\{\{g\}\}$. Otherwise, FN(g) can be generated recursively as follows. Since each node is assigned a unique *id*, we let $g = \{a,b,...,k\}$, where $id(a) < id(b) < ... < id(k)$. Now, let $G = \{g_1,g_2,...,g_m\}$, where each g_i is a subset of g containing node a. Since each group is encoded to a number, without loss of generality, let $e(g_1) < e(g_2) < e(g_3) < ... < e(g_m)$, where $e(g_i)$ is the encoded number of g_i. Now, FN(g) can be generated as follows:

$$FN(g) = \{\{g_1\} \cup r: r \in FN(g - g_1)\} \cup \{\{g_2\} \cup r: r \in FN(g - g_2)\} \cup \{\{g_3\} \cup r: r \in FN(g - g_3)\} ... \cup \{\{g_m\} \cup r: r \in FN(g - g_m)\}$$

This recursive scheme is based on the fact that any full-node state on g must contain an enclosing group of a, and two full-node states are different if the enclosing group of a is different in both.

Based on this recursive scheme, procedure *Construct* given below constructs the portion of **Q** that corresponds to full-node states for a given group. It accepts two input parameters, g and p, where g is the encoded number of a group, and p is a row number. Given these two, *Construct* fills each row of **Q**, starting from the pth row, with 1's in some columns such that the filled columns of each row denotes a full-node state on g. The procedure returns a result parameter, c, that gives the number of rows filled, i.e., number of full-node states on g.

Central to the procedure is a *while* loop. The jth iteration of the loop constructs the portion of **Q** that corresponds to the jth component of FN(g) given above, i.e., $\{\{g_j\} \cup r: r \in FN(g - g_j)\}$. During jth iteration, the variable u contains the encoded value of g_j, and w contains the encoded value of $g - g_j$. The matrix portion for FN(g) is constructed in two steps. First, the portion corresponding to FN($g - g_j$) is constructed using a recursive call. Then, the uth column of each row filled by the recursive call is set to 1.

In the procedure, we assume a function, *fact*, defined as follows. *fact(x)* gives the largest factor of x that is a power of 2; e.g., *fact*(12) = 4, *fact*(15) = 1, and *fact*(24) = 8.

The complete matrix **Q** is generated by procedure *Generate* that calls *Construct* with g set to denote the entire network.

```
procedure Construct(g,p:integer; var c:integer);
 /* In the following, q_{i,j} denotes the element at the ith row and jth column of Q */
 var f,r,s,t,u,v,w: integer;
 begin
   if g = 0 then  c := 1
    else
      begin
       c := 0;
       r := fact(g);   /* r denotes the group g_1 described above */
       u := r;   w := g - u;     t := 0;
       while w ≥ 0 do
          begin
           /* generate the matrix for FN(g - g_j) */
           Construct(w,p,s);
           /* Set the uth column in rows p to p+s-1 to 1 */
           for i := p to p+s-1 do  q_{i,u} := 1;
           c := c + s;   p := p + s;
           if w ≤ 0 then exit loop;
           /* The following five statements set u to denote the next
              group in G, i.e., g_{j+1}. w is set to denote g-g_{j+1} */
            f := fact(w);   v := t mod f;  t := t + f - v;  u := r + t;  w := g - u
          end while
    end
 end Construct;

procedure  Generate;
 var c: integer;
 begin
    Construct(2^n - 1,1,c);  /* n is the number of nodes in the network */
 end Generate;
```

# of nodes	length of A	# of rows in Q	maximum percentage of non- zero entries in Q
3	7	5	28.5%
4	15	15	20.0%
5	31	52	12.9%
6	63	203	7.9%
7	127	877	4.7%
8	255	4140	2.7%
9	511	21147	1.6%
10	1023	100603	0.9%

Table 1. The optimization problem size for systems of up to 10 nodes

size of system	# of samples tested	machine used	maximum time spent
5	5	Vax11/780	10 seconds
6	5	Vax11/780	30 seconds
7	5	Vax11/780	2 minutes
8	5	Vax11/780	10 minutes

Table 2. Experimental results for systems of 5 to 8 nodes.