AN EFFICIENT MUTUAL EXCLUSION METHOD IN DISTRIBUTED SYSTEMS

HAO CHEN

An Efficient Mutual Exclusion Method in Distributed Systems

By

Hao Chen B.Sc.

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science

Department of Computer Science
Memorial University of Newfoundland
August, 1994

St. John's Newfoundland Canada

ISBN 0-315-96037-X

Abstract

Many operations in a distributed system require mutual exclusion to guarantee correctness. Quorum methods have been widely proposed for implementing mutual exclusion. Majority quorum consensus is the best known quorum method. It has the merit of simplicity, but may incur high message overhead. Tree algorithm is an efficient structured quorum method to the mutual exclusion problems. The quorums generated by a tree algorithm are smaller on the average than those by a majority quorum consensus. However, the tree algorithm enforces a highly biased treatment to the nodes at different levels. This affects its performance in a distributed system where the nodes have similar characteristics. We propose a new structured quorum method called triangular net quorum algorithm, which treats the nodes more evenly than the tree algorithm while preserving a satisfactory availability, as well as lowering average quorum size. We believe that this method is desirable for implementing mutual exclusion in a truly distributed system.

Acknowledgments

I wish to express my thanks to my supervisor Prof. Jian Tang for his guidance, interest, suggestions and patience during my studies at Memorial University of Newfoundland. I learned a lot from the various discussions with him. Without his supervision, it would be impossible to give this thesis its current quality.

I also thank the other professors who taught me at Memorial University. Prof. Tony Middleton, Prof. John Shieh, Prof. Krishnamurthy Vidyasankar and Prof. Xiaobu Yuan.

I am also very grateful to the Administrative staff who have helped in one way or another in the preparation of this thesis.

Special thanks are due to my fellow graduate students, in particular to Mr. Zhimin Shi, Mrs. Zhengqi Lu, Mrs. Yongmei Sun and Mr. Zhengjie Chen for their valuable comments and useful suggestions. I would also like to thank Dr. Siwei Lu, Michael Winter, Elaine Boone and Patricia Murphy for their help and assistance.

I thank those many others not named here who provided encouragement and assistance during my graduate studies.

This thesis is dedicated to

my grandmother

and my parents

for their support and encouragement throughout

the course of my education

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Mutual Exclusion in Distributed Systems

We describe a distributed system as a system which consists of a set of separate computers. These computers are linked by a computer network. The general-purpose distributed systems are designed to enable the individual computers of which they are composed to use shared resources in the network, providing computing facilities which are more flexible and widely-applicable than centralized computer systems. Users of a distributed system are given the impression that they are using a single, integrated computing facility, although the facility is actually provided by more than one computer and the computers may be in different locations.

Mutual exclusion problem arises when processes share resources. When processes running at different sites of a distributed system attempt to concurrently access a resource, and it is required that no more than one such process is allowed to access the resource at any given instant, a mutual exclusion algorithm must be run to enforce this requirement. Many operations in a distributed system require mutual exclusion to guarantee correctness. We term these operations restricted operations. Examples of restricted operations include updating on replicated data, naming of distributed objects, atomic commitment and other operations which require that a resource be allocated to a single process at a time.

Generally, distributed mutual exclusion is complicated not only due to the explicit message passing needed and the asynchrony in the system, but also due to the possibility of component failures during the algorithm execution. Therefore, mechanisms guaranteeing mutual exclusion should be both resilient and efficient. Resiliency usually implies high availability of resources in the case of failures, while efficiency implies low overhead incurred by performing restricted operations.

1.2 Basic Approaches

The issues relating to mutual exclusion have been studied extensively [4, 7, 8, 17, 37, 42]. Among the solutions suggested, the methods based on *quorums* have been widely accepted as an effective mechanism for implementing mutual exclusion. In a quorum method, a set of groups, called a quorum, is predefined either directly or indirectly. At any time, the execution of a restricted operation is allowed only if a quorum can be constructed by following some specific rules, such as a quorum should contain majority sites in a distributed system. In general, quorums must have an intersection property to ensure mutual exclusion. In [14], the authors study the general properties of a quorum set. A paradigm for optimizing the availability of quorum sets is proposed in [38]. Almost all of the quorum-based algorithms can be broadly divided into two classes:

1. Algorithms imposing logical structures on the topology of the distributed system, and

2. Algorithms imposing no such structures.

The majority voting [15, 39] is the best known quorum method which imposes no structure on the system. It assigns a number of votes to each node, and allows only those nodes that can collect a majority of votes to perform a restricted operation. Since, at any one time, there is only one majority in the system, mutual exclusion is guaranteed. The merit of the majority voting scheme is its simplicity; it is simple to understand and simple to implement. Its shortcoming is just that: it always requires a majority to make the operation succeed. This may incur a high communication overhead.

To reduce the communication cost, methods based on logical structure of the network have been proposed [4, 11, 20, 21, 25, 44]. In [25], the author associated with each site a set of sites and all such sets pairwise intersect. The sizes of these sets can be $\sqrt{n}$, compared with $\lceil \frac{n+1}{2} \rceil$ required by a majority quorum consensus. The drawback of this scheme is that it exhibits a very low availability when n gets larger. In [11, 21], the authors use 'grid' as the logical structure. In [20], the author uses a hierarchy to define quorum. Both methods are used mainly for synchronizing the read and write of replicated data. The idea of using tree as the logical structure for achieving mutual exclusion is suggested in [4]. This method has a very low cost in the best case. It is also easy to implement, since at runtime, each site only maintains a tree representing the logical structure required by the algorithm. However, the costs of the quorums are highly unevenly distributed in the quorum set for the tree algorithms. The nodes at the higher levels are heavily favored over those at the lower levels. Consequently, in a distributed system where the nodes have similar characteristics, its performance in terms of either availability or the cost will be affected. Therefore, how to implement a high availability and low communication cost mutual exclusion method is an important problem for the design of distributed systems.

1.3 Thesis Overview

In this thesis, we propose a new structured quorum method. It is based on a special logical structure, called *Triangular Net*. Our algorithm organizes the network nodes into a triangular net structure. Like a tree structure, it contains a number of levels and the nodes at different levels are associated by parent-child relationship. Unlike a tree structure, a child may have more than one parent. Because of this increased sharing, our algorithm possesses some desirable properties which a tree algorithm does not have. The goal of our algorithm is twofold. One is to preserve the logical clarity and simplicity as well as the essential properties for mutual exclusions, such as the intersection and minimality properties of quorums and non-dominance of the quorum set. The other is to diminish the discrepancy of the nodes at different levels in terms of the capabilities of forming a quorum in the hope of reducing the cost and enhancing availability.

The triangular net structure can also be extended to allow an internal node to have more than two children, and a child may also have more than two parents. The generalized triangular net structure is the basis for the generalized triangular net quorum algorithm.

Our algorithm is resilient to both site and communication failures. In comparison with other logical structure quorum methods, the triangular net quorum algorithm treats the nodes more evenly than the tree quorum algorithm does. In addition, our algorithm has a smaller average quorum size compared with tree algorithm and majority quorum consensus, and maintains a competitive availability.

This thesis is organized into six chapters. In Chapter 2, we present the distributed system model for mutual exclusion, and introduce the properties of a coterie (quorum set). In Chapter 3, we briefly introduce some existing quorum algorithms for mutual exclusion, especially for structured methods. In Chapter 4, we present our proposed triangular net method for mutual exclusion, prove its correctness and some important properties, such as minimality and non-dominance properties, and analyze its performance. In Chapter 5, we extend the triangular net structure to the more general case and develop the generalized triangular net quorum algorithm. We summarize the contribution of this thesis in the conclusion chapter.

Chapter 2

A System Model

In this chapter, we introduce a model for distributed systems and mutual exclusion. We describe the concepts of quorum sets (coterie) which are widely used for mutual exclusion. We organize this chapter as follows: In section 2.1, we present a distributed system model. In section 2.2, we introduce the theory of coteries.

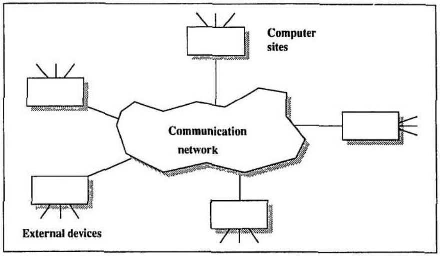


Figure 2.1: The general model of a distributed system

2.1 Model Description

A distributed system consists of a set of distinct sites and a communication network. Associated with each site is a unique site identifier. Each site may be a supercomputer, a workstation or a personal computer. Communication between different sites is done by sending messages through the communication network. The general structure is described in Figure 2.1.

Sites and links are the basic components in a distributed system. Due to *failures, or other exceptional reasons, the normal functions of a component may* be disrupted. Two states, *up* and *down* are used to simulate this phenomenon. At any instance of time, a component can be either up or down. A site that is up can send messages to and receive messages from the other sites and perform operations. We assume a down site simply stops functioning. This character is called *fail-stop*[34]. A link that is up can deliver messages between its two adjacent sites. Communication links may fail by crashing, or by failing to deliver messages. Combinations of such failures may lead to *partitioning failures* [13], where sites in a partition may communicate with each other, but no communication can occur between sites in different partitions.

The problem of mutual exclusion is one of the fundamental problems encountered in the design of distributed systems. Informally, the problem postulates the existence of a resource in the network, which may be accessed by a single process at a time. The execution of some operations, such as updating a replicated data in a distributed system, requires the participation of a group of sites. When this happens, the operation is first initiated at a site. We call the site where the operation is initiated the *coordinator* for the operation, and the other sites in the group *participants*. The coordinator must ask for the permission from all participants in the group before it is allowed to carry out the operation. If all participants grant the permission, the operation is performed, otherwise, the operation is rejected. We say that an operation requires mutual exclusion if any two disjoint groups of sites are disallowed to perform the operation in parallel. A natural way of ensuring mutual exclusion is to allow the operations to be performed only by a set of sites which pairwise intersect. Thus, in order to perform the operations, the coordinator must obtain the permissions from the participants which form a group of sites in the distributed system.

2.2 Properties of Coteries

The concept of intersecting groups captures the essence of mutual exclusion in distributed systems. In [14, 15, 19], the authors study this concept in detail. They proposed the notion of a *coterie*.

Definition: Let U be the set of sites that compose the system, and let a group g and h be a set of sites. Then, a coterie C under U is a set of groups, called quorums, which satisfy the following conditions.

- **The Intersection Property:** $\forall g, h \in C$, $g \cap h \neq \phi$.
- **The Minimality Property:** There are no g and $h \in C$ such that $g \subset h$.

Coteries can be used to develop quorum methods that guarantee mutual exclusion in distributed systems. Generally, a quorum method generates a coteric. When a process at a site wishes to perform a restricted operation, it must obtain permission from all sites in a quorum. Since any pair of quorums have at least one site in common, mutual exclusion can be guaranteed.

In terms of the likelihood that a site will get permission from a quorum, some coterie is more favorable than the others. Suppose a system consists of nodes a, b, c and d. Two coteries are defined as: $S = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$. $R = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}$. It is easy to see that a site can get permission from a quorum in S only if it can get permission fi .n a quorum in R. In other words, a site is more likely to be able to perform restricted operations under R than it is under S. Thus, intuitively, R is more favorable to us than S. This motivates the following concept[14]:

Definition: Let U be the set of sites that compose the system. Let C, D be coteries under U. C *dominates* D iff $C \neq D$ and for each $H \in D$, there is a $G \in C$ such $G \subseteq H$. (We say that G is the group that dominates H.)

Definition: A coterie S under U is *dominated* iff there is another coterie under U which dominates S. Otherwise, S is *nondominated* (ND).

Unfortunately, there exists an exponential number of ND coteries for any systems of size n, and no polynomial time algorithm is known at this time to check if a coterie is nondominated[14].

Two major criteria are widely used to judge the quality of a quorum-based method.

- *Quorum size*: the number of sites in a quorum. Since the number of messages needed is directly proportional to the size of the quorums, the smaller the quorum size, the lower the communication costs are and the better the system performance is.
- *Availability*: the probability of forming a quorum successfully by the quorum-based method. The higher the availability, the higher the chance that the quorum can be constructed. Thus, high availability is always preferred.

Generally, there is a tradeoff between the availability and the average quorum size[22]. Many previous works have concentrated on the unconstrained maximization of availability, disregarding communications costs which are directly related to the quorum size. Recently, some algorithms have been proposed to exploit a logical structuring of network site to reduce the communication cost while keeping reasonable high availability. In the next chapter, we will introduce some quorum-based mutual exclusion methods, especially for structured quorum methods.

Chapter 3

Quorum-Based Mutual Exclusion Methods

In this chapter, we present a survey of the existing mutual exclusion algorithms. All these algorithms implicitly or explicitly construct a class of ND coterie. For each one, we will describe how it defines the quorum. To make the idea clear, we will use simple examples whenever possible. We will also discuss the major advantages and disadvantages of each algorithm. The details of each method can be found in the relevant literature. This chapter is organized as follows: In section 3.1, we introduce majority consensus algorithm which is the best known method for mutual exclusion. In section 3.2, we present the $\sqrt{N}$ method. In section 3.3, we introduce binary tree quorum algorithm. In section 3.4, we will introduce several generalized structured quorum methods which are used to reach mutual exclusion for replica control.

3.1 Majority Consensus Method

Thomas[39] presented a very simple and elegant scheme to achieve mutual exclusion in a distributed system. In order to attain mutual exclusion, a site must obtain permission from a majority of sites in the distributed system. Since there can be only one majority at any instant, mutual exclusion is achieved.

The merit of the majority consensus algorithm is its simplicity, simple to understand and simple to implement. The majority quorum algorithm is robust and resilient to both site and communication failures. It provides the maximum availability in the voting methods[7]. Its shortcoming is that it always requires a majority to make the operation succeed. This may incur a high communication overhead.

A simple generalization to this method is proposed by Gifford[15]. In the paper, each site is assigned a non-negative vote, and a quorum consists of any set of sites with a majority of votes. It is more flexible when used in replica control.

3.2 *Sqrt N* Method

Maekawa[25] proposed a method to implement distributed mutual exclusion by imposing a logical structure on the network. In this scheme, a set of sites is associated with each site, and this set has a nonempty intersection with all sets corresponding to the other sites. The rule for constructing these sets is based on the structure of finite projective planes[1]. A process must obtain permission from all sites in the set associated with its home site before it can perform a restricted operation. Since the set intersects with every other set of sites, mutual exclusion is guaranteed.

For example, suppose a system consists of seven sites. We now list the sets associated with each site generated by this method. In the following, S_i denotes a set of sites associated with site i. We have S_1={1,2,3}, S_4={1,4,5}, S_6={1,6,7}, S_2={2,4,6}, S_5={2,5,7}, S_7={3,4,7}, S_3={3,5,6}.

The advantage of this method is that the size of each group in the coterie is roughly $\sqrt{N}$, where N is the number of sites in the network. Hence, a process needs to communicate with $\sqrt{N}$ sites to obtain permission for mutual exclusion. Thus, this method significantly reduces the overhead of achieving mutual exclusion compared with majority quorum consensus. However, it turns out that this method has a very low availability. It has been proven that as the number of nodes n approaches ∞, the availability approaches 0.

3.3 Binary Tree Method

Agrawal and El Abbadi[4] proposed a new class of ND coteries based on complete binary trees. They assume that the sites are logically organized into a complete binary tree. That is, if k is the level of the tree, then it has $2^k - 1$ sites.

The algorithm for constructing a quorum for a binary tree can informally be described as follows. A quorum is constructed by selecting any path starting from the root and ending with any of the leaves. If successful, this set of sites constitutes a quorum. If a path cannot be constructed due to the inaccessibility of a site, c, residing on a failed or inaccessible site (due to partitioning failures), then the algorithm must substitute for that site with two paths -- both of which start with the children of site c and terminate with leaves. Note that each path

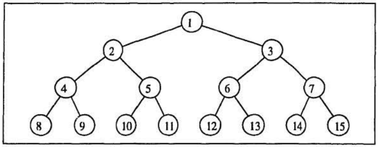


Figure 3.1: An example of 15 nodes organized as a binary tree

must terminate with a leaf; hence, if the last site in the path is inaccessible, the operation must be aborted. In paper [4], it is shown that any two quorums constructed using the above algorithm must have at least one site in common. The control mechanism is similar to the $\sqrt{N}$ algorithm.

As an example, consider the four-level binary tree in Figure 3.1. Following the algorithm, if no failures have occurred, then we can construct quorums such as {1,2,4,8}, {1,2,4,9}, {1,2,5,10}, etc. If the root is inaccessible(due to site failures or network partitioning), some other quorums, such as {2,4,8,3,6,12}, {2,4,8,3,7,14}, {2,4,9,3,6,12} can be constructed. If site 1, 2 and 3 are all inaccessible, then {4,8,5,10,6,12,7,14}, {4,8,5,11,6,13,7,14} can be constructed as quorums. If site 1,2,3,4,5,6 and 7 are inaccessible, then all the leaf sites construct a quorum.

The advantage of this algorithm is that it has very low communication cost in the best case, where only $\lceil log_2 n \rceil$ sites are needed to form a quorum. It is resilient to both site and communication failures, and it is also easy to implement. However, in the tree quorum algorithm, sites at the higher levels bear more weight than those at the lower levels when constructing a quorum. This affects its average case behavior.

3.4 Mutual Exclusion for Replica Control

As we have mentioned before, the mutual exclusion methods can be generalized to the problem of replica control in replicated database systems. Generally, a replicated database consists of a set of *objects*, where each object is implemented by a set of *copies* stored at different sites. Each copy has a version number, which is incremented whenever the copy is written. A user executes operations on the database by issuing a transaction[10], which is a set of partially ordered *read* and *write* operations. In a replicated database, a replica control method should ensure that the different copies of an object appear to the user as a single nonreplicated object. When restricted operations are performed on the replicated copies, the replica control is required. The only difference between normal mutual exclusion and replica control is that replica control distinguishes the types of restricted operations when it defines quorums. Specifically, for each object, it is associated with a *read quorum* and *write quorum*. A read operation is executed by accessing copies that constitute a read quorum. A write operation is executed by writing the copies in a write quorum.

The read quorums and write quorums are defined as sets of data copies that satisfy the following constraints.

1. **Write-write Intersection Constraint.** If g, h are two write quorums, then $g \cap h \neq \phi$.

2. **Read-write Intersection Constraint.** If g is a write quorum and h is a read quorum, then $g \cap h \neq \phi$.

These two constraints guarantee that any two conflicting operations (read and write operations or write and write operations) access at least one common data copy.

The difference among the existing quorum-based replicated data control methods is the way they define the read quorums and write quorums. In the next section, we will introduce some of the replica control methods which are influential in the database community.

3.4.1 Grid Method

Cheung, Ammar and Ahamad proposed the grid method in [11]. They consider a set of sites which are logically arranged into a grid. Each site stores a copy of a data object. Version numbers are used to identify the current copy of data. As an example, suppose 16 data copies are organized as a 4X4 grid, as shown in Figure 3.2.

The read quorum is defined as a C-cover of sites, which is a set of sites that contains one site from each column of the grid. The write quorum is defined as a set of sites that contains a column of sites and a C-cover of sites. Here, the size of read quorum is smaller than the write quorum size because people normally believe reading will be predominant. The write operation is required to synchronize with both write operations and read operations. Synchronization with read operations is accomplished by locking a column, and synchronization with write operations by locking a C-cover. For example, in Figure 3.2, {1,6,7,12} is a read quorum and {1,5,9,13,6,11,16} is a write quorum.

The advantage of this method is that the read quorum size and write quorum

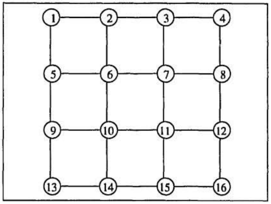


Figure 3.2: An example of 16 data copies organized as a 4X4 grid

size are smaller than voting methods. When a grid becomes square, both quorum sizes are $O(\sqrt{N})$, which is the smallest possible quorum size for both read quorum and write quorum for fully distributed replica control methods[45]. However, the disadvantage of the grid method is that the write availability decreases as N increases. This is because the write quorum contains a column of sites. The probability that all the data copies in a column are available is $1-(1-p^{\sqrt{N}})^{\sqrt{N}}$, here, p is the probability of a site which is up. We can see that for all $p < 1$, when $N \to \infty$, $p^{\sqrt{N}} \to 0$, $(1-p^{\sqrt{N}})^{\sqrt{N}} \to 1$ and $1-(1-p^{\sqrt{N}})^{\sqrt{N}} \to 0$. Thus the write availability $\to 0$ as $N \to \infty$.

3.4.2 Generalized Tree Method

Agrawal and El Abbadi generalized the mutual exclusion method[4] to manage replicated data[5]. Given a set of n copies of an object x, they are logically organized into a tree of height h, and degree d, i.e., the number of children of each node. An example of 13 data copies organized as a three-level ternary tree

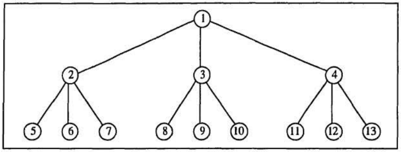


Figure 3.3: An example of 13 data copies organized as a ternary tree

is shown in Figure 3.3.

The read quorums and write quorums are defined recursively. The quorum constructed is called a tree quorum of length l and width w and will be denoted by the pair $< l, w >$. A read quorum is donated as $< l_r, w_r >$, a write quorum is donated as $< l_w, w_w >$, here, $l_r + l_w > l$, and $w_r + w_w > w$. Both read and quorum are constructed as the same way. We describe the method as follows. The method tries to construct a quorum by selecting the root and w children of the root, and for each node selected, w of its children, and so on, for depth l. If there is no failure, it forms a tree quorum of height l and degree w. However, if some node is inaccessible due to failure at depth h' from the root while constructing this tree quorum, then the node is replaced recursively by w tree quorums of height $l - h'$ starting from the children of the inaccessible node. The recursion terminates successfully when the length of the quorum to be constructed is zero. The algorithm fails to construct a quorum if the length of the quorum exceeds the height of the remaining subtree.

Consider a replicated object with thirteen copies. We superimpose a ternary tree of height 3 on the copies as illustrated in Figure 3.3, with the sites numbered as shown. In this case $d = 3$ and $h = 3$. We now construct tree quorums of length 1 and width 2. In the best case, the quorum need only contain the root. However, as a result of the failure of the root, a tree quorum of dimensions $< 1, 2 >$ can be formed from any majority(two) of the root's children, i.e., {2,3} or {2,4} or{3,4}. If a majority or more of the root's children have failed, then each such copy can be replaced by a majority of its children. Hence, if copies 1, 2, and 3 are inaccessible, then a quorum can be formed from copy 4 and a majority of either copy 2 or copy 3's children, e.g., the sets {4,5,6} and {4,8,10} form quorums.

The tree method has the advantage of a small quorum size. The smallest read quorum size is 1 and write quorum size could be as small as $O(N^{0.63})$ when the tree is a ternary tree. However, the write availability of the tree method is not better than the availability of a non-replicated data object.

3.4.3 Hierarchical Quorum Consensus Method

Kumar presented the hierarchical quorum consensus algorithm[20]. This algorithm is based on logically organizing a set of objects into a multi-level tree(of depth n) with the root at level 0. The physical objects are stored in the leaves of this tree, or at level n. Thus, the root has l_1 subobjects(or logical objects) at level 1, and each level 1 logical object in turn has l_2 subobjects at level 2, and so on. Consequently, there are l_n level n (physical) objects for each level $n - 1$ object. Therefore, the total number of physical objects is $l_1 * l_2 * ... * l_n$. As an example, Figure 3.4 is a two-level hierarchy where the root has 3 level 1 subobjects, each containing 3 physical objects.

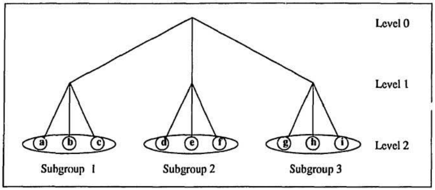


Figure 3.4: An example of 9 objects organized into three subgroups

Associated with each level i are two numbers r_i and w_i. For all r_i and w_i, the following two inequalities hold.

1. $r_i + w_i > l_i$, and

2. $2 * w_i > l_i$.

The read and write quorums are defined recursively. A read (write) quorum at level i is assembled by gathering $r_i(w_i)$ subobjects at level $i+1$, until the leaf level is reached.

Take the two-level tree in Figure 3.4 for an example. Suppose $r_1 = 1$, $w_1 = 3$, $r_2 = 2$ and $w_2 = 3$. Then, {a,b} and {g,h} are examples of the read quorums. {a,b,d,e,g,i} is an example of the write quorum. It is shown in [20] that the smallest quorum size is $O(N^{0.63})$. This happens when $l_i = 3$, $r_i = 2$, and $w_i = 2$, for all $0 \leq i \leq n$.

One advantage of this method is that the quorum size and availability can be controlled by adjusting l_i, r_i and w_i and n. It is easy to see that when $n = 1$, the method becomes the quorum consensus method. Thus, this method can be designed to have asymptotically high availability.

Chapter 4

The Proposed Triangular Net Quorum Method

In this chapter, we present our triangular net quorum algorithm. It is based on logically organizing the sites into a hierarchical structure, called triangular net structure. We organize the chapter as follows: In section 4.1, we present the motivation. In section 4.2, we describe the detailed triangular net quorum algorithm. In section 4.3, we prove the correctness of the algorithm and some of its properties. In section 4.4, we analyze the performance of the triangular net quorum method. In section 4.5, we prove that our method can't be implemented by any single level voting method. In section 4.6, we discuss several strategies to organize nodes which can't be arranged into a complete binary triangular net structure.

4.1 Motivation

4.1.1 Analysis of Tree Quorum Algorithm

As we mentioned in section 3.3, the tree quorum algorithms[4] logically organize nodes as a tree. (We only consider binary tree for easy presentation. The extension to the general case is straightforward.) It constructs a quorum by selecting a root-to-leaf path in the tree. If such a path exists, then it is a quorum. If no such path exists due to node failures, then to form a quorum the failed node in some path must be (recursively) substituted for a collection of the paths with each starting from one of its children and ending at a leaf node. As an example, consider the four-level binary tree in Figure 3.1.

If all nodes in the set {1,2,5,10} are functioning, then this set is constructed as a quorum, since the nodes in the set form a root-to-leaf path. If node 1 fails, but all nodes in set {2,5,10,3,6,12} are functioning, this set will also be constructed as a quorum, since {2,5,10} and {3,6,12} are the paths starting from the two children of node 1, which are nodes 2 and 3, and ending at leaf nodes, nodes 10 and 12, respectively. Finally, if in the above set, node 3 fails, but two more nodes, 7 and 14, are functioning, then it is still possible to construct a quorum, namely {2,5,10,6,12,7,14}. This is because nodes 6,12 and nodes 7,14 are paths from the children of the failed node 3.

Clearly, in a tree quorum algorithm, nodes at the higher levels bear more weight than those at the lower levels when constructing a quorum. In the previous example, a single node at level 0 (i.e., node 1) is always worth at least three lower-level nodes in terms of the capability of forming a quorum. In the general case, a single node s at level i is always worth at least $h - i - 1$ lower-level nodes, where h is the total number of levels in the tree. In other words, if node s fails, any quorum in which node s participated requires at least $h - i - 1$ lower-level nodes in lieu of s to remain a quorum. This means that single failure of level i nodes always increases the quorum size by at least $h - i - 1$, which may be substantial when i is small. We believe that this has a negative impact on the average size of a quorum. Another impact is on the availability. Since nodes at different levels have very different capabilities of forming a quorum, the overall availability will be affected if all nodes have similar failure characteristics (which we believe are the most common cases). The factors that contribute to this shortcoming have to do with the structure of a tree. Firstly, for a tree structure different paths initiated at distinct nodes at the same level will never intersect. Thus whenever a high level node must be substituted by the union of the paths in its two subtrees, the size of the union is the sum of the sizes of the individual paths. In other words, every node in the paths contributes to the increased size. Secondly, the height and the width of the bottom level in a tree differ enormously (the ratio is approximately $log_2 n{:}n$ where n is the total number of nodes). As a result, the quorums formed along the bottom can be an order of magnitude larger than the quorums formed along the height.

4.1.2 Outline of a Triangular Net Structure (TNS)

The goal of our algorithm is twofold. One is to preserve the logical clarity and simplicity as well as the essential properties for mutual exclusions, such as the intersection and minimality properties of quorums and non-dominance of the quo-

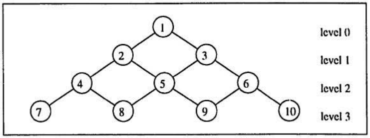


Figure 4.1: An example of 10 nodes organized as a four-level TNS

rum set. The other is to diminish the discrepancy of the nodes at different levels in terms of the capabilities of forming a quorum in the hope of reducing the cost and enhancing availability. We achieve these goals by organizing the nodes in such a way that the shortcoming of a tree structure illustrated in the last section can be overcome. To this end, we organize all nodes into a triangular net structure where the height and the width of the leaf level are roughly the same. Also, two subtrees of a node intersect each other. Shown in Figure 4.1 is a typical TNS, where each node has two children and a distinguished node, node 1, is the root of the TNS.

Note that, as shown in Figure 4.1, in a TNS except for the nodes in the two outer-most paths, each node has two parents. In a TNS, the higher level nodes have capabilities similar to those of the lower level nodes in forming a quorum. For example, in the TNS of Figure 4.1, {1,2,5,8} is a quorum. If node 1 fails, it can be replaced by a single lower-level node 3, resulting in another quorum {2,3,5,8}. In this case, node 1 and node 3 have the same capability of forming a quorum. Furthermore, quorums constructed along either the height or the bottom are roughly the same in size. For example, groups {1,2,5,8} and {7,8,9,10} are the two quorums constructed along the height and the bottom respectively, in the TNS of Figure 4.1. (See section 4.2 for the detail of triangular net quorum algorithm.)

4.2 The Triangular Net Quorum (TNQ) Algorithm

In this section, we give a detailed description of the triangular net quorum algorithm. We first give a formal definition of TNS. We then illustrate how a TNQ algorithm works, based on the TNS structure. For easy presentation, we only consider the simple case. We will extend it to the general case in later sections.

4.2.1 The Structure

As outlined in section 2, a TNS is a hierarchical structure which consists of a number of levels. We will use the convention that these levels are numbered as 0, 1,$\cdots$, in a top-down fashion.

Definition 1: For $h \geq 0$, an $h+1$ level binary TNS (We will omit the term 'binary' when no confusion is possible.) is a collection of interconnected nodes arranged by levels such that for all $i, 0 \leq i \leq h$:

1. There are exactly $i+1$ nodes, denoted by $s_{i0}, \cdots, s_{i,i}$, at level i;

2. For all i, j, $0 \leq i \leq h-1$ and $0 \leq j \leq i$, node $s_{i,j}$ has two children $s_{i+1,j}$ and $s_{i+1,j+1}$ at level $i+1$.

For the above definition, we call the node at level 0 the root of the TNS, the nodes at level h leaves. For all $i, 0 \leq i \leq h$, we call nodes s_{i0} and $s_{i,i}$ side nodes. Figure 4.2 is the general structure of the TNS.

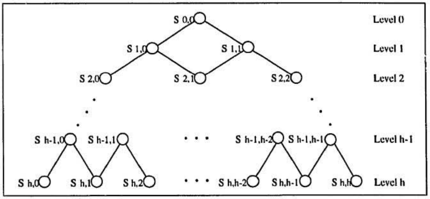


Figure 4.2: The general structure of a (binary) TNS

Since each node in a TNS represents an actual network node, it can only be in one of the two states: up and down. From the functional point of view, all nodes that are down are the same. However, from the viewpoint of a TNQ algorithm, all down nodes do not exhibit the same characteristics in forming a quorum: some down nodes may have enough up successors to form a quorum, while the others do not. To simulate such a scenario, we need two additional states.

Definition: A node s is *open* if the following conditions hold:

1. if s is a leaf, then s is up;
2. if s is an internal node, then either s is up and one of its children is open or s is down and both of its children are open.

otherwise, we say that node s is *closed*.

For easy reference, we will say an up or down state is a *UD-state*, and an open or closed state is an *OC-state*.

Our intention here is to use the OC-state to signify the significance of a node to any quorum which the TNQ algorithm can construct, based on the current network state. Specifically, a node being closed signifies that it is insignificant to any quorums constructed on the current network state. Let us consider again the TNS in Figure 4.1. Suppose nodes 2,3,5,4 and 9 are up, and all the rest are down. By definition, node 9 is open. This in turn implies node 5 is open, which in turn implies nodes 2 and 3 are open. Now consider node 1, which is presumably down. Since both of its children are open, it is open too. It can be verified that, except for these five nodes, all the rest are closed. Note that even though node 4 is up, it is closed, since both of its children are closed. Our algorithm in section 4.2 will tell us that the only quorum that can be constructed is {2,3,5,9}. Apparently, all the closed nodes are insignificant to this quorum. For example, none of the closed nodes 4, 7 and 8 is part of this quorum. Now, assume a different network state where the nodes that are up are 1, 4, 5 and 6. It is easy to verify that for this state none of the nodes is open. Accordingly, our algorithm will not be able to construct any quorum.

4.2.2 The Algorithm

The input to the algorithm is the complete set of network nodes organized into a TNS, as well as the states of each node (i.e., either up or down). For clarity, we present our algorithm as a two pass process, as shown in Figure 4.3. In the first pass, the procedure *Mark* marks all the nodes in the TNS rooted at t as either open or closed. This information will later be used by function *Formquorum* in the second pass to determine how to construct a quorum[1]. Both procedures work recursively. The logic underlying procedure *Mark* directly follows from the definition of an OC-state. It first checks if t has already been marked. This is because subtrees may overlap and a node may have been visited by many instances of recursive calls. The actual marking actions proceed from the bottom to the top. Procedure *Formquorum* will be called upon only if the root of the TNS is open. It returns a quorum in the TNS rooted at node t. Note that, with the exception of leaf nodes, *Formquorum* will always bypass a node with both children open (i.e., does not include it into the quorum) and go straight to process its children. In this case, it does not even care if the parent node is up. This *children-first* approach is different from the *parent-first* approach used by the tree quorum algorithm. (In the parent-first approach, if it is open the parent will never be bypassed unless it is down.) In general, children-first approach tends to construct a quorum along the bottom while parent-first approach constructs a quorum along the height. From the special structure of TNS where the two subtrees of a node always overlap, a parent-first approach does not automatically ensure minimality, and therefore would have to include extra mechanisms to ensure minimality. However, by using children-first approach, it can be proven that the minimality is always guaranteed. (See the proof in section 4.4.)

Consider again the TNS in Figure 4.1. Suppose nodes 2, 3, 4, 5, 6, 7 and 8 are up and all the rest are down. In the first pass, the procedure *Mark* will mark each node as either open or closed as shown in Figure 4.4.a. (The small letters

[1]In the real implementation, it is not difficult to combine these two passes into a single pass if doing so is deemed desirable.

```
ALGORITHM:
{ input: a TNS rooted at node T }
    Mark(T);
    IF T is marked as closed THEN
        stop;   { no quorum can be formed }
    ELSE
        Formquorum(T);
```

```
PROCEDURE Mark(t: NODE)

BEGIN
 IF t is not marked THEN
   IF (t is a leaf) THEN
       IF (t is up) THEN
           mark t as open;
       ELSE mark t as closed;
   ELSE {
       Mark(t.left);
       Mark(t.right);
       IF (t is up) THEN
         IF ((t.left is marked as open) OR (t.right is marked as open)) THEN
             mark t as open;
         ELSE  mark t as closed;
       ELSE IF ((t.left is marked as open) AND
               (t.right is marked as open)) THEN
                 mark t as open;
               ELSE   mark t as closed; }
END
```

```
FUNCTION Formquorum(t: NODE)
{ Note: t is assumed to be open }

BEGIN
   IF (t is a leaf) THEN
            RETURN ({t});
   ELSE
       IF ((t.left is marked as open) AND (t.right is marked as open)) THEN
           RETURN (Formquorum(t.left) ∪ Formquorum(t.right));
       ELSE IF (t.left is marked as open) THEN
            RETURN ({t} ∪ Formquorum(t.left));
         ELSE
            RETURN ({t} ∪ Formquorum(t.right));
END
```

Figure 4.3: The triangular net quorum algorithm

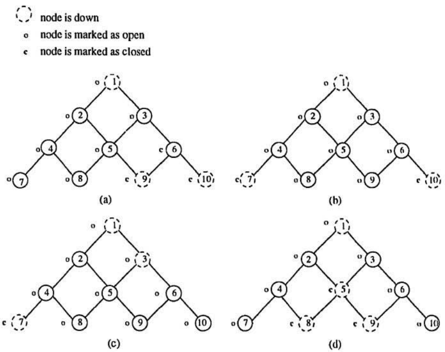


Figure 4.4: An example of different states of a TNS generated by the first pass

o and c attached to the nodes denote OC-states open and closed, respectively. Thι broken-lined circles denote the nodes that are down.) In the second pass, *Formquorum* will choose {3,5,7,8} as the quorum. Suppose node 9 comes up but node 7 goes down. Procedure *Mark* will make the TNS as depicted in Figure 4.4.b. *Formquorum* will construct quorum {4,6,8,9}. Now suppose node 10 comes up and node 3 goes down, resulting in the state of Figure 4.4.c. Accordingly, the algorithm will construct quorum {4,8,9,10}. Another example is shown as Figure 4.4.d. The algorithm will construct the quorum {2,3,4,6,7,10}.

From these examples, we have the following observations: 1. among the three quorums constructed by the TNQ algorithm based on the different network states, none of them is a subset of the others; 2. they pairwise intersect; 3. the differences in quorum sizes are small.

In the following sections, we will prove that the first two statements are in fact true in all cases. The third statement describes the property of the TNQ algorithm which underlies its low cost.

4.3 Correctness

In this section, we establish the basic properties of the TNQ algorithm that are essential to a 'good' mutual exclusion mechanism. These include the following. Her.., S stands for a coterie.

1. Minimality: $\forall G,\ H \in S,\ G \not\subseteq H$;
2. Intersection: $\forall G,\ H \in S,\ G \cap H \neq \phi$;
3. Non-dominance[2]: $\forall G$, if $G \cap H \neq \phi$ for all $H \in S$, then $\exists Q \in S$ such that $G \supseteq Q$.

These properties reflect some of the important criteria for judging the quality of a mutual exclusion method such as correctness and availability.

In what follows, we use T to denote the root of the TNS for which we estabiish the properties. We use S_T to denote the set of all quorums that can possibly be constructed by the TNQ algorithm with a particular TNS rooted at T. For a node p in the TNS, we use $p.left$ and $p.right$ to denote the left and right

[2]In some literature, the quorum set which satisfies the three properties listed here is called ND-coterie. The term 'non-dominance' follows from the term 'non-dominated' coterie.

child of p , respectively. Let R and S be two sets of groups of nodes; we define $R \otimes S \equiv \{U \cup V : U \in R \ \&\ V \in S\}$. In a reasonable abuse of symbols, we sometimes use a letter to represent both the root and the entire structure of a TNS. When this happens, we will precede the letter by the term 'node' or 'TNQ' to indicate its actual meaning. We will use the term 'TNS state' to denote the TNS in which each node is associated with a UD-state and an OC-state (i.e., all nodes have been marked.)

We first introduce two lemmas which will be used in the subsequent proofs.

Lemma 1: S_T can be written as $S_T = G_1 \cup G_2 \cup G_3$ where $G_1 = \{\{T\}\} \otimes S_{T.left}$, $G_2 = \{\{T\}\} \otimes S_{T.right}$ and $G_3 \subseteq S_{T.left} \otimes S_{T.left}$.

Proof. The fact that any group in G_1 or G_2 can be generated by our algorithm directly follows from the definition of function *Formquorum*. Now suppose g can be constructed by our algorithm but is not in $G_1 \cup G_2$. Thus the root must be down and both children are open. Let g_1 and g_2 be the groups returned by the recursive calls on $T.left$ and $T.right$, respectively. Thus $g = g_1 \cup g_2$. Now suppose we force all of the nodes not in structure $T.left$ to go down when the recursive call is made on $T.left$. This will not change the quorum returned by the call. This is because none of these nodes is a child of any node in structure $T.left$, and hence their state change will not affect the OC-states of the nodes inside structure $T.left$. Thus $g_1 \in S_{T.left}$. Using similar arguments we can show $g_2 \in S_{T.right}$. The lemma follows. □

Lemma 2: For any $g \in S_T$, g can be constructed by the TNQ algorithm with a TNS state M where $\forall s \in g$, s is up and $\forall r \notin g$, r is down in M.

Proof. Suppose g is constructed by the TNQ with a TNS state L and there is a node x such that $x \notin g$ and x is up in L. If x is closed in L, changing its state to down certainly will not affect the quorum returned. Now assume node x is open in L. Since $x \notin g$, from the definition of function *Formquorum*, both children of x must be open in L. From the way *Formquorum* works, changing the state of x to down still does not affect the quorum return. We now change the state of x to down, resulting in a new TNS state L_1 in which x is down. We can repeat this process until all the up nodes which are not in g are changed to down. On the other hand, from the way a quorum is constructed, all nodes in g must be up in L. □.

Lemma 3: Let M_1 and M_2 be two TNS states. If $\forall x$, x is up in $M_2 \Rightarrow x$ is up in M_1, then $\forall x$, x is closed in $M_1 \Rightarrow x$ is closed in M_2.

Proof: We prove the lemma by induction on the level where a node resides in the TNS.

Base: $h = l$. All nodes on level h are leaves. Thus if a node x at level h in M_1 is closed, x is down. This means x is down in M_2. By the definition of OC states, x is closed in M_2.

Inductive step: $h < l$. Assume x at level h is closed in M_1. There are two cases:

1. x is down and at least one of its children is closed in M_1. Thus x is down in M_2. Assume y is a child of x with a closed state in M_1. By the induction hypothesis, y is closed in M_2. By the definition of OC states, x is closed in M_2.

2. x is up and both of its children are closed. By the induction hypothesis, both of x's children are closed in M_2. Thus x is closed in M_2.

THEOREM 1. The set of all possible quorums constructed by a TNQ algorithm has the intersection property.

Proof: We prove the theorem by induction on the number h of levels in the TNS.

Base: $h = 1$. There is only one node T in the TNS. Our algorithm will generate $S_T \equiv \{\{T\}\}$. The intersection property follows.

Inductive step: $h > 1$. By Lemma 1, we can write S_T to be the union of the following three subsets:

1. $G_1 \;=\; \{\{T\}\} \otimes S_{T.left}$

2. $G_2 \;=\; \{\{T\}\} \otimes S_{T.right}$

3. $G_3 \;\subseteq\; S_{T.left} \otimes S_{T.right}$

Since $\forall g_1 \in G_1, \forall g_2 \in G_2$, $T \in g_1$ and $T \in g_2$, $g_1 \cap g_2 \neq \phi$. By the definition of G_3, we have $\forall g_3 \in G_3$, $\exists a_1 \in S_{T.left}$ and $\exists a_2 \in S_{T.right}$ such that $g_3 = a_1 \cup a_2$. Thus by the induction hypothesis, the intersection property holds for G_3. Since $g_1 - \{T\} \subseteq S_{T.left}$, by the induction hypothesis, $(g_1 - \{T\}) \cap a_1 \neq \phi$, Thus $g_1 \cap a_1 \neq \phi$. Similarly, $g_2 \cap a_2 \neq \phi$. Thus $g_1 \cap g_3 \neq \phi$ and $g_2 \cap g_3 \neq \phi$. It follows directly from the induction hypothesis that G_3 itself has intersection property. Thus, the theorem follows. □

THEOREM 2. S_T has the minimality property.

Let $g \in S_T$ be an arbitrary quorum, assume $q \subset g$. We now prove that $q \notin S_T$. Assume the contrary. By Lemma 2, g can be constructed in a TNS state, say M_1, such that $\forall s \in g, s$ is up in M_1, and $\forall s \notin g, s$ is down in M_1. Likewise, q can be constructed in a TNS state, say M_2, such that $\forall s \in q, s$ is up in M_2, and $\forall s \notin q, s$ is down in M_2. Since $q \subset g$, we have $\forall x$, x is up in M_2, x is up in M_1. Assume x is the highest level node in the TNS such that $x \in g$ and $x \notin q$, which means x is up in M_1 but down in M_2.

Since $x \in g$ and g can be constructed by the TNQ algorithm with M_1, x must be open. Thus x is either a leaf node or not both of its children are open in M_1. Since x is down in M_2, if it is a leaf node, then it must be closed in M_2. If it is not a leaf node, then not both of its children are open in M_1. By Lemma 3, not both of its children are open in M_2. Thus it must be closed in M_2. Note that x must not be the root, since otherwise q could not possibly be constructed from M_2.

Let y be one of its open parents on which the recursive call is made when g is constructed from M_1. (Note that one of the parents of such a kind must exist, since otherwise no recursive call could be made on node x and therefore node x would not have been included in g.) Let z be the other child of y. We claim y must be closed in M_2. There are two cases: 1. $y \in g$, then z must be closed in M_1, otherwise, both x and z are open, and by TNQ algorithm, we would have $y \notin g$. By Lemma 3, z is closed in M_2, then, both x and z are closed in M_2. Thus y is closed in M_2. 2. $y \notin g$, then y is down in M_1. Therefore it is down in M_2. Since x is closed in M_2, by the definition of OC states, y is closed in M_2.

Now let u be the parent of y on which the recursive call is made when g is constructed from M_1. The same argument as above can prove that u is closed in M_2. We can repeat this process until we reach the root of M_2. This means that no quorum can be constructed within M_2, a contradiction to the assumption that q can be constructed in M_2. $\square$

THEOREM 3: S_T has non-dominance property.

Proof: We prove the theorem by induction on the number h of levels in the TNS.

Base: $h = 1$. There is only one node T in the TNS. Our algorithm will generate $S_T \equiv \{\{T\}\}$. Obviously this set is complete.

Inductive step: $h > 1$. Again we write S_T as the union of the following three subsets.

1. $G_1 = \{\{T\}\} \otimes S_{T.left}$
2. $G_2 = \{\{T\}\} \otimes S_{T.right}$
3. $G_3 \subseteq S_{T.left} \otimes S_{T.right}$

Assume g is an arbitrary group such that g intersects all the groups in S_T. Two cases are possible.

Case 1: $T \in g$. Thus g must intersect all the groups in G_3. If g intersects all the groups in $S_{T.left}$, then by the induction hypothesis, $g \supseteq g_1$ for some $g_1 \in S_{T.left}$. Thus $g \supseteq \{T\} \cup g_1$, which is in the set G_1. Now assume g does not intersect all the groups in $S_{T.left}$. Assume $p_1 \in S_{T.left}$ such that $g \cap p_1 = \phi$. We claim that g intersects all the groups in $S_{T.right}$. If not, let $p_2 \in S_{T.right}$ such that $g \cap p_2 = \phi$.

Thus $g \cap (p_1 \cup p_2) = \phi$. By Lemma 2, p_1 and p_2 can be constructed respectively from the TNS states M_1 and M_2, where $\forall s \in p_1, \forall r \notin p_1$, s is up and r is down in M_1, and $\forall s \in p_2, \forall r \notin p_2$, s is up and r is down in M_2. We define a TNS state M for structure T as follows: $\forall s \in p_1 \cup p_2, \forall r \notin p_1 \cup p_2$, s is up and r is down in M. Clearly, $\forall s$, s is up in M_1 or $M_2 \Rightarrow s$ is up in M. Note that nodes $T.left$ and $T.right$ are open in M_1 and M_2, respectively, since otherwise p_1 or p_2 could not be constructed. Clearly, they will remain open in M. Thus node T is open in M. This means that a quorum can be constructed from M. Let p denote this quorum. Thus $p \subseteq G_3$. Since all the nodes not in $p_1 \cup p_2$ are down in M, we have $p \subseteq p_1 \cup p_2$. This implies $g \cap p = \phi$. This contradicts our assumption that g intersects all the groups in G_3. Thus g intersects every group in $S_{T.right}$. Using the similar arguments to those in the first half of this case, we can prove that g is a superset of some group in G_2.

Case 2: $T \notin g$. Thus g intersects all groups in $S_{T.left} \cup S_{T.right}$. By the induction hypothesis, $\exists g_1 \in S_{T.left}, g \supseteq g_1$ and $\exists g_2 \in S_{T.right}, g \supseteq g_2$. Thus $g \supseteq g_1 \cup g_2$. Using the similar arguments to those in case 1, we can prove $\exists p \in G_3, p \subseteq g_1 \cup g_2$. Thus $g \supseteq p$. □

4.4 Performance Analysis

4.4.1 Availability

We assume that each node has independent failure mode, with the probability p of being up. We define the availability of a mutual exclusion algorithm to be the probability that a quorum can be constructed by the algorithm.

Assume there are n nodes. For majority voting algorithm, the size of a quorum is always $\lceil (n+1)/2 \rceil$. Thus the availability is

$$A_{maj} = \sum_{i=\lceil (n+1/2) \rceil}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

The availability of the *tree* quorum algorithm is obtained from the following recursive relation. Let A_{h+1} be the availability of the tree algorithm for a tree of level $h+1$ where $h \geq 0$. We have

$$\begin{aligned} A_1 &= p; \\ A_{h+1} &= pA_h(1-A_h) + p(1-A_h)A_h + pA_h^2 + (1-p)A_h^2; \end{aligned}$$

i.e.

$$A_{h+1} = 2pA_h + (1-2p)A_h^2.$$

The calculation of the availability by TNQ algorithm is more complex due to the fact that subtrees of a node always overlap in the TNS. Thus the probabilities that a quorum can be constructed in the subtrees of a node are not independent of each other. At this time, we do not know how to use a closed form or a recursive relation to specify this availability in a general case. To calculate the availability, the following method is used. Clearly, the availability of a TNQ algorithm is the probability that the root of the associated TNS is open. In a TNS with $h+1$ levels, for all $i, 0 \leq i \leq h$, there are exactly $i+1$ nodes at level i. Let S_i be the set of all possible combinations of OC-states (open/closed) and R_i the set of all possible combinations of UD-states (up/down) of the nodes at level i. For convenience, we simply call the members of these two sets states. Clearly, S_i and R_i each contains 2^{i+1} different states. Note that from the way the OC-state is defined, for $0 \leq i \leq h-1$, each state in S_i is the union of the two states with one in R_i and the other in S_{i+1}, and the union of any two states with one in R_i and the other in S_{i+1} forms a state in S_i, with the exception that at the leaf level, we have $R_h = S_h$.

The algorithm we use to compute the availability of the TNQ algorithm is briefly described as follows. Given the probability p that each node in the TNS is up. Let P_{ij} and Q_{ij} be the probabilities that the jth states in R_i and S_i occur, where $1 \leq j \leq 2^{i+1}$. (We assume there is an order on the elements in S_i and R_i.) Since the jth state in R_i is just a collection of the UD-states of all the nodes at level i and the UD-states of different nodes are independent of each other, P_{ij} can be computed immediately (i.e, independently of any other levels). The algorithm computes Q_{ij} for all $i, j, 0 \leq i \leq h, 1 \leq j \leq 2^{i+1}$ level by level in a bottom up fashion. From the notes we made at the end of the last paragraph, Q_{ij} is the product of P_{ik} and $Q_{i+1,l}$, assuming the union of the kth state in R_i and the lth state in S_{i+1} gives the jth state in S_i. When Q_{01} and Q_{02} are finally generated, whichever of the two corresponds to the probability that the root is open, is the availability of the TNQ algorithm.

Using the algorithm described above, we evaluated the availabilities of TNQ algorithm and the majority voting algorithm for 6, 15 and 28 nodes. We evaluated the availabilities of the tree algorithm for 7, 15 and 31 nodes, since these numbers are the closest to those used by the TNQ algorithm while maintaining the balanced tree structure. The availability graphs in Figures 4.5 and 4.6 show that, in general, the TNQ algorithm has nearly the same availability with the tree quorum algorithm, and attains the comparable levels of availability with majority quorum algorithm. The availability of the TNQ becomes inferior to the majority quorum algorithm for the value of p ranging from approximately 0.5 to 0.8. When $p > 0.8$, the availabilities become indistinguishable.

For more detailed value, in each case we let the node probabilities vary from 0.5 to 1 with a step size 0.0025. In all cases but 6 nodes, the majority voting has the highest availability. For TNQ and tree algorithm, it is interesting to note that there is a turning point of the node probability which is close to 0.7. The tree algorithm has better availability below this point while the TNQ algorithm does better above it. We have listed ten sample data obtained for each of the three algorithms in cases of 15 and 28 nodes (15 and 31 nodes for the tree algorithm) in Table 1 and Table 2, respectively. The left-most column in each table is the list of the sample node probabilities based on which the availabilities of the three algorithms are evaluated. We use the double lines to indicate the estimated location of the turning points. Note that this comparison is made in cases where the TNQ algorithms use less nodes than the tree algorithm does. We will extend our binary complete structure to 31 nodes and compare the results in section 4.6.

4.4.2 Size of Quorums

In this section, we will compare the maximum, minimum and average sizes of the quorums constructed by the three algorithms.

For a system of n nodes, maximum, minimum and the average sizes of the quorums constructed by the majority voting algorithm are always $\lceil (n+1)/2 \rceil$. For a tree algorithm, the maximum and the minimum quorum sizes are $(n+1)/2$ and $log_2 n$, respectively. To obtain the average quorum size for the tree algorithm, let n_i and s_i be the total number of quorums and the average quorum sizes for a

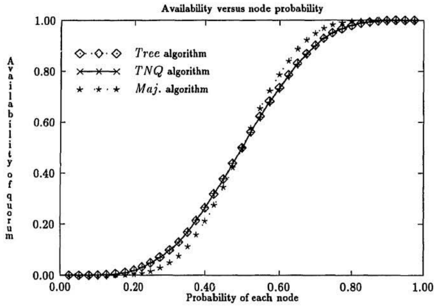


Figure 4.5: Availabilities of three algorithms with 15 nodes

Table 4.1: Some detailed availabilities when 15 nodes are used by the three algorithms

Probability of a node which is up	Tree algorithm	TNQ algorithm	Majority algorithm
0.5350	0.586881	0.585572	0.608726
0.5850	0.703873	0.701325	0.749973
0.6350	0.804545	0.801980	0.860720
0.6850	0.883253	0.881760	0.934645
0.7350	0.938493	0.938440	0.975475
0.7375	0.940667	0.940680	0.976815
0.7850	0.972582	0.973501	0.993238
0.8350	0.990407	0.991434	0.998825
0.8850	0.997755	0.998303	0.999907
0.9350	0.999775	0.999882	0.999998

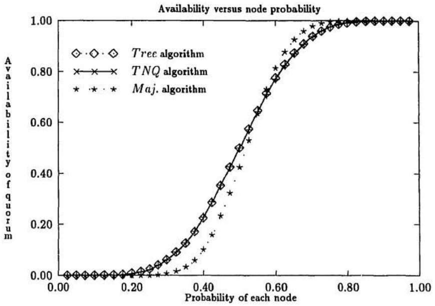


Figure 4.6: Availabilities of tree algo. with 31 nodes and the others with 28 nodes

tree with i levels. We have the following recursive relations[4]:

$$\begin{aligned} n_0 &= 1; \\ s_0 &= 1; \\ n_{i+1} &= 2 * n_i \ + \ n_i^2; \\ s_{i+1} &= (2*(s_i+1)*n_i \ + \ 2*s_i*n_i^2)/n_{i+1}. \end{aligned}$$

We now analyze the sizes for the quorums by the TNQ algorithm. It follows from the way a TNS is marked that the root will be open if all the leaf nodes are up. For a TNS with n nodes, there are approximately $\sqrt{2n}$ leaf nodes. Thus the minimum quorum size for the TNQ algorithm is at most $\sqrt{2n}$. For the maximum

Table 4.2: Some detailed availabilities when 31 nodes are used by tree algorithm and 28 nodes are used by others

Probability of a node which is up	Tree algorithm	TNQ algorithm	Majority algorithm
0.5500	0.646689	0.643741	0.635560
0.6000	0.774970	0.771155	0.813154
0.6500	0.872822	0.870531	0.926422
0.6975	0.935023	0.935012	0.977673
0.7000	0.937527	0.937624	0.979236
0.7500	0.974164	0.975709	0.996218
0.8000	0.991495	0.992996	0.999626
0.8500	0.998006	0.998732	0.999985
0.9000	0.999743	0.999990	0.999999
0.9500	0.999992	0.999999	0.999999

and the average sizes, we can only rely on the experimental results due to the complication caused by the overlap of the subtrees of a node. We have evaluated TNSs with the sizes ranging from 3 to 28. The method for our evaluation is based on the enumeration. The results show that the maximum size generated by the TNQ algorithm is slightly larger than the tree algorithm. For example, in case of 15 nodes, the maximum size by the TNQ algorithm is 9 and that by the tree algorithm is 8. In case of 28 nodes (31 nodes for the tree algorithm), the maximum size by the TNQ is the same as that by the tree algorithm, both being 16. However, the average size by the TNQ algorithm is constantly smaller than that by the tree algorithm. This has been depicted in Figure 4.7. Our data also indicates that in all cases the average quorum sizes by the TNQ algorithm is about 11% smaller than that by the tree algorithm. Among the three algorithm, the majority voting algorithm has the highest average cost.

The above result implies that the quorum sizes generated by the TNQ algorithm are more evenly distributed in the quorum space than that by the tree

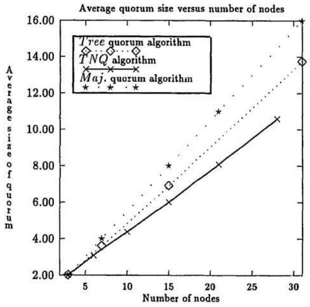


Figure 4.7: The average quorum sizes for the three algorithms

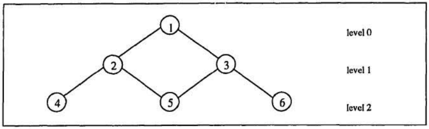


Figure 4.8: An example of 6 nodes organized as a three-level TNS

algorithm. In addition, our data also shows that the nodes at different levels have even more capabilities of forming a quorum under the TNQ algorithm than that under the tree algorithm. For example, under the TNQ algorithm with 15 nodes, the root participates in 96 out of a total of 258 quorums. The average size of the quorums in which the root participates is 5.375, while the average size of the quorums in which the root does not participate is 6.377. On the other hand, under the tree algorithm also with 15 nodes, the root only participates in 30 out of a total of 255 quorums. The average size of the quorums the root participates in is 4.6, compared with the average size of 7.2 of the quorums in which the root does not participate.

4.5 TNQ Algorithm Versus Single Level Voting Algorithm

We now argue that the triangular net quorum algorithm can not be implemented by assigning single level nonnegative integer votes to the nodes in the system(compared with the multi-dimensional voting method[12]). Our technique is similar to that of [14] which was used to show that there is no vote assignment to nodes in a *ND coteric.*

THEOREM 4. *The TNQ algorithm does not have an equivalent vote assignment with single level voting.*

Proof: Assume six nodes are organized as shown in Figure 4.8. Let v_i be the nonnegative integer voter associated with node i, and let T be the sum of total votes, and let M be the majority of total votes. We have the following equations:

$$\begin{array}{ll} v_1 + v_2 + v_4 > M & (1) \\ v_1 + v_3 + v_6 > M & (2) \\ v_4 + v_5 + v_6 > M & (3) \\ v_2 + v_3 + v_5 > M & (4) \\ v_1 + v_2 + v_3 + v_4 + v_5 + v_6 = T & (5) \end{array}$$

From (1)+(2) we have

$$2v_1 + v_2 + v_3 + v_4 + v_6 > T \qquad (6)$$

From (6) and (5) we have

$$v_1 \;>\; v_5$$

By a similar reason form (3) + (4) > (5) we have

$$v_5 \;>\; v_1$$

This is impossible. Thus, there is no one level integer vote assignment corresponding to the quorum set based on the TNS in Figure 4.8. □

4.6 The Methods for Organizing Incomplete Binary TNS

In this section, we discuss how to organize the nodes which can not be arranged as a complete binary TNS. The basic idea is to add or delete some nodes from the complete binary TNS. There are several strategies. We present the ideas by examples. In the following, we consider how to organize 31 nodes.

Method 1: all the extra nodes(with respect to the *complete binary TNS*) will be duplicated to the *root*. Therefore, the root is actually a group of nodes. We can use any method to form a *root quorum* which is constructed by the nodes in a root group, and a root quorum has the intersection property. Then, we can redefine the root's UD-state as follows: if the up state nodes in root group can form a root quorum, then we say the root is up; otherwise, it is down. For example, in Figure 4.9 method 1, we group nodes 1, 2, 3 and 4 as a root group, and we define {1,2}, {1,3}, {1,4} and {2,3,4} as the root quorum. When any of these root quorums can be constructed, the root is up. Otherwise, the root is down.

Method 2: we add extra nodes on the *leaf* level, and change the OC-state definition for nodes which have more than two leaf children. Each node s is marked as open state iff one of following conditions hold:

1. s is up and one of its children is up; or

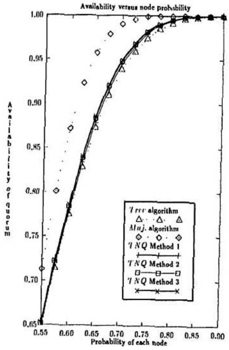
2. all of s's children are up.

For example, in Figure 4.9 method 2, node 17 is *open* only if node 17 is up and one of node 23, 24 and 25 is up; or node 23, 24 and 25 are all up if node 17 is down.

Method 3: we combine method 1 and method 2. The basic idea is that we add some nodes on the *leaf* level, and add some nodes in the root group. This method can be used when a large number of nodes are added to a complete TNS. The definition of UD-state of root and OC-state of leaf level nodes can be the same as used in method 1 and 2. For example, in Figure 4.9 method 3 we define {1,2}, {1,3} and {2,3} as the root quorum. Therefore, root is up only if a minimum of two of nodes 1, 2 and 3 are up. In addition, we define node 21 to be open if node 21 is up and one of nodes 27, 28 and 29 is up or nodes 27, 28 and 29 are all up.

In Figure 4.9, we use these three methods to add nodes to a complete binary TNS of 28 nodes, resulting in the TNS of 31 nodes. The left diagram in Figure 4.9 is the comparison of the availability with the tree quorum algorithm and majority quorum method with 31 nodes. From the results shown in Figure 4.9, our algorithm have better availability than tree algorithm when the probability of each node is higher than 0.5, and the majority quorum algorithm still has the highest availability. We have calculated the average quorum size for all these methods. The computational result shows that method 1 only increases the average size of quorum to 10.79, which is 2.95 less than the average size of 13.84 for the tree quorum algorithm with 31 nodes, and 5.21 less than 16 for majority quorum method.

The extension method we introduced above suggests a different way of organizing nodes. That is, a node can be either a real node, or a group of nodes. Our above example shows that such an organization may provide good availability and performance. The full potential of this method is an interesting topic for the future research.



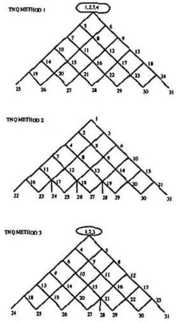


Figure 4.9: Various structures for organizing 31 nodes as the TNS and the availabilities of these methods compared with tree and majority quorum methods

Chapter 5

Generalization of Triangular Net Quorum Algorithm

In the previous chapter, we introduced the triangular net quorum algorithm on the complete binary TNS, and we further analyzed the performance of the TNQ algorithm. In this chapter, the binary TNS will be extended to general TNS where each node has more than two children, and each child may have more than one parent. We will develop the generalized TNQ algorithm for general mutual exclusion purpose. We organize this chapter as follows: In section 5.1, we describe the extension of TNS. In section 5.2, we present the generalized TNQ algorithm. Finally, in section 5.3, we provide a discussion about this generalized TNQ algorithm.

5.1 Extensions of TNS

The TNS we have discussed so far allows an internal node to have only two children. This restriction can be lifted to make the TNQ algorithm more general. Note that if a parent has more than two children, then a child may also have more than one parent. The assignment of the parent nodes to each child node must be properly distributed. In general, suppose each internal node has m children and, counted from the left the nodes at level i are s_{i0}, $s_{i1}, \cdots, s_{iq}$, we use the terminology *degree* to indicate the number of children for each parent. Besides each root's children having only one parent, other TNS nodes will be organized in such a way that s_{i0} and s_{iq} each has one parent, s_{i1} and $s_{i,q-1}$ each has two parents, $\cdots$, $s_{i,m-2}$ and $s_{i,q-m+2}$ each has $m-1$ parents, and for all $p, m-1 \le p \le q-m+1$, $s_{i,p}$ has m parents.

Let $r, \frac{m+1}{2} \le r \le m$ be an integer. We define the OC-states for a node s as follows.

Definition: A node s is open if the following conditions hold:

1. if s is a leaf node, then s is up;

2. if s is an internal node, then either it is up and has at least $m-r+1$ open children, or it has at least r open children.

otherwise it is closed.

As an example, 12 nodes are organized as three level generalized TNS shown in Figure 5.1. In Figure 5.1, each internal node has four children.

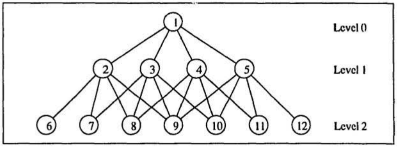


Figure 5.1: A generalized three-level four degree TNS

5.2 A Generalized TNQ Algorithm

The generalized TNQ algorithm for the extended TNS is similar to that for the binary TNS. It also contains two passes, with the first pass defining the OC-state for each node and the second pass constructing the quorum. The first pass for the general TNQ follows the definition of an OC-state here exactly the same way as the first pass for the restricted TNQ followed the definition of an OC-state in section 5.1. The second pass for the general TNQ in principle also follows the pattern for the TNQ in the binary TNQ algorithm. Briefly, it works as follows. The second pass constructs the quorums for top level to bottom level based on the marked TNS in the first pass. The root of the TNS is always assumed open when the second pass is called for. Starting from the root of the TNS, if a node has at least r open children, then form the quorum based on any r open children. If a node has less than r but at least $m-r+1$ open children, then quorum will be formed based on any of its $m-r+1$ open children. Although this simple extension method guarantees correctness, it may nonetheless violate the minimality property. Let us consider an example shown in Figure 5.2. We assume the threshold r is 3. We use a solid circle to identify an up node and a dotted circle for a down node. The small letters o and c attached to the nodes denote OC-states open and closed, respectively. When we recursively construct the quorum starting from the root, root 1 has only two open children, which is less than $r(r = 3)$. Therefore, it has only one choice, namely, to construct the subquorums starting from node 2 and node 3. When node 2 constructs its subquorum, it has four open children, such as nodes 6, 7, 8 and 9. It can choose any three of them to construct the subquorum. Suppose it chooses nodes 6, 7 and 8. In addition, when node 3 constructs its subquorum, it has only three open children. Then, it has only one choice to construct the subquorum as {7,8,9}. Finally, we construct the quorum as {1,6,7,8,9}. But, we can see that node 6 is not necessary in this quorum, since node 2 has four open children in this quorum and we only need three$(r = 3)$. This problem arises since when node 2 and node 3 independently construct their subquorums, they do not communicate with each other. Instead, they simply pick up any of their open children independently. This may cause redundancy.

From this example, it is quite obvious that the complexity of the extended TNS will make the generalized TNQ algorithm more complex compared with the binary TNQ algorithm. We consider the generalization problems as follows.

1. Since different parents may share more than one child in the extended TNS, in generalized TNQ algorithm, when different parents choose their children, they will influence each other more than in the binary TNQ algorithm.

2. The binary TNQ algorithm has only one chance to construct the subquorum for its children (if both children are marked as open, we choose both of them

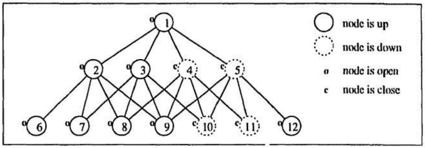


Figure 5.2: An example of the complexity for the generalized TNS

without including the parent in the quorum, otherwise, choose the parent and construct the subquorum for its one open child). However, in generalized TNQ algorithm, we may have several choices when the parent chooses its open children. Since a parent may have more than r open children, or when the parent is down, it may have more than $m - r + 1$ open children.

From the above analysis, we can see the key problem: how to choose the open children. We propose a solution based on the priority of each nodes. In our algorithm, after we mark the extended TNS, we construct the quorum level by level starting from the root level. For each level, we pick up the nodes according to their priorities. In function *Formquorum*(Figure 5.5), we define the priority of nodes at same level according to following rules:

1. If a node has more *relative parents*, then it has higher priority. Here, we define the relative parent d of a node s as a parent of s, on which an instance of the recursive call has been made. For example, in Figure 5.2, *root* 1 is a relative parent to children 2 and 3, since the call is first made on root. In addition, nodes 2 and 3 are the relative parents of their children since

root 1 must have generated two instances of recursive calls on nodes 2 and 3. Thus, nodes 7, 8 and 9 have the higher priority than node 6, since they have two relative parents compared with one relative parent for node 6.

2. If a node's parent has less than r but more than $m - r + 1$ open children, then this node has higher priority than other open nodes at same level whose parents have more than r open children.

3. At the same level, if two open nodes both meet conditions 1 and 2, then the left side node has higher priority than the right side node. For example, in Figure 5.2, node 7 has higher priority than node 8; as well, node 8 has higher priority than node 9.

The detailed algorithm is shown in Figure 5.3. In the algorithm, t is the root of the TNS. The parameter "degree(t)" is the number of children for each node t except for leaves. The parameter "threshold(t)" is the threshold r which we mentioned in section 5.1, such that $\frac{m+1}{2} \leq r \leq m$. In addition, each node t is associated with an integer number, *t.open_child_num*, which indicates the number of open children of this node. We also assume that *t.open_child_num* is initialized to 0.

In *Mark* procedure(Figure 5.4), we assume that each node(except for a leaf) has m children, *t.child*(i) denotes the ith child of t counted from left to right. Here, the leftmost position number is 1. Another difference with respect to the binary TNS marking procedure is that besides marking the OC-state of each node, we also calculate the *open_child_number* for each node except for leaves.

In Figure 5.5, we provide the generalized TNQ algorithm. According to the priority requirements, for each level, we first sort the opens node by the number of relative parents in descending order. Therefore, in the sorted list, the left node has higher priority than the right node. We always assume that the algorithm picks nodes from left to right in the sorted list. Our algorithm constructs the quorum from top level to bottom level. For each level i, we first identify *current_chosen_set* at level i. Then we decide which node in *current_chosen_set* must be constructed in the final quorum. When we reach level $i+1$, the *current_chosen_set* at level i will become *parent_chosen_set* for nodes on level $i+1$.

We construct the *current_chosen_set* at level i as follows.

1. At level 0, *current_chosen_set* = $\{root\}$. If the root has less than r open children, then it is included into the *quorum_set*.

2. At level $i > 0$, if an open node has a parent at level $i-1$, which is a member of *quorum_set* and which has less than $m - r + 1$ open children in the *current_chosen_set* at level i, then add this node to the *current_chosen_set*.

3. At level $i > 0$, If an open node has a parent which is not a member of *quorum_set* but is in the *parent_chosen_set*, and has less than r open children in the *current_chosen_set*, then this node is added to the *current_chosen_set*.

For each level, after we define the *current_chosen_set*, we will pick up some nodes from this set and add them to the *quorum_set*, according to the following rule: if a node in *current_chosen_set* has less than r open children, then add it to *quorum_set*. A complete example is provided in Figure 5.6.

5.3 Discussion

The extended TNS has the smaller height but wider bottom than the binary TNS, if roughly the same number of nodes are used. Thus, it decreases the minimum quorum size. It is not clear to us at this time how the maximum and average quorum sizes in a general TNQ differ from those in a binary TNS. Our conjecture is that these sizes will also decrease since it provides more overlapping among the subtrees of a node. On the other hand, It is not clear that our extended TNQ algorithm will hold the minimality and non-dominance properties since we haven't found an easy way to prove them, although the proof for intersection property is straightforward. These are interesting topics that deserve future study.

```
ALGORITHM
/* input: a TNS rooted at node t */

BEGIN
    For any node $x$ in TNS, $x.open\_child\_num = 0$;
    $Mark$($t$,degree($t$),threshold($t$));
    If ($t$ is marked as *closed*) THEN
            stop; /* *no quorum can be formed* */
    ELSE
            $Formquorum$($t$,height($t$),degree($t$),threshold($t$));
END
```

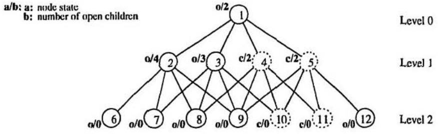
Figure 5.3: The generalized TNQ main algorithm

```
PROCEDURE $Mark$($t$:NODE; $m$,$r$:INTEGER)
/* $t$: root of the TNS; $m$: degree of the TNS; $r$: threshold of the TNS */

BEGIN
 IF ($t$ is not marked) THEN
    IF ($t$ is a $leaf$) THEN
       IF ($t$ is $up$) THEN {
          mark $t$ as $open$;
          $\forall q$, $q = t.parent$, $q.open\_child\_num$ INC 1;
       }
       ELSE mark $t$ as $closed$;
    ELSE {
       FOR $position = 1$ to $m$
          IF ($t.child(position)$ is not marked) THEN
             Mark($t.child(position), m, r$);
       IF ($t$ is $up$) THEN
          IF ($t.open\_child\_num \geq m - r + 1$) THEN {
             mark $t$ as $open$;
             $\forall q$, $q = t.parent$, $q.open\_child\_num$ INC 1;
          }
          ELSE mark $t$ as $closed$;
       ELSE IF ($t.open\_child\_num \geq r$) THEN {
               mark $t$ as $open$;
               $\forall q$, $q = t.parent$, $q.open\_child\_num$ INC 1;
             }
             ELSE mark $t$ as $closed$;
    }
END
```

Figure 5.4: The generalized TNQ mark algorithm

```
Function $Formquorum(t$: NODE; $h, m, r$: INTEGER)
    /* Note: the state of root is open */
    /* $h$: height of the TNS; $m$: degree of the TNS; $r$: threshold of the TNS */

BEGIN
    $quorum\_set = \Phi$;
    $parent\_chosen\_set = \Phi$;
    $current\_level$=0;
    WHILE $current\_level \leq h$ {
        $temp\_set \leftarrow$Open state nodes on $current\_level$ and one of its
                    parents exists in $parent\_chosen\_set$;
        sort $temp\_set$ by number of node's relative parents in descending order;
        IF ($current\_level = 0$) THEN
            $current\_chosen\_set = \{t\}$;
        ELSE $current\_chosen\_set = \Phi$;
    /* step one */
        $\forall$ node $t \in temp\_set$
            IF ($quorum\_set \neq \Phi$ and ( $\exists$ $t.parent \in quorum\_set$ and $t.parent$
                has less than $m - r + 1$ children in $current\_chosen\_set$ )) THEN
                    $current\_chosen\_set = current\_chosen\_set \cup \{t\}$;
        $temp\_set = temp\_set - current\_chosen\_set$;
    /* step two */
        WHILE $temp\_set \neq \Phi${
            IF ($t \in temp\_set$ and ( $\exists t.parent \in parent\_chosen\_set$ and
                $t.parent$ has less than $r$ children in $current\_chosen\_set$ )) THEN
                    $current\_chosen\_set = current\_chosen\_set \cup \{t\}$;
            $temp\_set = temp\_set - \{t\}$;
        }
    /* step three */
        $\forall$ $t \in current\_chosen\_set$
            IF ($t.open\_child\_num < r$) THEN
                $quorum\_set = quorum\_set \cup \{t\}$;
        $parent\_chosen\_set = current\_chosen\_set$;
        $current\_level$ INC 1;
    }
    RETURN($quorum\_set$);
END
```

Figure 5.5: The generalized TNQ formquorum algorithm



Formquorum steps: (THRESHOLD = 3)

current_level = 0

quorum_set = { }

parent_chosen_set = { }

temp_set = { }

current_chosen_set = { 1 }

step 1: do nothing

step 2: do nothing

step 3: quorum_set = { 1 }

current_level = 1

quorum_set = { 1 }

parent_chosen_set = { 1 }

temp_set = {2,3}

current_chosen_set = { }

step 1: current_chosen_set = {2,3}

step 2: do nothing

step 3: parent_chosen_set = {2,3}

current_level = 2

quorum_set = { 1 }

parent_chosen_set = {2,3}

temp_set = {7,8,9,6}

current_chosen_set = { }

step 1: do nothing

step 2: current_chosen_set = {7,8,9}

step 3: quorum_set = {1,7,8,9}

Therefore, the final quorum is

{1,7,8,9}

Figure 5.6: An example of generalized TNQ algorithm

Chapter 6

Conclusion

In this thesis, we describe the triangular net algorithm for achieving mutual exclusion. Our algorithm is based on organizing the network nodes into a triangular net structure. Like a tree structure, it contains a number of levels and the nodes at different levels are associated by a parent-child relationship. Unlike a tree structure, however, different children may share the same parents. It is because of this increased sharing that our algorithm possesses some desirable properties which a tree algorithm does not have. We show that our algorithm provides a more uniform treatment to the nodes, which we believe is desirable for a truly distributed system. We show that our algorithm has a good average case behavior for distributed systems, with reasonab.'. large sizes. We compare the performance of our algorithm with majority voting and tree algorithms when sites number less than 31. The results show that our algorithm has a better average-cost than either of them. In this cases, we believe these includes the most common cases among the current distributed systems. When the node probability is above the turning points, our algorithm provides a higher availability than the tree algorithm. We believe the triangular net algorithm is desirable for implementing mutual exclusion in a truly distributed system.

Bibliography

[1] A. A. Albert and R. Sandler, *An Introduction to Finite Projective Planes*, Holt, Rinehart, and Winston, New York, 1968.

[2] A. El Abbadi, "Adaptive protocols for managing replicated distributed systems", *IEEE Symp. on Parallel and Distributed Proc.*, pp. 36-43, 1991.

[3] A. El Abbadi, D. Skeen and F. Cristian "An efficient, fault-tolerant protocol for replicated data management", *Proc. of 4th ACM Symp. on Principles of Database Syst.*, pp. 215-228, 1985.

[4] D. Agrawal and A. El Abbadi, "An efficient solution to the distributed mutual exclusion problem", *ACM Trans. on Comput. Syst.*, Vol. 9, No. 1, pp. 1-20, 1991.

[5] D. Agrawal and A. El Abbadi, "The generalized tree quorum protocol: An efficient approach for managing replicated data", *ACM Trans. on Database Syst.*, Vol. 17, No. 4, pp. 689-717, 1992.

[6] D. Agrawal and A. El Abbadi, "Resilient logical structures for efficient management of replicated data", *Proc. of the 18th VLDB Conf.*, pp. 151-162, 1992.

[7] M. Ahamad and M. H. Ammar, "Performance characterization of quorum consensus algorithm for replicated data", *IEEE Trans. on Software Engineering*, Vol. 15, No. 4, pp. 492-496, 1989.

[8] P. A. Alsberg and J. D. Day, "A principle for resilient sharing of distributed resources", *Proc. 2nd Int'l. Conf. on Software Engineering*, pp. 562-570, 1976.

[9] M. Barborak and M. Malek, "The consensus problem in fault-tolerant computing", *ACM Comp. Surveys*, Vol. 25, No. 2, pp. 171-220, 1993.

[10] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[11] S. Y. Cheung, M. H. Ammar and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data", *Proc. Int'l. Conf. on Data Engineering*, pp. 438-445, 1990.

[12] S. Y. Cheung, M. Ahamad and M. H. Ammar, "Multi-dimensional voting: A general method for implementing synchronization in distributed systems", *ACM Comp. Syst.*, Vol. 9, No. 4, pp. 399-431, 1991.

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

[14] H. Garcia-Molina and H. Barbara, "How to assign votes in a distributed system", *J. ACM*, Vol. 32, No. 4, pp. 841-860, 1985.

[15] D. K. Gifford, "Weighted voting for replicated data", *Proc. 7th Symp. on Operating Syst. Principles*, pp. 150-162, 1979.

[16] J. Gray, "Notes on database operating systems", *Operating Systems: An Advanced Course*, New York: Springer-Verlag, 1979.

[17] J. M. Helary, N. Plouzeau and M. Raynal, "A distributed algorithm for mutual exclusion in an arbitrary network", *Computer J.*, Vol. 31, No. 4, pp. 289-295, 1988.

[18] M. Herlihy, "Dynamic quorum adjustment for partitioned data", *ACM Trans. on Database Syst.*, Vol. 12, No. 2, pp. 170-194, 1987.

[19] T. Ibaraki and T. Kameda, "A theory of coteries: mutual exclusion in distributed systems", *IEEE Trans. Parallel and Distribut. Syst.*, Vol 4, No. 7, pp. 779-794, 1993.

[20] A. Kumar, "Hierarchical quorum consensus: A new algorithm for managing replicated data", *IEEE Trans. Comput.*, Vol. 40, No. 9, pp. 996-1004, 1991.

[21] A. Kumar and S. Y. Cheung, "A high availability $\sqrt{N}$ hierarchical grid algorithm for replicated data", *Inform. Process. Lett.*, Vol. 40, No. 6, pp. 311-316, 1991.

[22] A. Kumar and A. Segev, "Cost and Availability Tradeoffs in Replicated Data concurrency control", *ACM Trans. on Database Syst.*, Vol. 18, No. 1, pp. 102-131, 1993.

[23] L. Lamport, "The implementation of reliable distributed multiprocess systems", *Comput. Networks*, Vol. 2, No. 2, pp. 95-114, 1978.

[24] G. Lelann, "Motivations, objectives and characterization of distributed systems", *Distributed Systems - Architecture and Implementation*, LNCS 105, Springer Verlag, pp. 1-9, 1981.

[25] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems", *ACM Trans. on Comput. Syst.*, Vol. 3, No. 2, pp. 145-159, 1985.

[26] S. R. Mahaney and F. B. Schneider, "Inexact agreement: accuracy, precision, and graceful degradation", *Proc. of the fourth ACM Symp. on Principles of Distributed Computing*, pp. 237-249, 1985.

[27] P. E. O'Neil, "The escrow transactional method", *ACM Trans. on Database Syst.*, Vol. 11, No. 4, pp. 405-430, 1986.

[28] M. Rabinovich and E. D. Lazowska, "A fault-tolerant commit protocol for replicated databases", *Proc. of 11th ACM SIGAT-SIGMOD-SIGART Symp. on Principles of Database Syst.*, pp. 139-148, 1992.

[29] K. V. S. Ramarao and K. Brahmadathan, "Divide and conquer for distributed mutual exclusion", *Proc. 2th IEEE Symp. on Para. Distr. Processing*, pp. 113-120, 1990.

[30] K. Raymond, "A tree-based algorithm for distributed mutual exclusion" *ACM Trans. on Comp. Syst.*, Vol. 7, No. 1, pp. 61-77, 1989.

[31] B. A. Sanders, "The information structure of distributed mutual exclusion algorithms", *ACM Trans. on Comp. Syst.*, Vol. 5, No. 3, pp. 284-299, 1987.

[32] S. K. Sarin, B. T. Blaustein and C. W. Kaufman, "System architecture for partition-tolerant distributed databases", *IEEE Trans. on Comp.*, Vol. c-34, No. 12, pp. 1158-1163, 1985.

[33] R. Satyanarayanan and D. R. Muthukrishnan, "A note on Raymond's tree based algorithm for distributed mutual exclusion", *Inform. Process. Lett.*, Vol. 43, No. 5, pp. 249-255, 1992.

[34] R. Schlicting and F.B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems", *ACM Trans. Comput. Syst.*, Vol. 1, No. 3, pp. 222-238, 1982.

[35] F. B. Schneider, "Synchronization in distributed programs", *ACM Trans. on Program. Lang. Syst.*, Vol. 4, No. 2, pp. 125-148, 1982.

[36] M. Sloman and J. Kramer, *Distributed Systems and Computer Networks*, Prentice-Hall International, 1987.

[37] I. Suzuki and T. Kasami, "A distributed mutual exclusion algorithm", *ACM Trans. on Comput. Syst.*, Vol. 3, No. 4, pp. 344-349, 1985.

[38] J. Tang and N. Natarajan, "Obtaining coteries that optimize the availability of replicated databases", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 2, pp. 309-321, 1993.

[39] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases", *ACM Trans. on Database Syst.*, Vol. 4, No. 2, pp. 180-209, 1979.

[40] P. Triantafillou and D. Taylor, "Efficient maintaining availability in the presence of partitionings in distributed systems", *Proc. 7th Int'l. Conf. on Data Engineering*, pp. 34-41, 1991.

[41] P. Triantafillou and D. Taylor, "A new paradigm for high availability and efficiency in replicated distributed databases", *Proc. 2nd IEEE Symp.on Parallel Distributed Processing*, pp. 136-143, 1990.

[42] J. L. Van de Snepscheut, "Fair mutual exclusion on a graph of processes", *Distribut. Comput.*, Vol. 2, No. 2, pp. 113-115, 1987.

[43] O. Wolfson and S. Jajodia, "Distributed algorithms for dynamic replication of data", *Proc. 11th ACM SIGAT-SIGMOD-SIGART Symp. on Principles of Database Syst.*, pp. 149-163, 1992.

[44] C. Wu and G. G. Belford, "The triangular lattice protocol: A highly fault tolerant and highly efficient protocol for replicated data", *IEEE 11th Proc. Symp. on Reliable Distributed Syst.*, pp. 66-73, 1992.

[45] C. Wu, "Replica control protocols that guarantee high availability and low access cost", *Research Proposal, Dept. of Computer Science, University of Illinois at Urbana-Champaign*, 1992.