

A TRIANGULAR-MESH-BASED APPROACH TO
FAULT-TOLERANT DISTRIBUTED MUTUAL EXCLUSION

A Thesis

Submitted to the Faculty

of

National Sun Yat-sen University

by

Yao-Jen Chang

In Partial Fulfillment of the
Requirements for the Degree

of

Master of Science

June, 1995

TABLE OF CONTENTS

| | Page |
|---------------------------------------------------------------|------|
| LIST OF FIGURES | ii |
| LIST OF TABLES | iv |
| ABSTRACT | v |
| 1. Introduction | 1 |
| 1.1 Fault Tolerance in Distributed Mutual Exclusion | 1 |
| 1.2 Motivations | 3 |
| 1.3 Organization of the Thesis | 4 |
| 2. A Survey of Replica Control Protocols | 5 |
| 2.1 The System Model | 5 |
| 2.2 Conventional Replica Control Protocols | 6 |
| 2.2.1 Majority Voting | 7 |
| 2.2.2 Quorum Consensus | 8 |
| 2.2.3 Coterie | 9 |
| 2.2.4 Dynamic Voting | 12 |
| 2.3 Quorum Protocols Imposing Logical Structures | 14 |
| 2.3.1 Tree Protocol | 14 |
| 2.3.2 Grid Protocol | 16 |
| 2.3.3 Hierarchical Quorum Consensus | 16 |
| 3. The Triangular Mesh Protocol | 19 |
| 3.1 Definitions | 19 |
| 3.2 The Algorithm for Constructing Quorums | 22 |
| 3.3 Proof of Correctness | 28 |

| | Page |
|----------------------------------------------------------|-----------|
| 3.4 Properties of the Protocol | 29 |
| 4. The Triple Triangular Mesh Protocol | 34 |
| 4.1 Definition | 34 |
| 4.2 The Algorithm for Constructing a Quorum | 36 |
| 4.3 Proof of Correctness | 45 |
| 4.4 Properties of the Protocol | 46 |
| 5. The Dynamic Triangular Mesh Protocol | 49 |
| 5.1 Definition | 49 |
| 5.2 The Algorithm for Constructing Quorums | 50 |
| 5.3 Proof of Correctness | 55 |
| 5.4 An Extension | 58 |
| 5.5 Properties of the Protocol | 58 |
| 6. The Performance | 60 |
| 6.1 Quorum Size | 60 |
| 6.2 Availability | 61 |
| 6.3 Fault Tolerance | 64 |
| 7. Conclusion | 72 |
| 7.1 Summary | 72 |
| 7.2 Future Work | 74 |
| BIBLIOGRAPHY | 76 |

LIST OF FIGURES

| Figure | Page |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 2.1 A complete binary tree of seven copies of a replicated object | 15 |
| 2.2 A 4×4 grid in the grid protocol | 16 |
| 2.3 Organizes 9 copies into a 2-level hierarchy | 17 |
| 3.1 A 6-triangular mesh | 20 |
| 3.2 Examples of type-1 quorums in TM: (a) a type-1 quorum in a 6-triangular mesh; (b) a type-1 quorum whose subquorum 1 contains only the center. . . | 21 |
| 3.3 Examples of type-2 quorums in TM: (a) a type-2 quorum in a 6-triangular mesh; (b) a type-2 quorum whose subquorum 2 contains only the center. . . | 21 |
| 3.4 The flowchart for procedure <i>construct_quorum</i> in TM | 24 |
| 3.5 The flowchart for procedure <i>construct_subquorums</i> in TM | 25 |
| 3.5 The flowchart for procedure <i>construct_subquorums</i> in TM (continued) . . . | 26 |
| 3.6 A 3-triangular mesh. | 31 |
| 3.7 The partition of nodes in a 7-triangular mesh | 32 |
| 4.1 Subtriangles in a 6-triangular mesh in TTM | 35 |
| 4.2 Sides of subtriangles in TTM: (a) subtriangle 0; (b) subtriangle 1; (c) sub- triangle 2. | 35 |
| 4.3 Examples of quorums associated with a node a in a 6-triangular mesh where a resides inside the triangle in TTM: (a)-(h). | 37 |
| 4.4 Examples of quorums associated with a node a in a 6-triangular mesh where a resides at one side but is not at a corner in TTM: (a)-(d). | 38 |

| Figure | Page |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 4.5 The flowchart for procedure <i>construct_quorum</i> in TTM | 41 |
| 4.6 The flowchart for procedure <i>construct_subquorums</i> in TTM | 42 |
| 4.6 The flowchart for procedure <i>construct_quorum</i> in TTM (continued) | 43 |
| 5.1 An example of a quorum in a 9-triangular mesh in DTM | 50 |
| 5.2 The flowchart for procedure <i>construct_quorum</i> in DTM | 53 |
| 5.3 The flowchart for procedure <i>construct_subquorums</i> in DTM | 54 |
| 5.4 Examples of quorums in DTM: (a)-(f) | 57 |
| 5.5 Examples when N can not be expressed as $\frac{k(k+1)}{2}$ exactly in DTM: (a)N=16 (k=5, r=1); (b)N=18 (k=5, r=3); (c)N=20 (k=5, r=5). | 59 |
| 6.1 A comparison of the quorum size when no node failure occurs | 62 |
| 6.2 A comparison of the quorum size when node failures occur in the worst case (N = 15) | 64 |
| 6.3 The availability of TM protocol with different N | 65 |
| 6.4 The availability of TTM protocol with different N | 66 |
| 6.5 The availability of DTM protocol with different N | 67 |
| 6.6 A comparison of availability when N=15 | 68 |
| 6.7 An example of node failures which make all quorums of TM and TTM protocols unavailable | 69 |
| 7.1 Systems used to illustrate how the topology impacts the design of dis- tributed mutual exclusion protocols: (a)-(b) | 74 |

LIST OF TABLES

| Table | Page |
|--------------------------------------------------------------------------------------------------------------------------------------------|------|
| 6.1 The simulation results of TM, TTM and DTM protocols: (a) $N = 6$; (b) $N = 10$; (c) $N = 15$; (d) $N = 21$; (e) $N = 28$ | 63 |
| 6.2 A comparison of six mutual exclusion algorithms imposing logical structures | 71 |

ABSTRACT

In the problem of mutual exclusion, concurrent access to a shared resource using a structural program abstraction called a *Critical Section (CS)* must be synchronized such that at any time only one process can enter the CS. In a distributed system, due to the lack of both in shared memory and a global clock, and due to unpredictable message delay, the design of a distributed mutual exclusion protocol that is free from deadlock and starvation, is much more complex than that in a centralized system. To reduce the overhead of achieving mutual exclusion while supporting fault tolerance, in the thesis, we apply the *triangular-mesh-based* approach to design fault-tolerant protocols for mutual exclusion, in which the nodes in the system are organized into a triangular mesh. We propose three protocols based on the triangular mesh: the *triangular mesh* protocol, the *triple triangular mesh* protocol and the *dynamic triangular mesh* protocol. In the triangular mesh protocol, we associate with each node two special patterns. The nodes in each special pattern constitute a quorum. In the triple triangular mesh protocol, a quorum contains nodes from some side of each of three subtriangles in the triangular mesh. In the dynamic triangular mesh protocol, three dynamic paths constitute a quorum in the given triangular mesh. The quorum size of these proposed protocols is K that is $O(\sqrt{N})$, where $N = \frac{K*(K+1)}{2}$ is the number of nodes in the system. In the worst case, the triangular mesh protocol, the triple triangular mesh protocol and the dynamic triangular mesh protocol are fault-tolerant up to $(K - 2)$, $(K - 2)$ and $(K - 1)$ site and communication failures, respectively. From our simulation study, these three proposed protocol can have higher availability than Cheung et al.'s grid protocol. Moreover, the quorum size of the proposed protocols will be less than that in Kumar's HQC protocol when N is greater than or equal to 15 and less than that in Agrawal et al.'s tree quorum protocol when node failures of some sort exist.

CHAPTER I

Introduction

A distributed system consists of a collection of geographically dispersed autonomous nodes connected by a communication network. The nodes have no shared memory, no global clock, and communicate with one another by passing messages. Message propagation delay is finite but unpredictable.

Distributed systems offer many advantages, including resource sharing and fault-tolerance. The latter can be achieved by replicating a resource at nodes with independent failure modes. Replication can also improve performance when load is shared among the nodes that have instances of a resource. In many applications, users need to synchronize access to shared resources. For example, when data is replicated to improve its availability, updating the file requires *mutually exclusive* access. This is necessary for maintaining the consistency of the data. The synchronization technique should work in the presence of node and communication failures.

The disadvantage, however, is that the read and write operations performed on a replicated object by various concurrent transactions must be synchronized. For instance, two write operations from independent transactions must not be allowed to simultaneously update different copies of the object. In order to achieve this synchronization between multiple copies, possibly residing at different sites, additional communications and processing cost is incurred as compared to the case in which only one copy of an object exists.

1.1 Fault Tolerance in Distributed Mutual Exclusion

The mutual exclusion problem was originally considered in centralized systems for the synchronization of exclusive access to the shared resource. In the problem of mutual

exclusion, concurrent access to a shared resource or the Critical Section (CS) must be synchronized such that at any time only one process can access the CS. In a distributed system, due to the lack of both in shared memory and a global clock, and due to unpredictable message delay, the design of a distributed mutual exclusion protocol that is free from deadlock and starvation, is much more complex than that in a centralized system.

Over the past decade, many protocols have been proposed to achieve mutual exclusion in distributed systems. These protocol can be classified into two classes: *token-based* and *non-token-based* [38] (or *permission-based* [25]). In *token-based* protocols [27, 29, 31, 36, 39], only the node holding the token can execute the CS. In *non-token-based* protocols [6, 22, 24, 32, 33, 37], a requesting node can execute the CS only after it has received permission from each member of a subset of nodes in the system. In general, token-based protocols suffer from poor failure resilience. In particular, if the node holding the token fails, a complexity token regeneration protocol must be executed, and then a recovery phase must be reinitiated to reconfigure the whole network [26]. On the other hand, for these non-token-based protocols [6, 22, 24, 32, 33, 37], when any node in the subset of nodes from which one requesting node must get permission fails, the requesting node cannot enter the CS. Therefore, although the primary node protocol [4] requires only 3 messages and Maekawa's protocol [24] requires $O(\sqrt{N})$ messages, they are vulnerable to node and communication failures, where N is the number of nodes in the system.

To make distributed mutual exclusion protocols fault-tolerant to node and communication failures, many researches apply the replica control strategies to achieve mutual exclusion. The majority voting protocol [40] and the quorum consensus protocol [15] are such examples. However, these protocols require high communication cost which is $O(N)$ due to the large quorum size. Although the size of a quorum can be reduced by using unequal votes [15], the resulting protocol is not a fully distributed one because the nodes with higher weight bear more responsibility in controlling mutual exclusion than do others.

1.2 Motivations

To reduce the overhead of achieving mutual exclusion while supporting fault tolerance, many protocols imposing a logical structure on the network are proposed [3, 10, 20]. The hierarchical quorum consensus protocol (HQC) [20] requires $O(N^{0.63})$ messages, the tree quorum protocol [3] requires $O(\log N)$ messages in the best case and degrades gracefully, and the grid protocol [10] requires $O(\sqrt{N})$ messages. All the quorums constructed from these protocols can be used to replace the set of nodes in Maekawa's protocol [24] to achieve mutual exclusion. (Note that in Maekawa's protocol, only one set is associated with each node, which makes the protocol non-tolerant to failures. While these three protocols [3, 10, 20] can support a node with several alternative sets.)

In the thesis, we use a *triangular mesh* approach to develop protocols for distributed mutual exclusion, in which the nodes in the system are organized into a triangular mesh. We propose three protocols based on the triangular mesh: the *triangular mesh* protocol, the *triple triangular mesh* protocol and the *dynamic triangular mesh* protocol. In the triangular mesh protocol, we associate with each node two special patterns. The nodes in each special pattern constitute a quorum. In the triple triangular mesh protocol, a quorum contains nodes from some side of each of three subtriangles in the triangular mesh. In the dynamic triangular mesh protocol, three dynamic paths constitute a quorum in the given triangular mesh. The quorum size of these proposed protocols is K that is $O(\sqrt{N})$, where $N = \frac{K*(K+1)}{2}$ is the number of nodes in the system. In the worst case, the triangular mesh protocol, the triple triangular mesh protocol and the dynamic triangular mesh protocol are fault-tolerant up to $(K - 2)$, $(K - 2)$ and $(K - 1)$ site and communication failures, respectively. From our simulation study, these three proposed protocol can have higher availability than Cheung et al.'s grid protocol. Moreover, the quorum size of the proposed protocols will be less than that in Kumar's HQC protocol when N is greater than or equal to 15 and less than that in Agrawal et al.'s tree quorum protocol when node failures of some sort exist.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we give a survey of replica control protocols. This does help since the problem of distributed mutual exclusion can be considered as a special case of the replica control problem. Next, we present the three proposed triangular-mesh-based protocols, i.e., the triangular mesh protocol, the triple triangular mesh protocol and the dynamic triangular mesh protocol in Chapter 3, Chapter 4 and Chapter 5, respectively. In Chapter 6, we study the performance, including availability, quorum size, and fault-tolerance of these proposed protocols. We also make a comparison of the proposed protocols with the other three protocols which also impose logical structures on the network, including the tree, the hierarchical quorum consensus and the grid protocol. Finally, Chapter 7 gives a summary and points out some future research directions.

CHAPTER II

A Survey of Replica Control Protocols

In this Chapter, we present the system model and a survey of replica control protocols including conventional ones and those protocols imposing logical structures. The conventional methods include majority voting, quorum consensus, coterie and dynamic voting, while the protocols imposing logical structures include tree, grid and hierarchical protocols.

2.1 The System Model

A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network.

We assume that site failures are fail-stop, i.e., nodes stop executing without performing any incorrect actions and that node crashes are detectable by other nodes. We do not consider Byzantine failures [30] where sites may act in an arbitrary and malicious manner. Communication links may fail to deliver messages, too. Combinations of such failures may lead to *partitioning failures*, where sites in a *partition* may communicate with each other, but no communication can occur between sites in different partitions. A site may become inaccessible due to site or partitioning failures.

A *distributed database* consists of a set of *objects* stored at several sites in a computer network. Users interact with the database by invoking *transactions*. A transaction is a partially ordered sequence of read and write operations that are executed atomically. The execution of a transaction must appear atomic, i.e., a transaction either *commits* or *aborts*. A commonly accepted correctness criteria in databases is the *serializable* execution of transactions. The serializable execution is guaranteed by employing a *concurrency*

control mechanism, e.g., locking, timestamp ordering, or optimistic concurrency control protocols.

In a replicated database, copies of an object may be stored at several sites in the network. Multiple copies of an object must appear as a single logical object to the transactions. This is termed as *one-copy equivalence* and is enforced by a *replica control* protocol. The correctness criteria for replicated databases is *one-copy serializability*, which ensures both one-copy equivalence and the serializable execution of transactions.

We now formally define the *replica control problem*. In order to ensure one-copy equivalence, each object x has associated with it a read quorum and a write quorum. A read operation is executed by accessing copies that constitute a read quorum, and then by reading a copy with the highest version number. A write operation is executed by writing the copies in a write quorum, and assigning them a version number that is one more than the maximum encountered in the write quorum. To ensure one-copy equivalence the read and write quorums must satisfy the following two conditions:

1. **Write-write Intersection Property.** If g, h are two write quorums, then they must have a nonempty intersection, i.e., $g \cap h \neq \phi$.
2. **Read-write Intersection Property.** If g is a write quorum and h is a read quorum, then they must have a nonempty intersection, i.e., $g \cap h \neq \phi$.

The problem of mutual exclusion is a special case of the replica control problem. Consider an object x , where no distinction is made between read and write operations. In this case any quorum may be used by a read or a write operation, and trivially both the read-write and write-write conditions of replica control are satisfied.

2.2 Conventional Replica Control Protocols

In this section, we briefly describe the conventional replica control protocols such as majority voting [40], quorum consensus [15], coterie [13] and dynamic voting [19].

2.2.1 Majority Voting

Voting is a common technique to ensure that only a single group of nodes in a faulty distributed system performs some restricted operation. For example, voting can be used to manage replicated data [15, 40]. In this application, a network partition can break the system into isolated groups. If these groups were allowed to independently modify their copies of the data, the data would diverge and inconsistencies would arise. Votes can be used to avoid this. Since at most one group can have a majority, only one group at a time will update the database. Voting can also be used in other applications like coordinator election [13] and modular redundancy systems for fault tolerant computing [35].

Although voting guarantees that there will never be more than one group performing the restricted operation at the same time, it cannot guarantee that there will be an active group. For example, consider a system with three nodes a, b , and c , where each node has a single vote. If a partition terminates communications between all pairs of nodes, then no group will have a majority of two votes, and no node will perform the operation (e.g., update the database). Similarly, if nodes a and b crash, node c will be in the same situation. We call a system state *halted* if the restricted operation cannot be performed by any node.

The assignment of votes or the choice of set of groups can have a critical effect on the reliability of a distributed system [13]. Consider, for example, a system with nodes a, b, c and d and an assignment that gives one vote to each. This seems like a natural choice because it gives each node equal weight. Since three votes are needed for a majority, this is equivalent to the set of groups $S = \{\{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}\}$. But now, consider an assignment that gives node a two votes and the rest a single vote. The majority is still three votes, so this is equivalent to $R = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c, d\}\}$. Set R and its associated vote assignment is clearly superior to S because all groups of nodes that can operate under S can operate under R , but not vice versa. For instance, a and b can form a group under R but not under S . So, if the system splits into group $\{a, b\}$ and $\{c, d\}$, there will be one active group under R but none under S . So clearly, no system designer should *ever* select set S (or its equivalent vote assignment), in spite of the fact it seems "natural."

We use the terms "*R dominates S*" to mean that *R* is always superior to *S*. Obviously, we want to ignore dominated sets (or vote assignments).

2.2.2 Quorum Consensus

Many distributed mutual exclusion algorithms are based on the concepts of quorums. A quorum is defined as a set of nodes. To access a shared resource, permissions from a quorum of nodes should be obtained. Exclusive access of the shared resources is guaranteed by requiring that any two quorums contain at least one common node. In this way, for any two attempts to access the shared resource, at least one node detects these two attempts and thus at least one node can schedule these two attempts to ensure the exclusive use of the shared resource.

The quorum consensus algorithm describes how access by read and write operations to n copies of a replicated object can be synchronized. A read operation must lock q_r copies, and a write must lock q_w copies, such that

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

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

2.2.3 Coterie

A coterie, the general notion of quorums proposed by Lamport [23] as well as Garcia-Monlina and Barbara [13], is a set of sets with the property that any two members of a coterie have a nonempty intersection.

In [13], the following definitions and theorems about coteries are provided.

Definition 2.1 *Coterie: Let U be the set of nodes that compose the system. A set of groups S is a coterie under U iff*

- $G \in S$ implies that $G \neq \phi$, and $G \subseteq U$.
- **Intersection property:** IF $G, H \in S$, then G and H must have at least one common node.
- **Minimality:** There are no $G, H \in S$ such that $G \subset H$.

Note that not all nodes must appear in a coterie. For instance $\{\{a\}\}$ is a coterie under $\{a, b, c\}$.

Definition 2.2 *Domination for Coteries: Let R, S be coteries (under U). R dominates S iff $R \neq S$ and, for each $H \in S$, there is a $G \in R$ such that $G \subseteq H$. (We say that G is the group that dominates H).*

Definition 2.3 *Dominated and Non-dominated Coteries: A coterie S (under U) is dominated iff there is another coterie (under U) which dominates S . If there is no such coterie, then S is non-dominated (ND).*

A dominated coterie should *not* be used because there is a coterie that provides more protection against partitions [13] and incurs less communication overhead. Moreover, non-dominated coteries are more resilient to network and site failures than dominated ones; that is, the availability and reliability of a distributed system is better if ND coteries are used [28]. For instance, the coterie $\{\{a, b, c\}, \{c, d, e\}\}$ should be replaced by $\{\{c\}\}$, and the coterie $\{\{a, b\}, \{b, c\}\}$ should be replaced by $\{\{a, b\}, \{a, c\}, \{b, c\}\}$ or by $\{\{b\}\}$. To

check if a coterie is ND without enumerating all other coteries, we can apply the following theorem [13]:

Theorem 2.1 *Let S be a coterie under U . S is dominated iff there exists a group $G \subseteq U$ such that*

1. G is not a superset of any group in S .
2. G has the intersection property. That is, $\forall H \in S, G \cap H \neq \phi$.

Coteries have advantages over the voting scheme since there are ND coteries such that no vote assignments corresponds to them. For example, the ND coterie $S_1 = \{\{a, b\}, \{a, c, d\}, \{a, c, e\}, \{a, d, f\}, \{a, e, f\}, \{b, c, f\}, \{b, d, e\}\}$ cannot be represented by votes. To see this, one can set up inequalities for the constraints that votes must satisfy (e.g., $v_i(a) + v_i(b) \geq \text{MAJ}(v_i)$, where $\text{MAJ}(v_i)$ is the vote majority in v_i). It is then simple to see that the inequalities have no solution. Moreover, the total number of vote assignments is of the order of $O(2^{N^2})$, and total number of set of groups is of the order of $O(2^{cN})$, where N is the number of nodes in the system and c is a constant. As you can see, vote assignments yield only a very small portion of the ND coteries, thus coteries are a more powerful concept than vote assignments. However, it can be shown [13] that in systems with five or less nodes, all ND coteries have corresponding vote assignments. In other words, with five or less nodes, coteries and votes are equivalent concepts, and only with six or more nodes, are there coteries that have no vote assignment.

A second advantage of coteries is that they are easier to enumerate because they have no duplicates (i.e., all duplicate vote assignment correspond to the ND vote assignments) for systems with five or less nodes is *extremely small*. It seems counterintuitive, but for four nodes there are *only three* basic choices that make sense: give one node all votes; give three nodes a single vote and the fourth none; or give one node two votes and the rest one. Considering permutations, this becomes a total of 12 choices. (Contrast with the $2^{15} - 1$ or 32,767 possible sets of groups for a 4-node system.) No other cases need be considered for an optimization procedure. Even for five nodes, there are only seven basic cases, or 131

including permutations (which is much smaller than the more than 2×10^9 possible sets of groups).

Disadvantages also exist. Voting is appealing because it is flexible and can be easily implemented. In contrast, in a system that uses coterie, operations must know all the groups of the coterie and test if the nodes that responded positively to its request form a group of the coterie. In a coterie-based system, adding and removing a node may cause the addition and deletion of numerous groups [8].

Toward the general coterie design, Garcia-Molina and Barbara defined a domination relation between coterie in order to narrow down the solution space considered by the system designer. Garcia-Molina and Barbara [13] developed a transformation algorithm to enumerate ND coterie for small systems. By this enumeration, some objective functions may be maximized. However, the TA proposed in [13] is not appropriate for the case of large coterie design due to the unexpected properties of the new produced ND coterie. The partial enumeration algorithm proposed in [13] is also not appropriate due to its high computation complexities.

Neilsen and Mizuno propose a join algorithm [28], which takes nonempty coterie, as input, and returns a new, larger coterie. The new coterie is called a *composite coterie*. They prove that a composite coterie is nondominated if and only if the input coterie are nondominated.

In [17], by assigning a Boolean variable to each element in U , Ibaraki and Kameda represent a family of subsets of U by a Boolean function of these variables. They characterize the ND coterie as exactly those families which can be represented by positive, self-dual functions. In this Boolean framework, they prove that any function representing an ND coterie can be decomposed into copies of the three-majority function, and this decomposition is representable as a binary tree.

In [34], Shou and Wang provide a simple way to produce new larger ND coterie through transformation from smaller ND coterie. By systematically applying our transformation, coterie can scale well for a large number of nodes.

The problem of enumerating coterie so that performance can be optimized by choosing the best one has been addressed by several researchers. In [13], an algorithm is described that can be used to generate a subset of the mutual exclusion coterie which include all coterie obtained from vote assignments. The authors presented an algorithm in [7] to generate all vote and quorum assignments that need to be considered in optimizing reading and writing of replicated data. The method presented in [21] generates a subset of the mutual exclusion coterie obtained from vote assignments.

2.2.4 Dynamic Voting

The quorum-based methods stated above are all *static*; that is, those coterie adopted are not changed according to the state of the system. In contrast, *dynamic* schemes use the state information to determine the best quorum set of to reconfigure the logical structure, and adapt invocation algorithms such that the availability of the system can be improved [5, 12, 16, 19].

The weakness of majority consensus voting and of all other static voting protocols is that the quorum is fixed—it can not change once the system has begun operation. Because of this, a few failures can render the data inaccessible. Dynamic voting [11] is a consistency and recovery control algorithm for replicated objects tailored to environments susceptible to site failures and network partitions. Dynamic voting algorithms are especially interesting because they provide high reliability and availability while handling network partitions correctly. This policy adjusts the necessary quorum of physical copies required for an access operation without manual intervention. A group of physical copies, comprised of a majority of the current physical copies that can communicate among themselves, is referred to as the *majority block*. Their implementation is complicated by the requirement of instantaneous state information, which is costly in terms of network traffic.

Basically, dynamic voting is based on the concept of the connection vector. The *connection vector* instantaneously records the state of the network with respect to all sites. Each physical copy of a replicated data object has an associated ensemble of state information consisting of the version number and partition vector. The version number of a physical

copy represents the number of successful write operations to the file that are known to the physical copy. The partition vector at site records the version numbers of all sites as they were most recently received by that site.

In its original form dynamic voting allows accesses to proceed so long as a strict majority of the current physical copies are accessible. In situations where the number of current physical copies within a group of mutually communicating sites is equal to the number of current copies not in communication, dynamic voting cannot proceed and declares the replicated file to be inaccessible.

The basis of core protocol is the algorithm for detecting whether the access request is originating within the majority partition. Since there can be only one majority partition, mutual exclusion is guaranteed and consistency is preserved.

The advantages of the dynamic voting scheme over the static one are summarized as follows:

1. The number of sites necessary for an update is a function of the number of current copies in existence at the time of the update.
2. Thus required quorum changes dynamically without any manual intervention
3. Thus required quorum changes dynamically (without any manual intervention) according to changes in the configuration that are due to site and/or communication link failures.
4. A file can be updated in a partition if it contains more than half of the current copies rather than half of the all copies.
5. Simple to state as well as implement.
6. It does not require a complicated message-based coordination mechanism.

2.3 Quorum Protocols Imposing Logical Structures

Conventional algorithms for managing replicated data, such as *majority voting* [40] and *quorum consensus* [15], require that a majority of data copies be accessed for write access. Variations [18] of the quorum consensus algorithm have been proposed to improve the availability of the quorum consensus protocol. The advantages of these algorithms are that they offer very high availability. However, the data accessing cost is very high because the quorum size is $O(N)$, where N is the number of data copies.

Recently, many algorithms have been proposed to exploit a *logical structuring* of data copies to reduce the quorum size while keeping high availability. Recent research on management of replicated data has focused on organizing multiple copies of an object into a logical structure [2] and exploiting it to obtain smaller quorum sizes than is possible with conventional methods. Examples of techniques based on logical structure are: tree protocol [1], grid protocol [9] and hierarchical quorum consensus [20]. In this section, we give a brief description of these protocols imposing logical structures.

2.3.1 Tree Protocol

In [1], it is proposed that the n copies of an object be logically organized to form a complete binary tree; i.e., if k is the level of the tree, then it has $2^{k+1} - 1$ copies, where the root is at level 0. The standard tree terminology, i.e., root, child, parent, leaf, etc., is used. A path in the tree is defined to be a sequence of copied $s_1, s_2, \dots, s_i, s_{i+1}, \dots, s_n$ such that s_{i+1} is a child of s_i .

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 copies constitutes a quorum. If a path cannot be constructed due to the inaccessibility of a copy, c , residing on a failed or inaccessible site (due to partitioning failures), then the algorithm must substitute for that copy with two paths, both of which start with the children of copy c and terminate with leaves. Note that each path must terminate with a leaf, hence if the last copy in the path is inaccessible, the

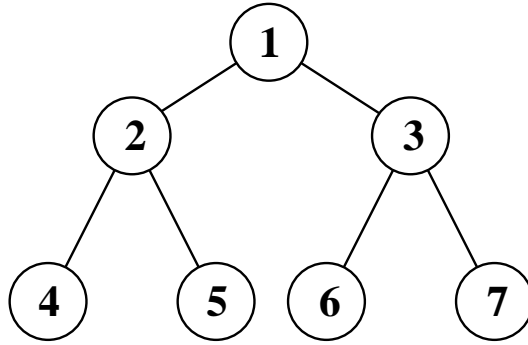


Figure 2.1 A complete binary tree of seven copies of a replicated object

operation must be aborted. It is shown that in the best case the size of the quorum is $\log n$. This case arises when there are no failures or under certain favorable failure distributions. In the worst case, the size of the quorum increases to $\lceil n/2 \rceil$. This situation occurs, for example, when all interior nodes of the tree are inaccessible. In Figure 2.1, 7 copies of a replicated object are organized into a complete binary tree. When there is no inaccessible sites, the sets of nodes, $\{1, 2, 4\}$, $\{1, 2, 5\}$, $\{1, 3, 6\}$ and $\{1, 3, 7\}$ are examples of valid quorums. If the root is inaccessible, the sets of nodes, $\{2, 4, 3, 6\}$ and $\{2, 5, 3, 6\}$ are examples of valid quorums. Now, if node 1, 2, and 3 are inaccessible, the set of node $\{4, 5, 6, 7\}$ is a valid quorum.

A simple extension of this protocol for read and write access requires that both read and write operations obtain quorums using the tree protocol. Thus, the sizes of the read and write quorums range from $\log n$ to $\lceil n/2 \rceil$. However, this scheme has the undesirable property that there is unnecessary redundancy since two read operations always have a nonempty intersection. Thus, the efficiency of this solution is at the expense of unnecessary redundancy for read operations.

The availability of quorums in the tree protocol is also sensitive to failure patterns. In the worst case, the failure of $\log n$ sites, which form a path in the tree, prevent the formation of the quorum. On the other hand, if every site except the above $\log n$ sites fail, a quorum can still be constructed. Thus in the best case the protocol can tolerate $n - \log n$ failures.

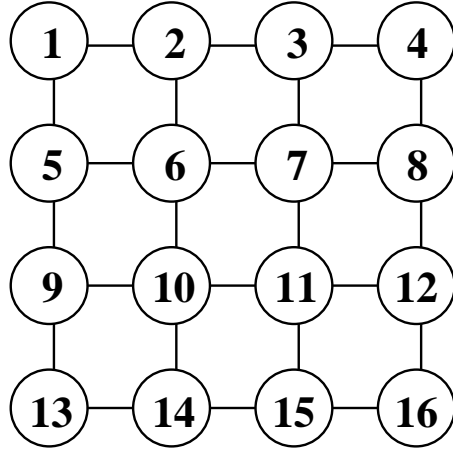


Figure 2.2 A 4×4 grid in the grid protocol

2.3.2 Grid Protocol

The grid protocol [10] logically arranges the nodes in the system into a grid network. The protocol also assumes that node failures are fail-stop. A read quorum group contains exactly one node from each column. A write quorum group consists of nodes in a read group and all nodes in a column of the grid. Figure 2.2 shows a 4×4 grid. The sets of nodes $\{ 5, 6, 11, 16 \}$ and $\{ 2, 6, 10, 14, 9, 7, 16 \}$ constitute a read and a write group, respectively. The grid topology is only a conceptual tool used to describe the protocol. The availability of quorums in the grid protocol is sensitive to failure patterns. In particular, if one column is not accessible, neither read nor write quorum can be formed. If we consider a $\sqrt{N} \times \sqrt{N}$ grid, the size of read and write quorum is \sqrt{N} and $2\sqrt{N} - 1$, respectively. In the worst case, read and write operations are resilient to $\sqrt{N} - 1$ failures. In the best case, read and write operations are resilient to $N - \sqrt{N}$ and $N - 2\sqrt{N} + 1$ failures.

2.3.3 Hierarchical Quorum Consensus

The hierarchical quorum consensus protocol [20] logically organizes a set of copies of an object in a database into a multilevel tree (of depth m) with the root as level 0. The physical copies of an object are stored only in the leaves of this tree or at level m , while the higher level nodes of the tree correspond to logical groups. A node at level i , where i varies

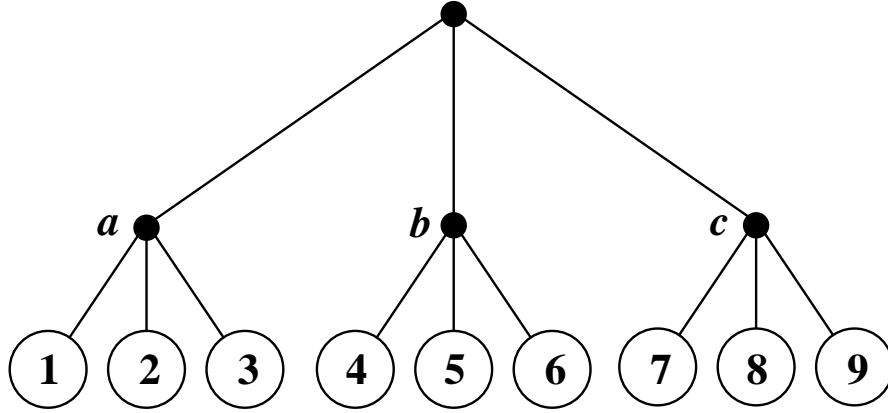


Figure 2.3 Organizes 9 copies into a 2-level hierarchy

from 0 to $m - 1$, is viewed as a logical group which in turn consists of l_{i+1} subgroups at level $i + 1$. A quorum is associated with each level and to access a logical group at a certain level, a quorum consisting of its subgroups must be first assembled. A read (write) quorum at level i is defined as the number of subgroups of a level $i - 1$ group that must be locked by a read (write) operation to obtain read (write) access to the group. The read (write) quorum at level i is denoted by $r_i(w_i)$. Note that this is a recursive definition. Therefore, each level i group must in turn assemble r_{i+1} of its subgroups at level $i + 1$, and so on. This would eventually translate into a quorum consisting of physical copies of the object. For $i = 1, \dots, m, r_i, w_i$, and l_i must satisfy the following constraints:

$$r_i + w_i > l_i, \text{ and} \tag{2.1}$$

$$2 * w_i > l_i. \tag{2.2}$$

To perform read (write) operations on the replicated object, a read (write) quorum at level 0 must be obtained first. In Figure 2.3, 9 copies of a replicated object are organized into a 2-level hierarchy with $l_1 = l_2 = 3$, and $r_1 = r_2 = w_1 = w_2 = 2$. The set of nodes $\{ 1, 2, 5, 6 \}$ is a valid quorum since the set $\{ 1, 2 \}$ and $\{ 5, 6 \}$ constitute a quorum at node a and b , respectively. The set of nodes $\{ a, b \}$ in turn constitutes a quorum at root. Also, the set of nodes $\{ 1, 2, 7, 9 \}$ and $\{ 5, 6, 7, 8 \}$ are valid quorums.

The quorum size of this protocol is $N^{0.63}$. Contrast this with the majority consensus method where the quorum size is $\lceil \frac{N+1}{2} \rceil$. It is evident that the quorum size in the hierarchical quorum consensus protocol will grow at a much slower rate with respect to N than in the majority voting method.

CHAPTER III

The Triangular Mesh Protocol

In this Chapter, we present the triangular mesh (TM) protocol. We first give the definitions and examples of TM quorums. Next, we describe the algorithm for constructing a quorum of the triangular mesh protocol. Then, we present a proof of correctness of the proposed protocol. Finally, several properties of the protocol are described.

3.1 Definitions

We organize nodes into a triangular mesh. A k -triangular mesh consists of a vertex set V and an edge set E . V is defined as a set of (x, y) -tuples, where x, y are both integers, $0 \leq x \leq k - 1, 0 \leq y \leq k - 1$ and $0 \leq x + y \leq k - 1$. The y -axis is slanted to the right 30 degree to accommodate the left-hand side of the right triangle. E is defined as a set of vertex pairs (v_1, v_2) , where v_1 and v_2 are in V , $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$ and x_1, y_1, x_2 and y_2 satisfy one of the following conditions:

$$\begin{aligned}x_1 - x_2 &= y_2 - y_1 = 1, \\x_2 - x_1 &= y_1 - y_2 = 1, \\|x_1 - x_2| + |y_1 - y_2| &= 1.\end{aligned}$$

A 6-triangular mesh is shown in Figure 3.1. In this example, node 3 is at $(0, 3)$ and node 13 is at $(3, 1)$. The left-hand, right-hand and bottom sides of the right triangle are referred to as side 0, side 1 and side 2, respectively.

Definition 3.1 Each quorum consists of a center and subquorum i , for $i = 0, 1, 2$. Subquorum i consists of nodes of a subcolumn parallel with a side determined by the type of quorum, starting at the center and ending at side i .

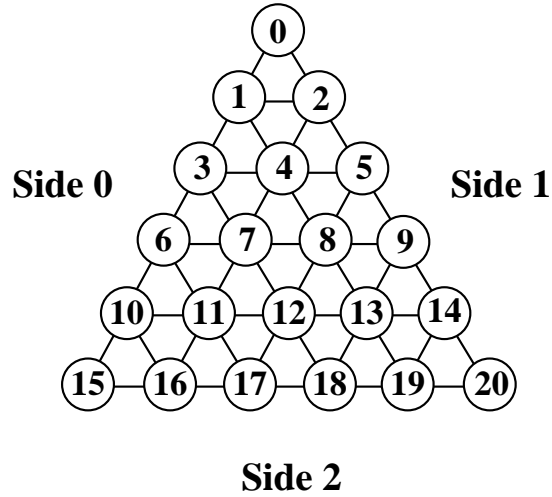


Figure 3.1 A 6-triangular mesh

Definition 3.2 *The subquorum i of a type-1 quorum is parallel with side $((i + 1) \bmod 3)$, for $i = 0, 1, 2$.*

Example 3.1 In Figure 3.2-(a), the set $\{ 7, 3, 8, 9, 11, 16 \}$ constitutes a quorum whose center is 7. In Figure 3.2-(b), a quorum consists of the set $\{ 9, 5, 2, 0, 13, 18 \}$, with node 9 as its center, and *subquorum 1* of this quorum contains the center only. Also, the set $\{ 15, 16, 17, 18, 19, 20 \}$ is a valid quorum whose center is node 15, and *subquorum 0* and *2* contain the center only.

Definition 3.3 *The subquorum i of a type-2 quorum is parallel with side $((i + 2) \bmod 3)$, for $i = 0, 1, 2$.*

Example 3.2 In Figure 3.3-(a), the set $\{ 8, 7, 6, 5, 13, 19 \}$ constitutes a quorum whose center is 8. In Figure 3.3-(b), a quorum consists of the set $\{ 18, 17, 16, 15, 13, 9 \}$, with node 18 as its center, and *subquorum 2* of this quorum contains the center only. Also, the set $\{ 20, 19, 18, 17, 16, 15 \}$ is a valid quorum whose center is node 20, and *subquorum 1* and *2* contain the center only.

We can describe the quorum defined above in another way. A quorum consists of a center and subquorum i , for $i = 0, 1, 2$. Subquorum i is a sequence of nodes, (v_0, \dots, v_m) , where v_0 is the center of the quorum, and v_m is the ending node of the subquorum. For any

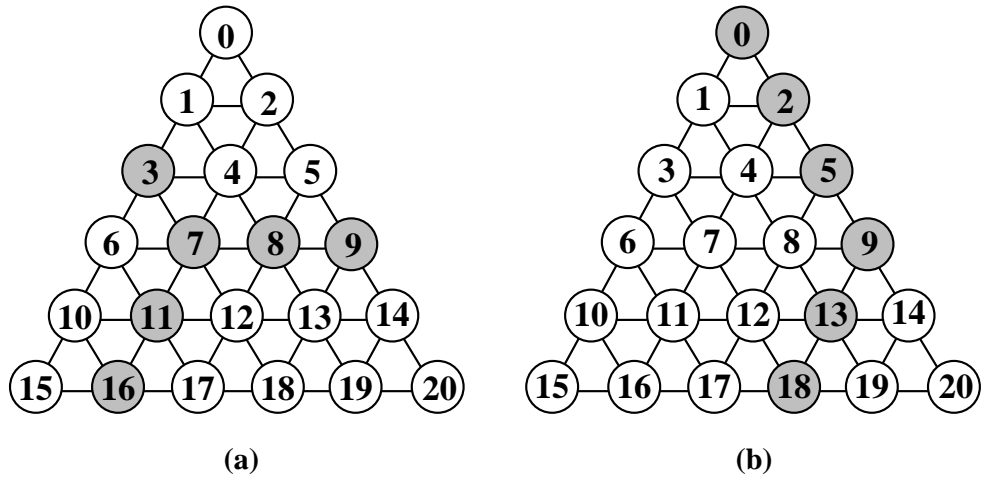


Figure 3.2 Examples of type-1 quorums in TM: (a) a type-1 quorum in a 6-triangular mesh; (b) a type-1 quorum whose subquorum 1 contains only the center.

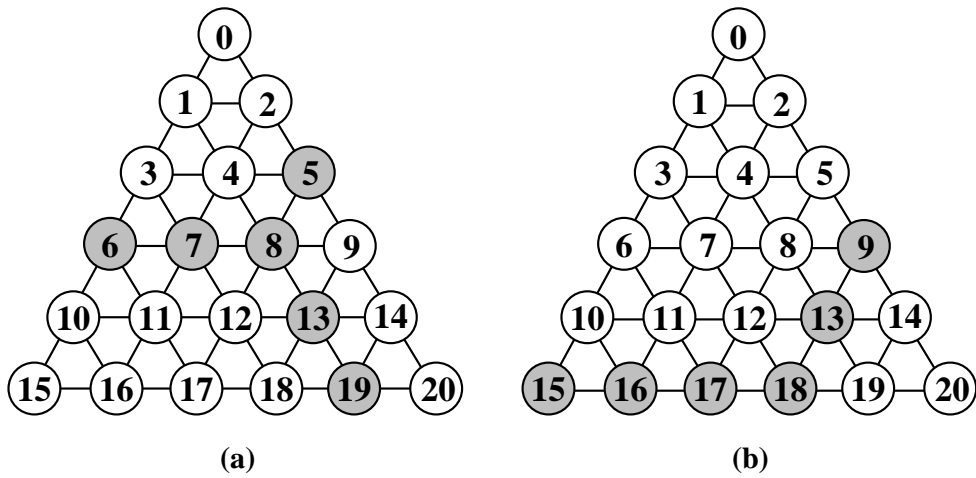


Figure 3.3 Examples of type-2 quorums in TM: (a) a type-2 quorum in a 6-triangular mesh; (b) a type-2 quorum whose subquorum 2 contains only the center.

two adjacent nodes v_j, v_{j+1} , for $j = 0, \dots, m - 1$, they must satisfy one of the following conditions according to i :

1. For type-1 quorum:

- (a) $x_{j+1} - x_j = -1$ and $y_{j+1} - y_j = 1$, when $i = 0$;
- (b) $x_{j+1} - x_j = 1$ and $y_{j+1} = y_j$, when $i = 1$;
- (c) $x_{j+1} = x_j$ and $y_{j+1} - y_j = -1$, when $i = 2$.

2. For type-2 quorum:

- (a) $x_{j+1} - x_j = -1$ and $y_{j+1} = y_j$, when $i = 0$;
- (b) $x_{j+1} = x_j$ and $y_{j+1} - y_j = 1$, when $i = 1$;
- (c) $x_{j+1} - x_j = 1$ and $y_{j+1} - y_j = -1$, when $i = 2$.

Also, we will call a quorum "associated with" node x if it uses node x as its center.

3.2 The Algorithm for Constructing Quorums

In our algorithm to construct a quorum, each node is associated with two kinds of information: *node status* and *quorum status*. *Node status* indicates whether a node is *Available*, *Unavailable* or *Unknown*. *Quorum status* shows that whether it is possible to construct a type-1 or type-2 quorum using a given node as its center. Initially, we set node status to be *Unknown*, and quorum status to be *Type-1_Maybe_Available*, *Type-2_Maybe_Available* for all nodes. The set *node_of_quorum* is used to record the nodes which grant the request and are included in the quorum being constructed. The set *node_granted* is used to record the nodes granting the request but not belonging to *node_of_quorum*. First, we try to construct a type-1 quorum associated with the requesting node. If one failed node is encountered, we find those nodes which will contain the failed node in their type-1 quorum and mark these nodes as *Type-1_Unavailable*, since the failed node will prevent the construction of type-1 quorums, which is proved in lemmas 1 and 2 in Section 3.4. Also, we find those nodes which will contain the failed node in their type-2 quorum and mark

these nodes *Type-2_Unavailable*. Next, if there exists any node in *node_granted* which is *Type-1_Maybe_Available*, we try to construct its associated type-1 quorum which will contain the largest fraction of *node_granted*. If it fails, all nodes in *node_granted* are *type-1_Unavailable*. Then, we try to construct a type-2 quorum in the similar way. If it fails again, we will find whether there exists a node which has *node_status* equal to *Unknown* and *quorum_status* equal to *Type-1_Maybe_Available* or *Type-2_Maybe_Available*. If such a node exists, we repeat the above steps again to construct a type-1 or type-2 quorum. When all of the cases fail, there is no type-1 or type-2 quorum available. (Note that in Section 4.3, we show one of patterns of failed nodes which prevent the construction of any quorum). Figure 3.4 and 3.5 show the flowcharts for procedure *construct_quorum* and *construct_subquorums*, respectively.

Algorithm for the Triangular Mesh Protocol

```

procedure initialize_status;
begin
  for each node (x, y) in the system do
    set_status(x, y, Unknown, Type-1_Maybe_available, Type-2_Maybe_available);
    empty the lists: node_of_quorum, node_granted;
end; (* initialize_status *)

procedure construct_subquorums(quorum_type: integer, center): boolean;
var subquorum: integer; (* indicate which subquorum is being constructed *)
    subquorum_finished: boolean;
begin
  for subquorum = 0 to 2 do
    begin
      set (i, j) to center.
      subquorum_finished = FALSE;
      repeat
        if (query_status(i, j, Unknown) == TRUE) then
          begin
            if(request(i, j) == GRANTED) then
              begin
                set_status(i, j, Available);
                add node (i, j) into node_of_quorum;
                if(query_status(i, j, Ending_Point, subquorum) == TRUE) then
                  subquorum_finished = TRUE
                else set (i, j) to the next node according to subquorum and quorum_type;
              end
            end
            else (* request is denied by (i, j) *)
              begin
                set_status(i, j, Unavailable);
                for all nodes (x, y) belonging to type-2 quorum whose center is (i, j) do
                  set_status(x, y, Type-1_Unavailable);
                for all nodes (x, y) belonging to type-1 quorum whose center is (i, j) do
                  set_status(x, y, Type-2_Unavailable);
                append the content of node_of_quorum to node_granted;
                empty node_of_quorum;
                return (FALSE);
              end
            end
          end
        end
      end
    end
  end

```

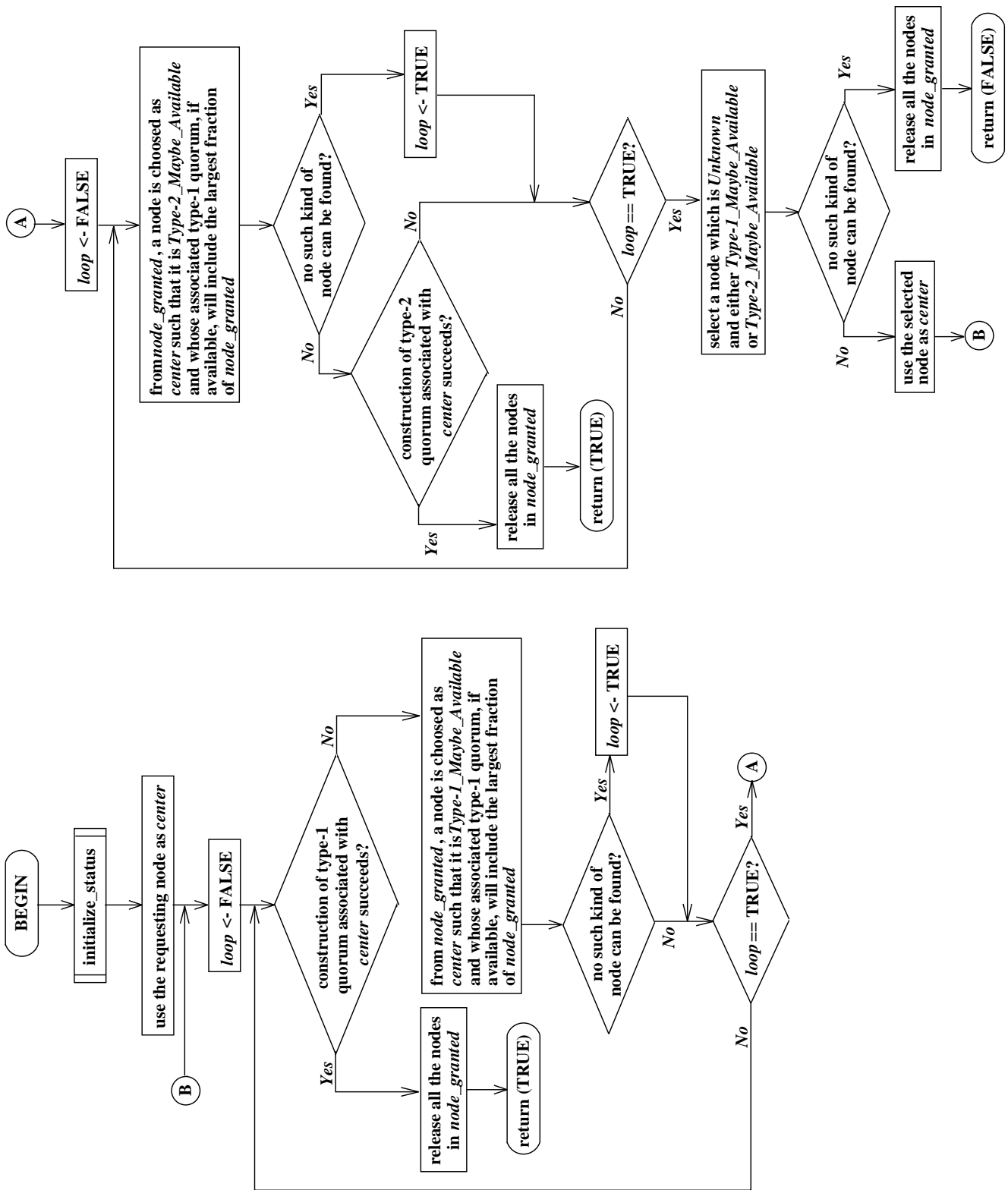


Figure 3.4 The flowchart for procedure *construct_quorum* in TM

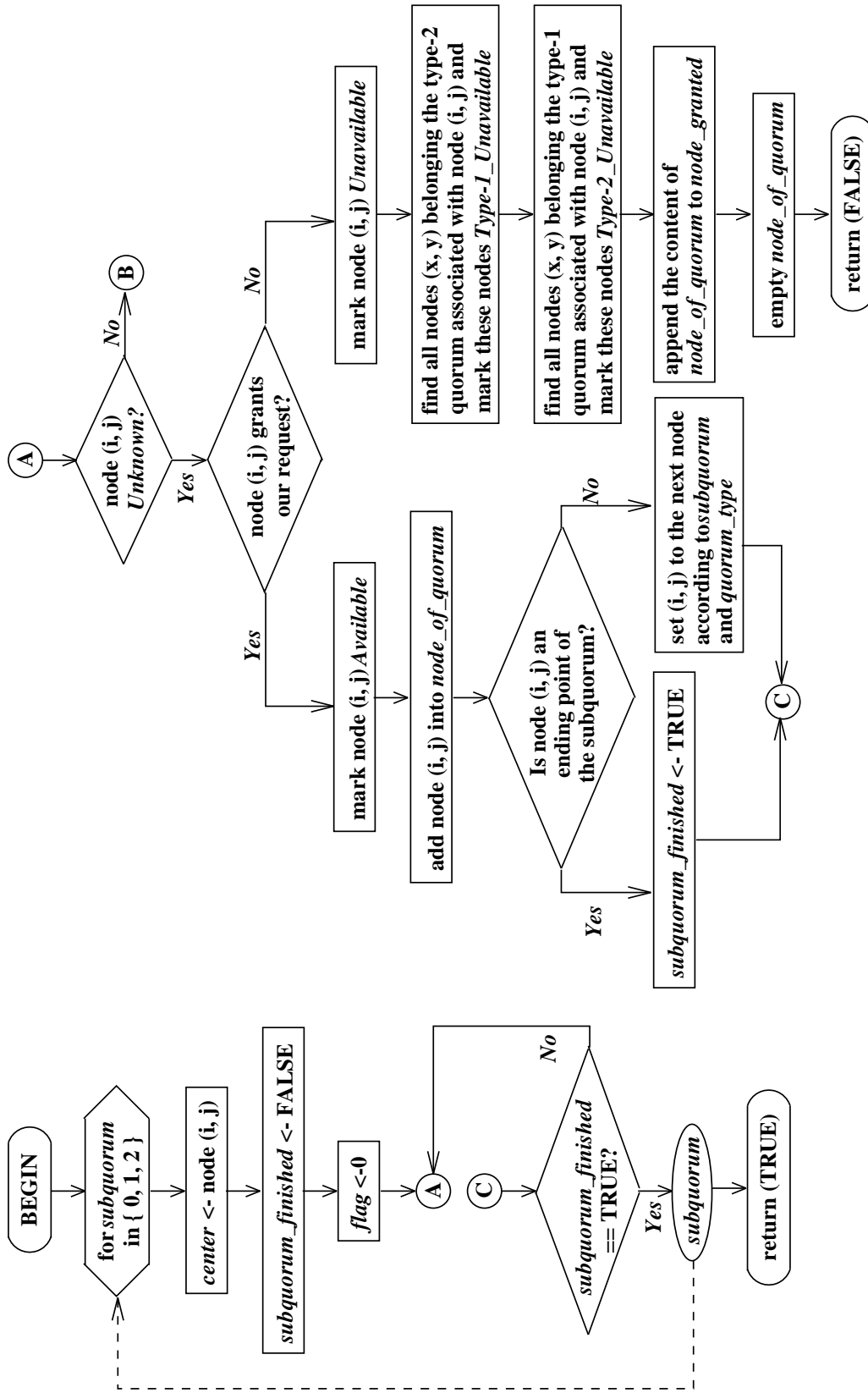


Figure 3.5 The flowchart for procedure *construct_subquorums* in TM

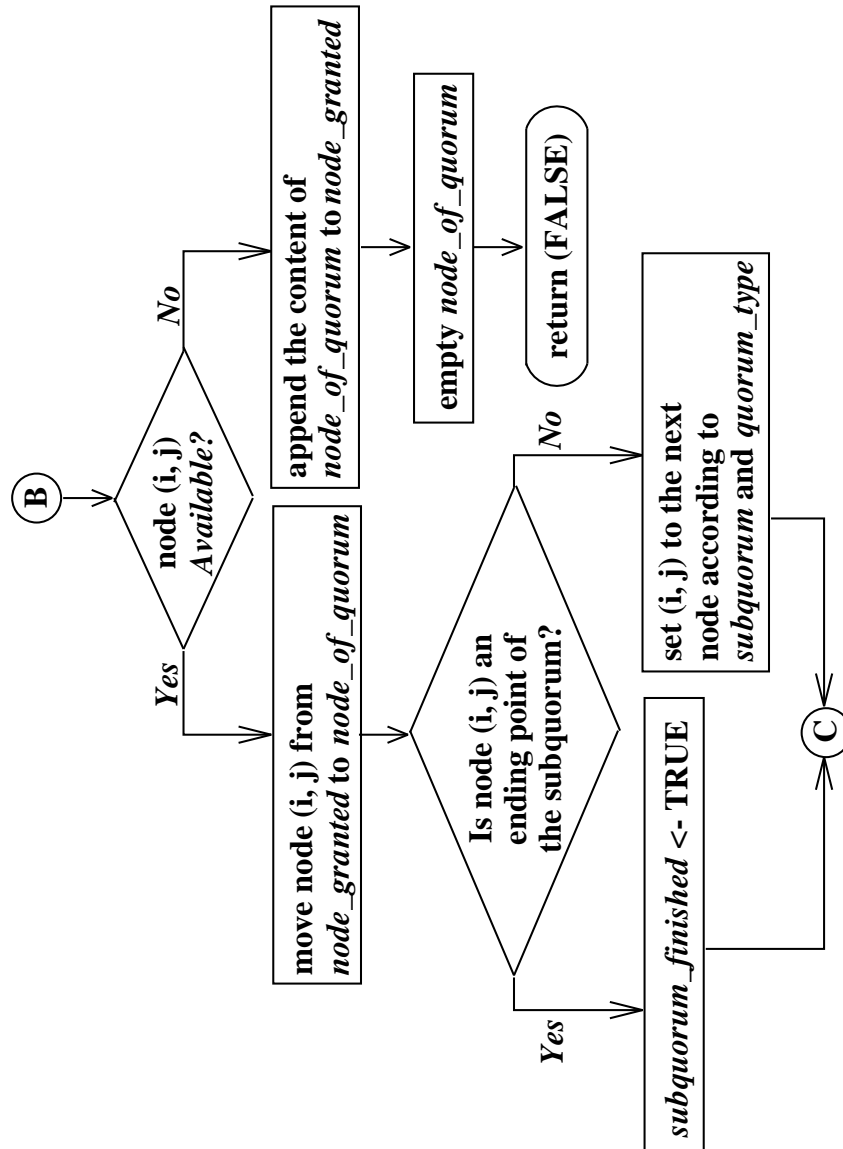


Figure 3.5 The flowchart for procedure *construct_subquorums* in TM (continued)

```

    end;
end;
else (* query_status(i, j, Unknown) == FALSE, maybe Available or Unavailable *)
begin
  if(query_status(i, j, Available) == TRUE) then
  begin
    move node (i, j) from node_granted to node_of_quorum;
    if (query_status(i, j, Ending_Point, subquorum) == TRUE) then
      subquorum_finished = TRUE
    else
      set (i, j) to the next node according to subquorum and quorum_type;
    end
  else (* node (i, j) is Unavailable *)
  begin
    append the content of node_of_quorum to node_granted;
    empty node_of_quorum;
    return (FALSE);
  end;
  end;
  until (subquorum_finished);
end; (* for *)
end; (* construct_subquorums *)

procedure construct_quorum: boolean;
var loop: boolean;
begin
  initialize_status;
  use the requesting node as center;
  while (TRUE) do
  begin
    loop = FALSE;
    repeat
    if (construct_subquorums(1, center) == TRUE) then
    begin
      release all the nodes not included in the quorum;
      return (TRUE);
    end;
    from node_granted, a node is chosen as center which is Type-1_Maybe_Available and
    whose associated type-1 quorum, if available, will include the largest fraction of node_granted;
    if no node of this kind is available then loop = TRUE;
    until (loop);
    loop = FALSE;
    repeat
    from node_granted, a node is chosen as center which is Type-2_Maybe_Available and
    whose associated type-2 quorum, if available, will include the largest fraction of node_granted;
    if no node of this kind is available then loop = TRUE
    else
    begin
    if (construct_subquorums(2, center) == TRUE) then
    begin
      release all the nodes not included in the quorum;
      return (TRUE);
    end;
    end;
    until (loop);
    select a node which is Unknown, and either Type-1_Maybe_Available
    or Type-2_Maybe_Available or both;
    if no such kind of node can be found then
    begin
      release nodes in node_granted;
      return (FALSE);
    end;
    use the selected node as center;

```

end; (* end while *)
end; (* construct quorum *)

3.3 Proof of Correctness

To show that the proposed protocol can ensure mutual exclusion, the following theorem is used.

Theorem 3.1 *Any two quorums of the triangular mesh protocol intersect.*

Proof. We prove by first constructing a quorum x , then further construction of a quorum y will intersect some node in quorum x . There are three cases which should be taken into account about where the center of quorum x is:

- (1) If the center of quorum x is a node inside the triangular mesh, quorum x will divide the triangular mesh into three regions and no nodes of quorum x belong to any region. To prevent from intersecting quorum x , the center of quorum y should reside in any region. Since every region can provide nodes of two sides, only the other two regions can provide nodes of the third side which quorum y needs. The path from the center of quorum y to another region will intersect some node of quorum x since every two regions are separated by quorum x .
- (2) If the center of quorum x resides on any of three sides but is not any of three corners, quorum x will divide the triangle into two regions. Since every region can provide nodes of two sides, only the other region can provide nodes of the third side which quorum y needs. The path from the center of quorum y to another region will intersect some node of quorum x .
- (3) If the center of quorum x is one of three corners, quorum x will contain all nodes of one side of the triangular mesh, quorum y will intersect quorum x at least one node since quorum y must get at least one node from every side. □

3.4 Properties of the Protocol

The following lemmas show how a failed node affects the construction of quorums of other nodes.

Lemma 3.1 *In a k -triangular mesh, given a node x , then those nodes of the type-1 quorum associated with x will include node x in their associated type-2 quorums.*

Proof. Let (x_0, y_0) be the (x, y) -tuple associated with node x . Three sets of nodes:

$$\{(x, x_0 + y_0 - x) \mid 0 \leq x \leq x_0\}, \quad (3.1)$$

$$\{(u - y_0, y_0) \mid x_0 + y_0 \leq u \leq k - 1\}, \quad (3.2)$$

$$\{(x_0, y) \mid 0 \leq y \leq y_0\}, \quad (3.3)$$

constitute subquorum 0, 1, 2 of the type-1 quorum, respectively. And the following sets:

$$\{(x, y_0) \mid 0 \leq x \leq x_0\}, \quad (3.4)$$

$$\{(x_0, u - x_0) \mid x_0 + y_0 \leq u \leq k - 1\}, \quad (3.5)$$

$$\{(x_0 + y_0 - y, y) \mid 0 \leq y \leq y_0\}, \quad (3.6)$$

is subquorum 0, 1, and 2 of the type-2 quorum, respectively. Take a node y from subquorum 0 of the type-1 quorum associated with x , let its (x, y) -tuple be (x_1, y_1) , and from (1), we get $0 \leq x_1 \leq x_0$, and $y_1 = x_0 + y_0 - x_1$. These two relations imply that $y_0 \leq y_1$. From (6), the (x, y) -tuple of any node of subquorum 2 of the type-2 quorum associated with node y is $(x_1 + y_1 - y, y)$, where $0 \leq y \leq y_1$. Since $y_0 \leq y_1$, we can replace y with y_0 in $(x_1 + y_1 - y, y)$ and get $(x_0, y_0) = \text{node } x$; i.e., node x is a node of type-2 quorum associated with y . In a similar way, we can show that the nodes of subquorum 1, 2 of the type-1 quorum associated with x will include x in the subquorum 0, 1, respectively. \square

Lemma 3.2 *In a k -triangular mesh, given a node x , then those nodes of the type-2 quorum associated with x will include node x in their associated type-1 quorums.*

Proof. This can be proved in the similar way as in Lemma 3.1. \square

Theorem 3.2 *Given a k -triangular mesh, the quorum size of the proposed protocol is k , which is proportional to \sqrt{N} , where N is the number of nodes in the system.*

Proof. Assume that the (x, y) -tuple associated with the center of a given quorum is (x_0, y_0) , subquorum 0 is a sequence of nodes (v_0, \dots, v_m) , where v_0 is the center and v_m is the ending node. Take any two adjacent nodes $v_{i+1} = (x_{i+1}, y_{i+1})$, and $v_i = (x_i, y_i)$, the condition $(x_{i+1} - x_i = -1)$ is true for all $i = 0, \dots, m - 1$. When we trace from the center to the ending node, we find that the value of x in the (x, y) -tuple is changed from x_0 to zero; therefore, there are $(x_0 + 1)$ nodes in subquorum 0. Apply the same approach and notation to any two adjacent nodes v_{i+1} and v_i in subquorum 1, the condition $((x_{i+1} + y_{i+1}) - (x_i + y_i) = 1)$ holds. Since the value of $(x + y)$ is changed from $(x_0 + y_0)$ to $(k - 1)$, there are $(k - x_0 - y_0)$ nodes in subquorum 1. Similarly, for any two adjacent nodes v_{i+1} and v_i in subquorum 2, the condition $(y_{i+1} - y_i = -1)$ holds. Since the value of y is changed from y_0 to 0, there are $(y_0 + 1)$ nodes in subquorum 2. Since the center is counted three times, the number of nodes in the quorum is $(x_0 + 1) + (k - x_0 - y_0) + (y_0 + 1) - 2 = k$. Given $N = \frac{k(k+1)}{2}$ nodes in a k -triangular mesh, we get $k < \sqrt{2N} < k + 1$, i.e., $\sqrt{2N} - 1 < k < \sqrt{2N}$. Apparently, k is of $O(\sqrt{N})$. \square

The following theorem shows that the triangular mesh protocol is not vote assignable.

Theorem 3.3 *The triangular mesh protocol does not have an equivalent vote assignment for $k \geq 3$.*

Proof. There are two cases which we have to take into consideration: $k = 3$ and $k \geq 4$.

- (1) We label each node in a 3-triangular mesh as depicted in Figure 3.6. Assume that there exists an equivalent vote assignment which assigns a positive vote v_i to node i . The set of nodes $\{0, 1, 3\}$, $\{0, 2, 4\}$, $\{1, 2, 5\}$ and $\{3, 4, 5\}$ are valid quorums. Therefore, four inequalities are obtained:

$$v_0 + v_1 + v_3 > v_2 + v_4 + v_5, \quad (3.7)$$

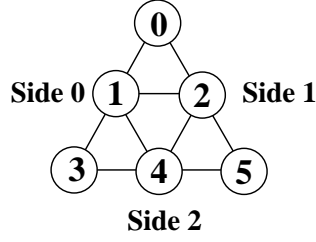


Figure 3.6 A 3-triangular mesh.

$$v_0 + v_2 + v_4 > v_1 + v_3 + v_5, \quad (3.8)$$

$$v_1 + v_2 + v_5 > v_0 + v_3 + v_4, \quad (3.9)$$

$$v_3 + v_4 + v_5 > v_0 + v_1 + v_2. \quad (3.10)$$

From (3.7) and (3.8), we get

$$v_0 > v_5. \quad (3.11)$$

But from (3.9) and (3.10),

$$v_5 > v_0. \quad (3.12)$$

This is a contradiction. Hence, the triangular mesh protocol does not have an equivalent vote assignment when $k = 3$.

- (2) For a k -triangular mesh where $k \geq 4$, there will be nodes which do not reside at any side of the triangular mesh. We partition the nodes of the triangular mesh into seven disjoint sets: $C_0, C_1, C_2, S_0, S_1, S_2$ and I . The set C_i , where $i = 0, 1, 2$, contains only the node not residing at side i . The set S_i , where $i = 0, 1, 2$, contains those nodes residing at side i of the triangular mesh but not at any corner. The set I contains the nodes not residing at any side. Figure 3.7 shows the partition of the nodes in a 7-triangular mesh into the sets described above.

Assume that there exists an equivalent vote assignment which assigns a positive vote to each node. The notation V_X stands for the sum of the votes of every node in set X . Since all nodes of a side constitute a quorum, we can get three inequalities immediately:

$$V_{C_1} + V_{C_2} + V_{S_0} > V_{C_0} + V_{S_1} + V_{S_2} + V_I, \quad (3.13)$$

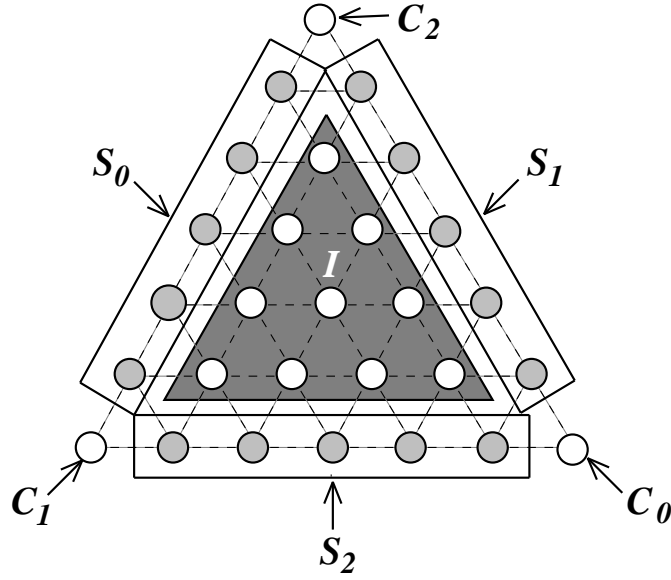


Figure 3.7 The partition of nodes in a 7-triangular mesh

$$V_{C_0} + V_{C_2} + V_{S_1} > V_{C_1} + V_{S_0} + V_{S_2} + V_I, \quad (3.14)$$

$$V_{C_0} + V_{C_1} + V_{S_2} > V_{C_2} + V_{S_0} + V_{S_1} + V_I. \quad (3.15)$$

Now, we add any two inequalities and remove identical items from the resulting inequality, three new inequalities are obtained:

$$V_{C_2} > V_{S_2} + V_I, \quad (3.16)$$

$$V_{C_0} > V_{S_0} + V_I, \quad (3.17)$$

$$V_{C_1} > V_{S_1} + V_I. \quad (3.18)$$

Now, add (3.16), (3.17) and (3.18) together, we get

$$V_{C_0} + V_{C_1} + V_{C_2} > V_{S_0} + V_{S_1} + V_{S_2} + 3 \times V_I. \quad (3.19)$$

It is obvious that the set of three nodes residing at corners does not constitute a valid quorum, we get

$$V_{C_0} + V_{C_1} + V_{C_2} < V_{S_0} + V_{S_1} + V_{S_2} + V_I. \quad (3.20)$$

From (3.19) and (3.20), we get

$$V_I < 0. \quad (3.21)$$

This violates the fact that we assign each node a positive vote. Therefore, the triangular mesh protocol does not have an equivalent vote assignment. \square

CHAPTER IV

The Triple Triangular Mesh Protocol

In this Chapter, we present the triple triangular mesh (TTM) protocol. We first give the definition and examples of TTM quorums. Next, we describe the algorithm for constructing a quorum of the triple triangular mesh protocol. Then, we present a proof of correctness of the proposed protocol. Finally, several properties of the protocol are described.

4.1 Definition

The *k*-triangular mesh used in TTM is the same as that used in TM. Given a node x residing inside the triangular mesh, we can draw three lines along those edges of the triangular mesh, such that each of them is parallel with one side of the triangular mesh and passes through the given node. Based on these three sides and three lines, three smaller triangles are formed in the triangular mesh. Figure 4.1 shows such an example, where node 7 is the given center. We define a small triangle as subtriangle i such that one of its sides is a subset of side i of the triangular mesh, and define subtriangle i 's two sides counterclockwisely, which are not subsets of any side of the triangular mesh, as side (a) and side (b), respectively. Figure 4.2 shows sides of three subtriangles from Figure 4.1.

Definition 4.1 *Each quorum consists of a center and subquorum i , for $i = 0, 1, 2$. Subquorum i contains all the nodes on either side (a) or side (b) of subtriangle i , and we refer to the former as subquorum i -(a) and the latter as subquorum i -(b), respectively.*

Also, we will call a quorum "associated with" node x if it uses node x as its center, and a subquorum "associated with" node x if it is a subquorum of the quorum associated with node x .

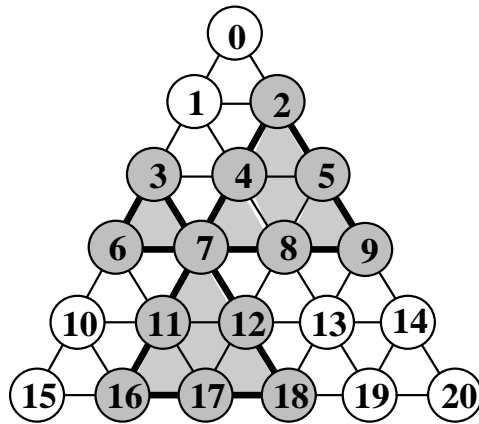


Figure 4.1 Subtriangles in a 6-triangular mesh in TTM

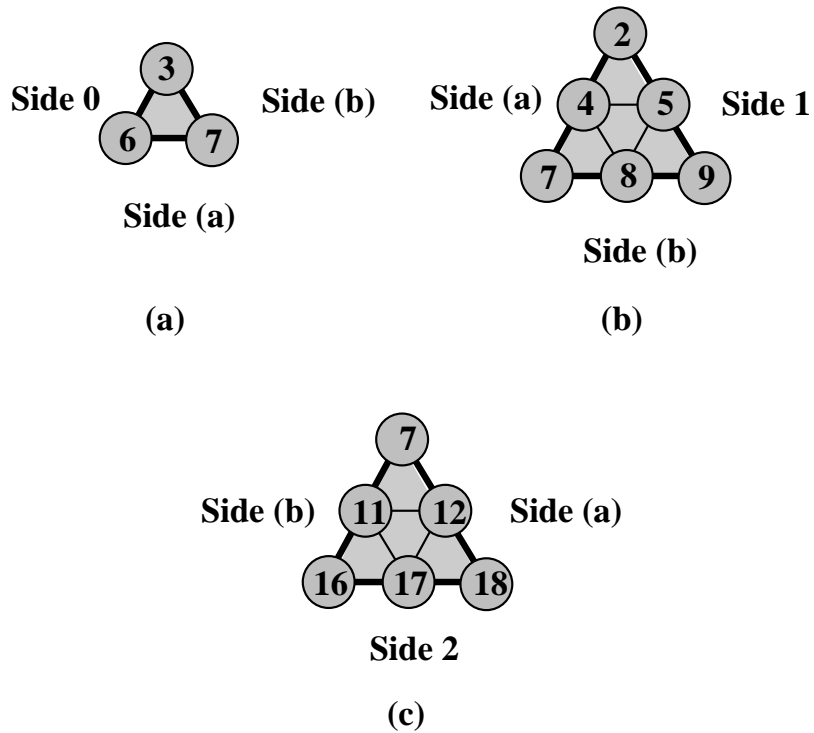


Figure 4.2 Sides of subtriangles in TTM: (a) subtriangle 0; (b) subtriangle 1; (c) subtriangle 2.

Example 4.1 For a node residing inside a triangular mesh, there are eight quorums associated with it. Figure 4.3 shows eight quorums associated with node a residing inside a 6-triangular mesh.

Example 4.2 For a node residing at one side of a triangular mesh but not at any corners, there are four quorums associated with it. Figure 4.4 shows four quorums associated with node a residing at one side of the triangular mesh. (Note that the number of subtriangles generated depends on the location of the center of the quorum.)

We can describe the quorum defined above in another way. A quorum consists of a center and subquorum i , for $i = 0, 1, 2$. Subquorum i is a sequence of nodes, (v_0, \dots, v_m) , where v_0 is the center of the quorum, and v_m is the ending node of the subquorum. Any two adjacent nodes v_j, v_{j+1} , $0 \leq j < m$ in the subquorum must satisfy the condition according to its type:

- (1) $x_{j+1} - x_j = -1$ and $y_{j+1} = y_j$, for subquorum 0-(a);
- (2) $x_{j+1} - x_j = -1$ and $y_{j+1} - y_j = 1$, for subquorum 0-(b);
- (3) $x_{j+1} = x_j$ and $y_{j+1} - y_j = 1$, for subquorum 1-(a);
- (4) $x_{j+1} - x_j = 1$ and $y_{j+1} = y_j$, for subquorum 1-(b);
- (5) $x_{j+1} - x_j = 1$ and $y_{j+1} - y_j = -1$, for subquorum 2-(a);
- (6) $x_{j+1} = x_j$ and $y_{j+1} - y_j = -1$, for subquorum 2-(b).

4.2 The Algorithm for Constructing a Quorum

In our algorithm to construct a quorum, each node is associated with two kinds of information: *node status* and *subquorum status*. *Node status* indicates whether a node is *Available*, *Unavailable* or *Unknown*. Since a quorum consists of three subquorums, we associate with each node three *subquorum status* elements which show that whether it is possible to construct subquorum i associated with a given node, for $i = 0, 1, 2$. The legal values of *subquorum status* of subquorum i are 0, 1, 2 or 3 which indicate that

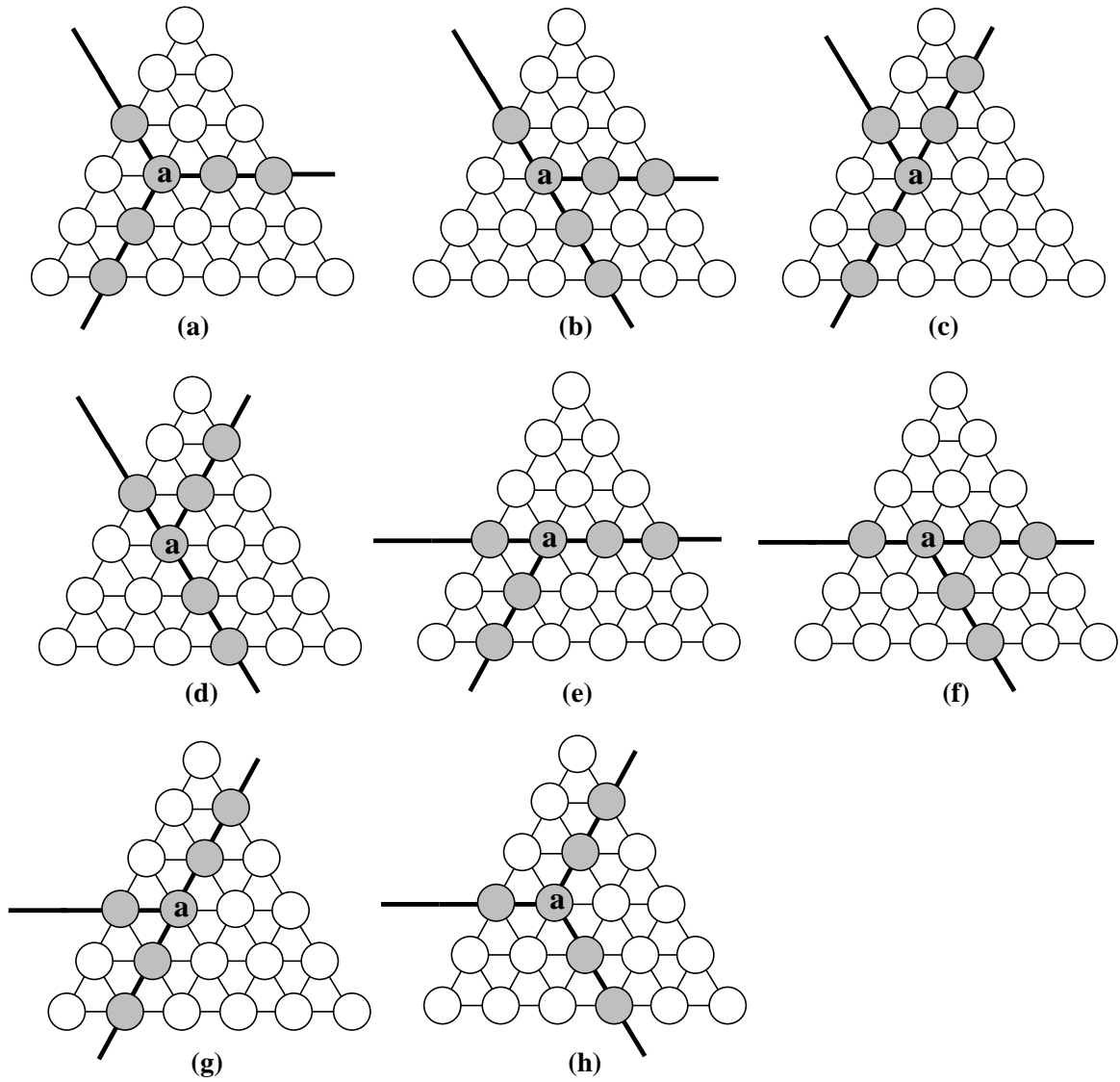


Figure 4.3 Examples of quorums associated with a node a in a 6-triangular mesh where a resides inside the triangle in TTM: (a)-(h).

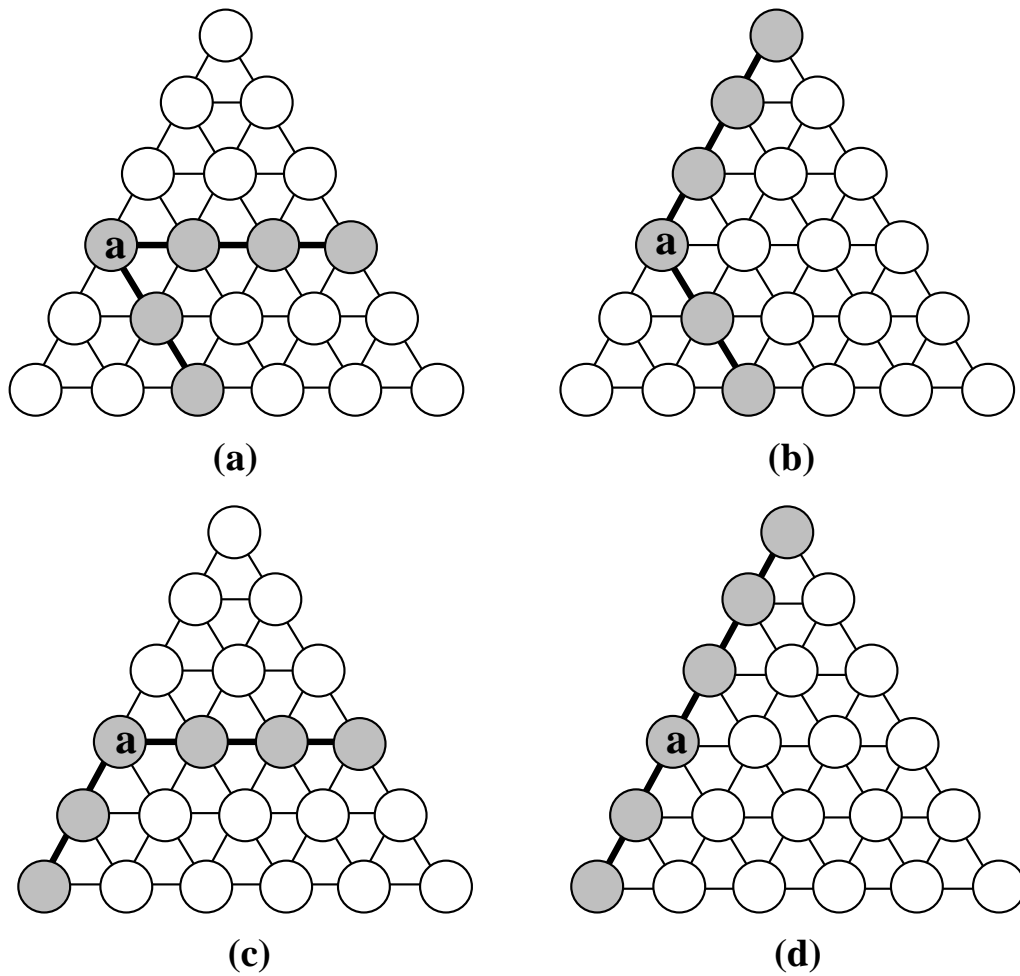


Figure 4.4 Examples of quorums associated with a node a in a 6-triangular mesh where a resides at one side but is not at a corner in TTM: (a)-(d).

subquorum i -(a) or i -(b) may be available, subquorum i -(a) is unavailable, subquorum i -(b) is unavailable, and neither subquorum i -(a) nor (b) is available, respectively.

Initially, we set node status to be *Unknown*, and subquorum status to be zero for all nodes. The set *node_of_subquorum* is used to record the nodes which grant the request and are included in the subquorum being constructed. When the construction of a subquorum succeeds, the nodes in *node_of_subquorum* are moved to the set *node_of_quorum*; otherwise, they are moved to the set *node_granted*.

Each quorum consists of three subquorums and each subquorum can be constructed in a similar way which will be described later. A quorum associated with a given node x is available only if all of its subquorums i , for $i = 0, 1, 2$, are available. If one of them is unavailable, no quorum associated with node x is available. In this case, we select a node y as the new center from those granted nodes such that none of node y 's subquorum status is of value 3 and node y will include the largest part of those granted nodes in its associated quorum. Then, we release those granted nodes which will not be included in node y 's associated quorums and try to construct a quorum associated with node y . If no node like y can be found, we randomly choose a untried node to be the new center such that none of its subquorum status is of value 3 and repeat the same steps to construct a quorum.

We now describe how to construct a subquorum associated with node x . Since there are two alternatives for a subquorum except the case where the subquorum consists of only one node, we first try to construct one instance of the subquorum by issuing requests to nodes in the direction from the center to the ending node. If the construction succeeds, the subquorum is available. Otherwise, the construction fails due to some unavailable node v , we mark node v unavailable, find those nodes whose associated subquorums will include node v and change their subquorum status accordingly, then try to construct another instance of the subquorum. If it fails again, no instance of this subquorum is available; therefore, no quorum associated with node x is available, we apply the steps described previously to choose a new center and then repeat the construction steps. Finally, if there

is no node which can be used as the new center and no quorum is available yet, the construction fails. Figure 4.5 and 4.6 show the flowcharts for procedure *construct_quorum* and *construct_subquorum*, respectively.

Algorithm for the Triple Triangular Mesh Protocol

```

procedure initialize_status;
begin
  for each node (x, y) in the system do
    begin
      set_status(x, y, Unknown);
      subquorum_status[x, y, 0]=subquorum_status[x, y, 1]=subquorum_status[x, y, 2]=0;
    end;
    empty the lists: node_of_subquorum, node_of_quorum, node_granted;
  end; (* initialize_status *)

procedure construct_subquorums(center): boolean;
var subquorum: integer; (* indicate which subquorum is being constructed *)
    subquorum_finished: boolean;
    flag: integer; (* 0: subquorum i-(a); 1: subquorum i-(b) *)
begin
  for subquorum  $\in$  { 0, 1, 2 } do
    begin
      set (i, j) to center;
      subquorum_finished = FALSE;
      flag = 0;
      repeat
        if (query_status(i, j, Unknown) == TRUE) then
          begin
            if (request(i, j) == GRANTED) then
              begin
                set_status(i, j, Available);
                add node (i, j) into node_of_subquorum;
                if (query_status(i, j, Ending_Point, subquorum) == TRUE) then
                  begin
                    subquorum_finished = TRUE;
                    move nodes in node_of_subquorum into node_granted;
                  end
                else set (i, j) to the next node according to subquorum and flag;
              end
            else (* request is denied by (i, j) *)
              begin
                set_status(i, j, Unavailable);
                find all nodes such that some of their subquorums will include (i, j) and
                mark these subquorums unavailable by setting proper values to subquorum_status's;
                move nodes in node_of_subquorum to node_granted;
                if (flag == 1) then return (FALSE);
                flag = 1;
                set (i, j) to center;
              end;
            end;
          end
        end
      end
    end
  end

```

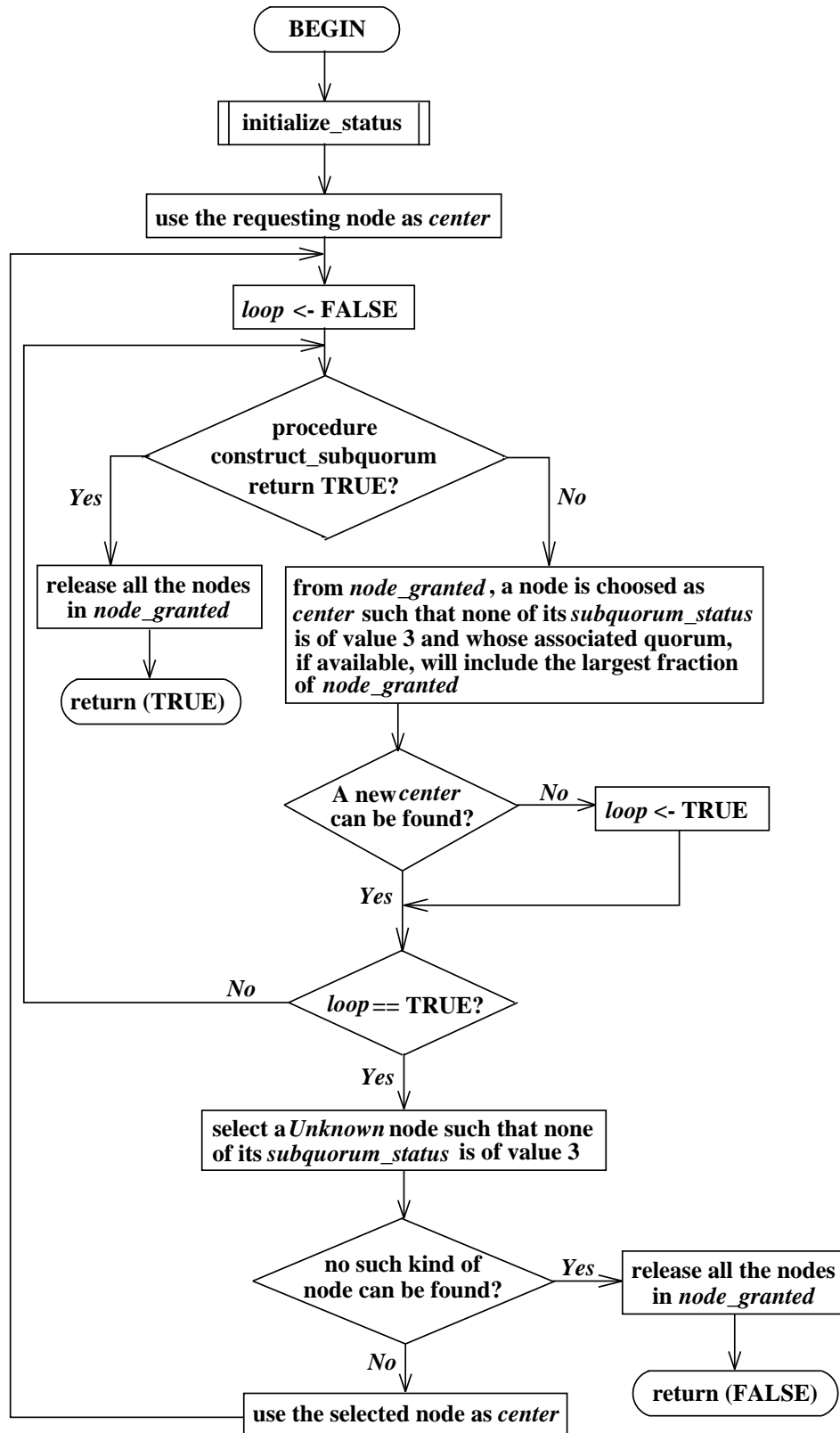


Figure 4.5 The flowchart for procedure *construct_quorum* in TTM

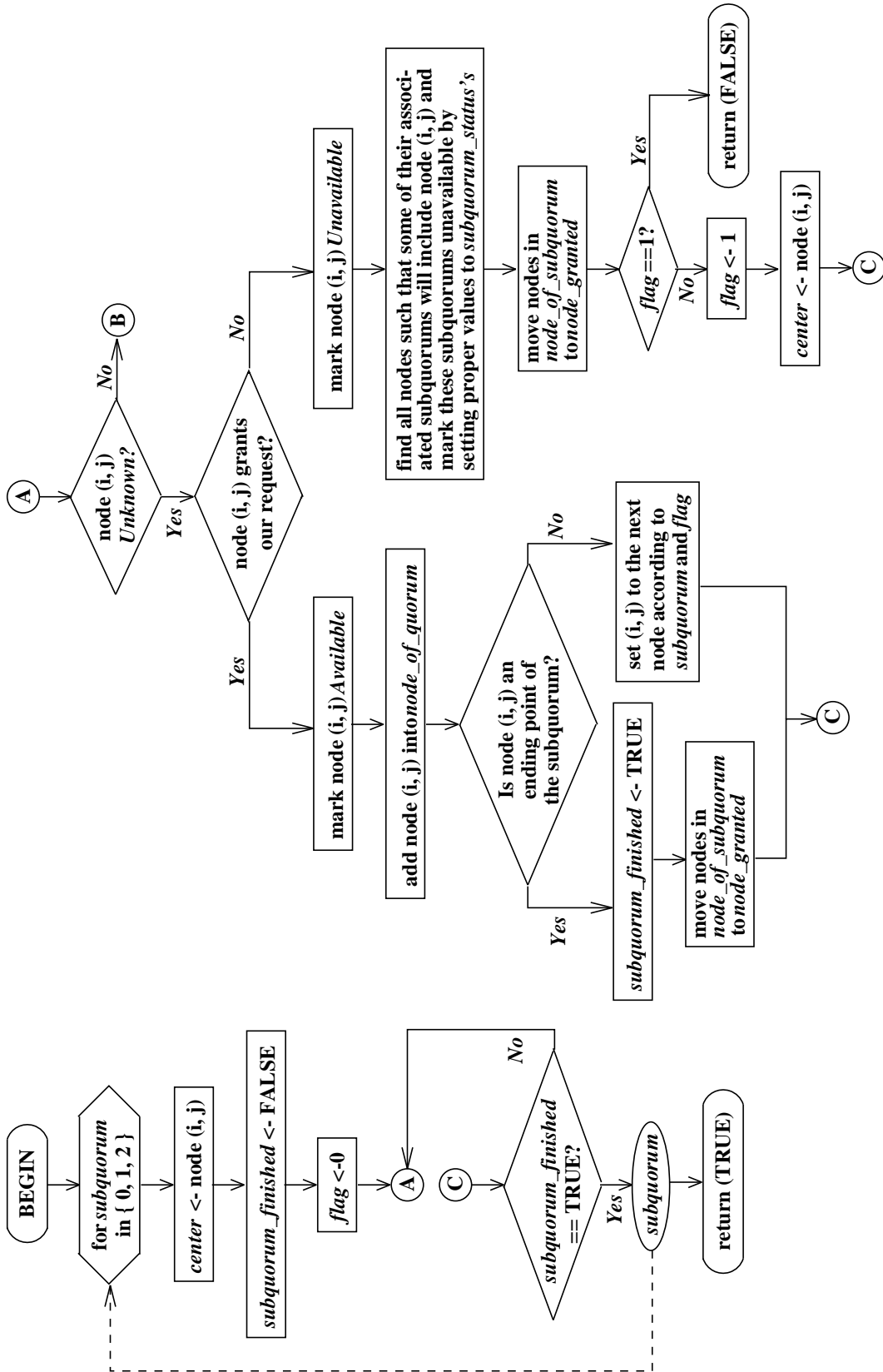


Figure 4.6 The flowchart for procedure *construct_subquorums* in TTM

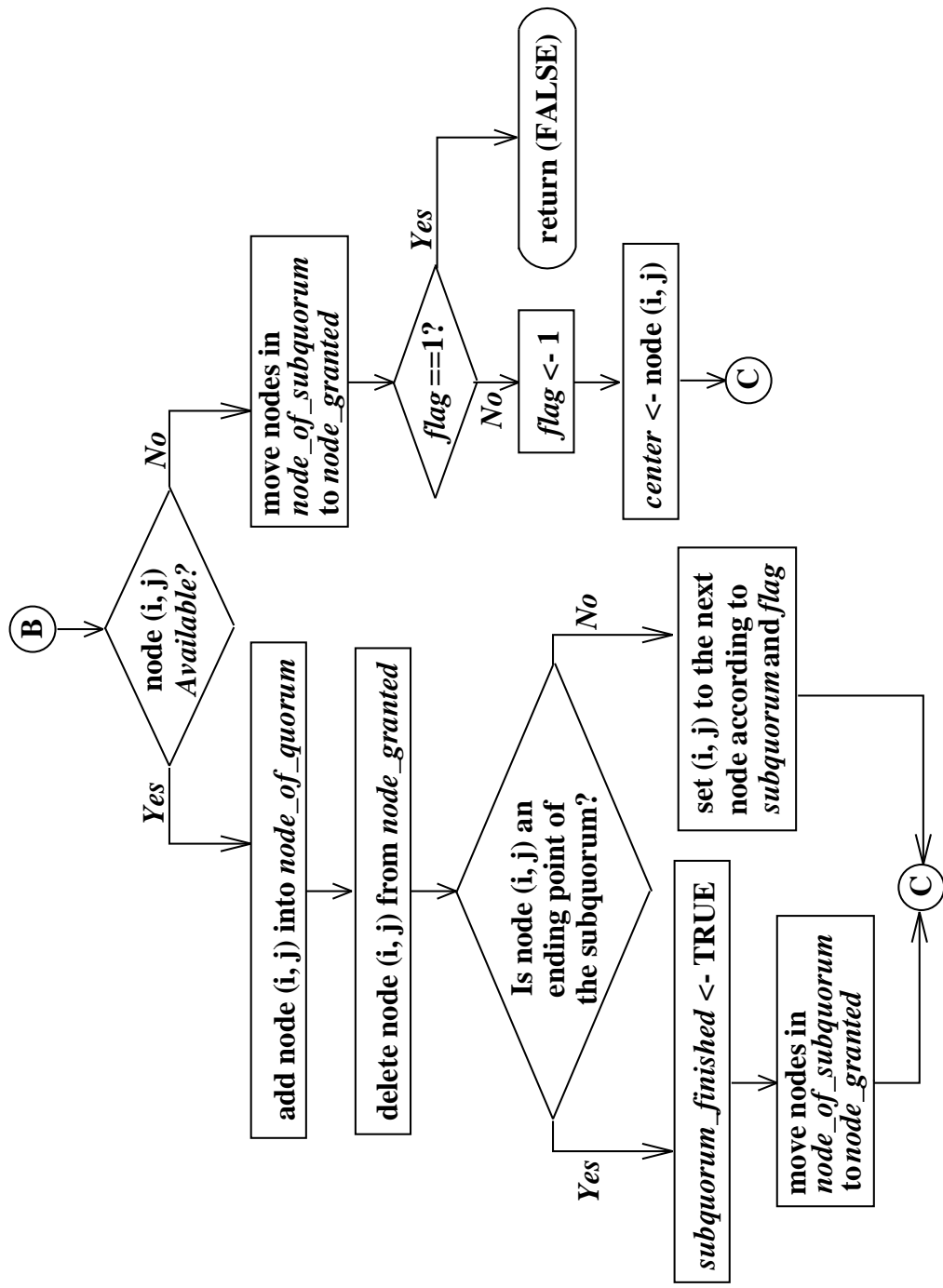


Figure 4.6 The flowchart for procedure *construct_quorum* in TTM (continued)

```

else (* query_status(i, j, Unknown) == FALSE, maybe Available or Unavailable *)
begin
  if (query_status(i, j, Available) == TRUE) then
  begin
    add node (i, j) to node_of_subquorum;
    delete node (i, j) from node_granted;
    if (query_status(i, j, Ending_Point, subquorum) == TRUE) then
    begin
      subquorum_finished = TRUE;
      move nodes in node_of_subquorum to node_of_quorum;
    end
    else set (i, j) to the next node according to subquorum and flag;
  end
  else (* node (i, j) is Unavailable *)
  begin
    move nodes in node_of_subquorum to node_granted;
    if (flag == 1) then return (FALSE);
    flag = 1;
    set (i, j) to center;
  end;
end;
until (subquorum_finished);
end; (* for *)
return (TRUE);
end; (* construct_subquorums *)

```

```

procedure construct_quorum: boolean;
var loop: boolean;
begin
  initialize_status;
  use the requesting node as center;
  while (TRUE) do
  begin
    loop = FALSE;
    repeat
    if (construct_subquorums(center) == TRUE) then
    begin
      release all the nodes in node_granted;
      return (TRUE);
    end;
    from node_granted, a node is choosed as center such that none of its
    subquorum_status's is of value 3 and whose associated quorum, if available,
    will include the largest fraction of node_granted;
    if no node of this kind is available then loop = TRUE;
  until (loop);
  select a node which is Unknown, and none of its subquorum_status's is of value 3;
  if no such kind of node can be found then
  begin
    release nodes in node_granted;
    return (FALSE);
  end;
  use the selected node as center;

```

```
end; (* end while *)
end; (* construct quorum *)
```

4.3 Proof of Correctness

The following theorem is used to show that the proposed protocol can ensure mutual exclusion.

Theorem 4.1 *Any two quorums of the triple triangular mesh protocol intersect.*

Proof. We prove by first constructing a quorum x , then further construction of a quorum y will intersect some node in quorum x . There are three cases which should be taken into account according to where the center of quorum x is:

- (1) If the center of quorum x is a node inside the triangular mesh, quorum x will divide the triangular mesh into three regions and no nodes of quorum x belong to any region. Figure 4.3 is a demonstration. To prevent from intersecting quorum x , the center of quorum y should reside in any region. Since every region can provide nodes of two sides, only the other two regions can provide nodes of the third side which quorum y needs. The path from the center of quorum y to another region will intersect some node of quorum x since every two regions are separated by quorum x .
- (2) If the center of quorum x resides on any of three sides but is not any of three corners, there are three cases should be taken into account: none, one or two of x 's subquorums resides at some side of the triangular mesh. When none of x 's subquorum resides at some side (ex., Figure 4.4-(a)), the triangular mesh is divided into three parts by quorum x . In a similar way as in case (1), we can show that no other quorum can be constructed without intersecting quorum x . If only one of x 's subquorum resides at some side, the triangle will be divided into two regions (ex., Figures 4.4-(b) and (c)). Since every region can provide nodes of two sides, only the other region can provide nodes of the third side which quorum y needs. The path from the center

of quorum y to another region will intersect some node of quorum x . If two of x 's subquorums reside at some side of the triangle, quorum x is one side of the triangle (ex., Figure 4.4-(d)), and quorum y will intersect quorum x at least one node since quorum y must get at least one node from every side.

- (3) If the center of quorum x is one of three corners, quorum x will contain all nodes of one side of the triangular mesh, quorum y will intersect quorum x at least one node since quorum y must get at least one node from every side. \square

4.4 Properties of the Protocol

The following lemma shows how a failed node affects the construction of quorums of other nodes.

Lemma 4.1 *In a k -triangular mesh, given a node x , then those nodes of the subquorum 0-(a), 0-(b), 1-(a), 1-(b), 2-(a), and 2-(b) associated with x will include node x in their associated subquorum 1-(b), 2-(a), 2-(b), 0-(a), 0-(b), and 1-(a), respectively.*

Proof. Here, we show the case that nodes in subquorum 0-(b) associated with node x will include node x in their associated subquorum 2-(a), and the other cases can be proved in a similar way. Let (x_0, y_0) be the (x, y) -tuple associated with node x . The set of nodes:

$$\{(x, y) \mid 0 \leq x \leq x_0, y = x_0 + y_0 - x, x, y \in N \cup \{0\}\},$$

constitutes subquorum 0-(b) associated with node x . For any node $y = (x_1, y_1)$ in the subquorum 0-(b), x_1, y_1 must satisfy:

$$0 \leq x_1 \leq x_0, \text{ and} \tag{4.1}$$

$$y_1 = x_0 + y_0 - x_1. \tag{4.2}$$

Also, the subquorum 2-(a) associated with node y consists of the nodes of the (x, y) -tuples, where $x, y \in N \cup \{0\}$, and must satisfy:

$$x_1 \leq x \leq x_1 + y_1, \text{ and} \tag{4.3}$$

$$y = x_1 + y_1 - x. \quad (4.4)$$

From (4.1), and (4.3), we know that there is a node with x-coordinate x_0 which is included in subquorum 2-(a) of node y . From (4.4) and (4.3), we know that the y-coordinate of that node is y_0 . Therefore, the case that node x is included in the subquorum 2-(a) of node y is proved. \square

Theorem 4.2 *Given a k -triangular mesh, the quorum size of the proposed protocol is k , which is proportional to \sqrt{N} , where N is the number of nodes in the system.*

Proof. This can be proved in the same way as in the TM protocol. \square

Theorem 4.3 *The TTM protocol dominates the TM protocol when $k \geq 3$.*

Proof. We denote the set of all quorums defined by the TM and TTM protocols in a k -triangular mesh as C_{TM} and C_{TTM} , respectively. We first prove that C_{TM} is a coterie by showing that C_{TM} satisfies the three constraints in Definition 2.1:

- (1) It is apparent that C_{TM} is not empty.
- (2) Any $G, H \in C_{TM}$, $G \cap H \neq \phi$, since any two quorums defined by the TM protocol intersect. This is proved in Theorem 3.1.
- (3) The size of every quorum of the TM protocol in a k -triangular mesh is k , i.e., any $G, H \in C_{TM}$, $|G| = |H|$, this implies that $G \not\subseteq H$ and $H \not\subseteq G$.

The fact that C_{TTM} is a coterie can be proved in the similar way. From the definition of quorums, we observe that all the quorums in the TM protocol are also quorums in the TTM protocol, i.e., $C_{TM} \subseteq C_{TTM}$. Since $C_{TTM} - C_{TM} \neq \emptyset$, we know that $C_{TM} \subset C_{TTM}$. From Definition 2.2, we know that the TTM protocol dominates the TM protocol. \square

Theorem 4.4 *The triple triangular mesh protocol does not have an equivalent vote assignment for $k \geq 3$.*

Proof. From the definition of quorums, we knew that all the quorums in the TM protocol are also quorums in the TTM protocol. Since the TM protocol does not have an equivalent vote assignment as shown in Theorem 3.3, there is also no equivalent vote assignment for the TTM protocol. □

CHAPTER V

The Dynamic Triangular Mesh Protocol

In this Chapter, we present the dynamic triangular mesh (DTM) protocol. We first give the definitions and examples of DTM quorums. Next, we describe the algorithm for constructing a quorum of the dynamic triangular mesh protocol. Then, we present a proof of correctness of the proposed protocol. Finally, several properties of the protocol are described.

5.1 Definition

The k-triangular mesh used in DTM is the same as that used in TM. The subtriangles used in DTM are the same as those used in TTM.

Definition 5.1 *The quorum consists of a center and subquorum i , for $i = 0, 1, 2$. Subquorum i is a sequence of nodes, (v_0, \dots, v_m) which forms a path in subtriangle i , where v_0 is the center of the quorum, and v_m is one of the nodes located in side i . For any two adjacent nodes v_j, v_{j+1} , for $j = 0, \dots, m - 1$, they must satisfy one of the following conditions according to i :*

(1) *For $i = 0$, either*

(a) $x_{i+1} - x_i = -1$ and $y_{i+1} = y_i$ or

(b) $x_{i+1} - x_i = -1$ and $y_{i+1} - y_i = 1$.

(2) *For $i = 1$, either*

(a) $x_{i+1} = x_i$ and $y_{i+1} - y_i = 1$ or

(b) $x_{i+1} - x_i = 1$ and $y_{i+1} = y_i$.

(3) *For $i = 2$, either*

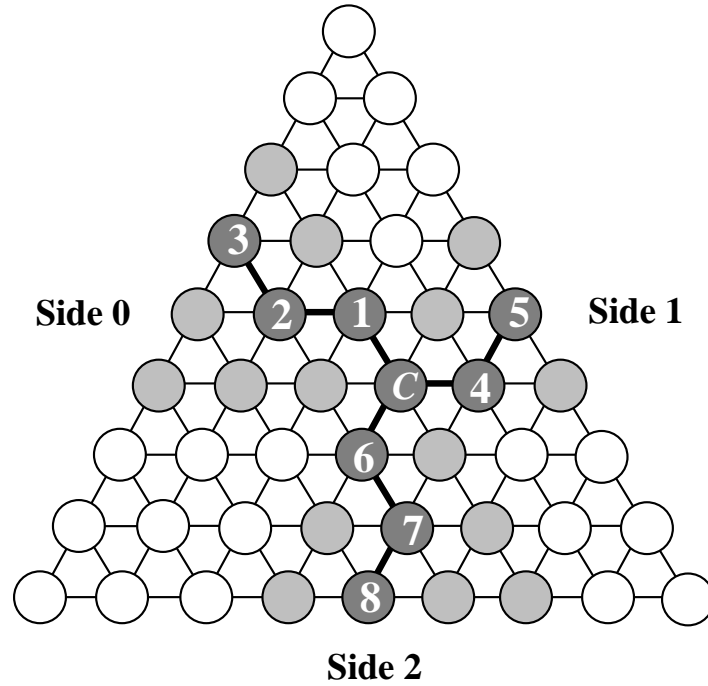


Figure 5.1 An example of a quorum in a 9-triangular mesh in DTM

- (a) $x_{i+1} - x_i = 1$ and $y_{i+1} - y_i = -1$ or
- (b) $x_{i+1} = x_i$ and $y_{i+1} - y_i = -1$.

Example 5.1 Figure 5.1 shows an example of quorum in a 9-triangular mesh. A node C is used as the center of a quorum. Only shaded nodes are possible to be included in a subquorum and nodes linked by bold line (i.e., nodes 1, 2, 3, 4, 5, 6, 7, 8 and C) constitute examples of subquorums.

Also, we will call a quorum "*associated with*" node x if it uses node x as its center, and a subquorum "*associated with*" node x if it is a subquorum of some quorum associated with node x . Moreover, from the definition of quorums, we observe that subquorum i associated with node x will not include those nodes residing outside subtriangle i .

5.2 The Algorithm for Constructing Quorums

In our algorithm to construct a quorum, each node is associated with two kinds of information: *node status* and *subquorum status*. *Node status* indicates whether a node is

Available, Unavailable or Unknown. Since a quorum consists of three subquorums, we associate with each node three *subquorum-status* elements which indicate that whether it is possible to construct subquorum i associated with a given node, for $i = 0, 1, 2$. The legal values of *subquorum status* of subquorum i are 0, 1 which indicate that subquorum i may be available or is unavailable, respectively.

Initially, we set node status to be *Unknown*, and subquorum status to be zero for all nodes. The set *node_of_quorum* is used to record the nodes which grant the request and are included in the quorum being constructed. The set *node_granted* is used to record the nodes granting the request but not belonging to *node_of_quorum*. Each quorum consists of three subquorums and each subquorum can be constructed in a similar way which will be described later. A quorum associated with a given node x is available only if all of its subquorums i , for $i = 0, 1, 2$, are available. If one of them is unavailable, no quorum associated with node x is available. In this case, we select a node y as the new center from those granted nodes such that none of node y 's subquorum status is of value 1 and node y will include the largest part of those granted nodes in its associated quorum. If no node like y can be found, we randomly choose a untried node to be the new center such that none of its subquorum status is of value 1 and repeat the same steps to construct a quorum.

We now describe how to construct a subquorum associated with node x . We construct a subquorum in a recursive manner. When we try to construct an instance of node x 's associated subquorum i , node x must be available. If node x resides at side i , the construction succeeds since node x is the ending point. Otherwise, there are two nodes y, z adjacent to node x such that the edges $\{x, y\}$ and $\{x, z\}$ are parallel with either side $((i + 1) \bmod 3)$ or $((i + 2) \bmod 3)$ and we have to construct any instance of subquorum i associated with y or z . Node x and any instance of the subquorum i associated with either node y or z constitute an instance of subquorum i associated with node x . If both constructions of subquorum i associated with node y and z fail, no instance of subquorum i associated with node x is available, and we set the subquorum i 's status to 1 and return back to the node issuing requests for node x . If we go back to the center, the construction of the subquorum fails; therefore, no quorum associated with that center is available. We apply the steps described

previously to choose a new center and then repeat the construction steps. Finally, if there is no node which can be used as the new center and no quorum is available yet, the construction fails. Figure 5.2 and 5.3 show the flowcharts for procedure *construct_quorum* and *construct_subquorums*, respectively.

Algorithm for the Dynamic Triangular Mesh Protocol

```

procedure initialize_status;
begin
  for each node (x, y) in the system do
    begin
      set_status(x, y, Unknown);
      subquorum_status[x, y, 0]=subquorum_status[x, y, 1]=subquorum_status[x, y, 2]=0;
    end;
  empty the lists: node_of_quorum, node_granted;
end; (* initialize_status *)

```

```

procedure construct_subquorums(center, subquorum:integer): boolean;
(* subquorum is used to indicate which subquorum is being constructed *)
begin
  set (i, j) to center;
  if (query_status(i, j, Unavailable)==TRUE) then
    return (FALSE);
  (* node status is either Unknown or Available *)
  if (query_status(i, j, Unknown)==TRUE) then
    begin
      if (request(i, j) == GRANTED) then
        set_status(i, j, Available);
      else
        begin
          set_status(i, j, Unavailable);
          subquorum_status[i, j, 0]=subquorum_status[i, j, 1]=subquorum_status[i, j, 2]=1;
          return (FALSE);
        end;
      end;
    (* node (i, j) is available *)
    if (subquorum_status[i, j, subquorum]==0) then
      begin
        add node (i, j) into node_of_quorum;
        delete node (i, j) from node_granted;
        if (query_status(i, j, Ending_Point, subquorum)==TRUE) then
          return (TRUE);
        else
          begin
            (* Since node (i, j) is not the ending point, therefore, there are two nodes
            adjacent to node (i, j), say next_node_1 and next_node_2, such that
            all subquorum passing through node (i, j) will include either of them. *)
            if (construct_subquorum(next_node_1, subquorum)==TRUE OR
              construct_subquorum(next_node_2, subquorum)==TRUE) then
              return (TRUE);
            else
              begin
                subquorum_status[i, j, subquorum]=1;
                move (i, j) from node_of_quorum to node_granted;

```

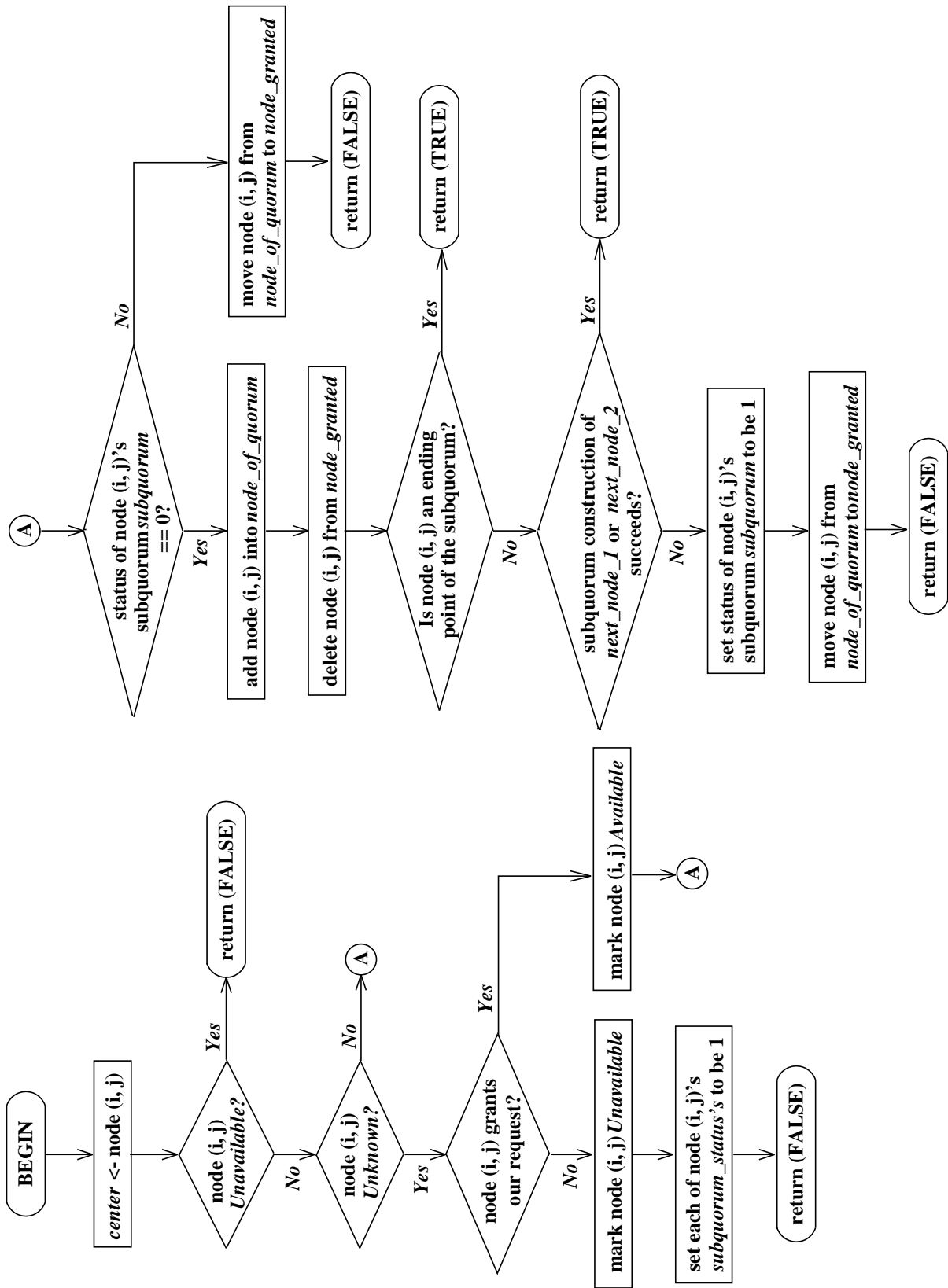



Figure 5.3 The flowchart for procedure *construct_subquorums* in DTM

```

        return (FALSE);
    end;
end
else
begin
    move (i, j) from node_of_quorum to node_granted;
    return (FALSE);
end;
end; (* construct_subquorum *)

procedure construct_quorum: boolean;
var loop: boolean;
begin
    initialize_status;
    use the requesting node as center;
    while (TRUE) do
    begin
        loop=FALSE;
        repeat
            if (construct_subquorums(center, 0) == TRUE AND
                construct_subquorums(center, 1) == TRUE AND
                construct_subquorums(center, 2) == TRUE ) then
                begin
                    release all the nodes in node_granted;
                    return (TRUE);
                end;
            end;
            from node_granted, a node is choosed as center such that none of its
            subquorum_status is of value 1 and whose associated quorum,
            if available, will include the largest fraction of node_granted;
            if no such kind of node can be found then loop=TRUE;
        until (loop);
        select a node which is Unknown, and none of its subquorum_status is of value 1;
        if no such kind of node can be found then return (FALSE);
        use the selected node as center;
    end; (* end while *)
    end; (* end construct quorum *)

```

5.3 Proof of Correctness

The following theorem is used to show that the proposed protocol can ensure mutual exclusion.

Theorem 5.1 *Any two quorums of the dynamic triangular mesh protocol intersect.*

Proof. We prove by first constructing a quorum x , then further construction of a quorum y will intersect some node in quorum x . We can draw three lines along edges of the triangular

mesh such that each line is parallel with one of three sides and passes through the center of quorum x . The number of subtriangles generated depends on the location of the center of quorum x . There are three cases which should be taken into account according to where the center of quorum x is:

- (1) If the center of quorum x is a node inside the triangular mesh, three subtriangles will be generated. From the definition of quorums, we know that there will be one path from the center to the side in each subtriangle. These three paths will divide the triangle into three regions and nodes on the paths do not belong to any region. Figure 5.4-(a) is a demonstration. To prevent from intersecting quorum x , the center of quorum y should reside in any region. Since every region can provide nodes of two sides, only the other two regions can provide nodes of the third side which quorum y needs. The path from the center of quorum y to another region will intersect some node of quorum x since every two regions are separated by quorum x .
- (2) If the center of quorum x resides on any of three sides but is not any of three corners, two subtriangles are generated. There are three cases which should be taken into account: none, one or two of x 's subquorums such that all nodes in the subquorum are nodes of some side of the triangular mesh. When none of x 's subquorum resides at some side (ex., Figure 5.4-(b)), the triangular mesh is divided into three regions by quorum x . In a similar way as in case (1), we can show that no other quorum can be constructed without intersecting quorum x . If only one of x 's subquorum resides at some side, the triangle will be divided into two regions (ex., Figures 5.4-(c) and (d)). Since every region can provide nodes of two sides, only the other region can provide nodes of the third side which quorum y needs. The path from the center of quorum y to another region will intersect some node of quorum x . If two of x 's subquorums reside at some side of the triangle, quorum x is one side of the triangle (ex., Figure 5.4-(e)), and quorum y will intersect quorum x at least one node since quorum y must get at least one node from every side.
- (3) If the center of quorum x is one of three corners, only one subtriangle is generated and is just the triangle (ex. Figure 5.4-(f)). There are two cases to be taken into

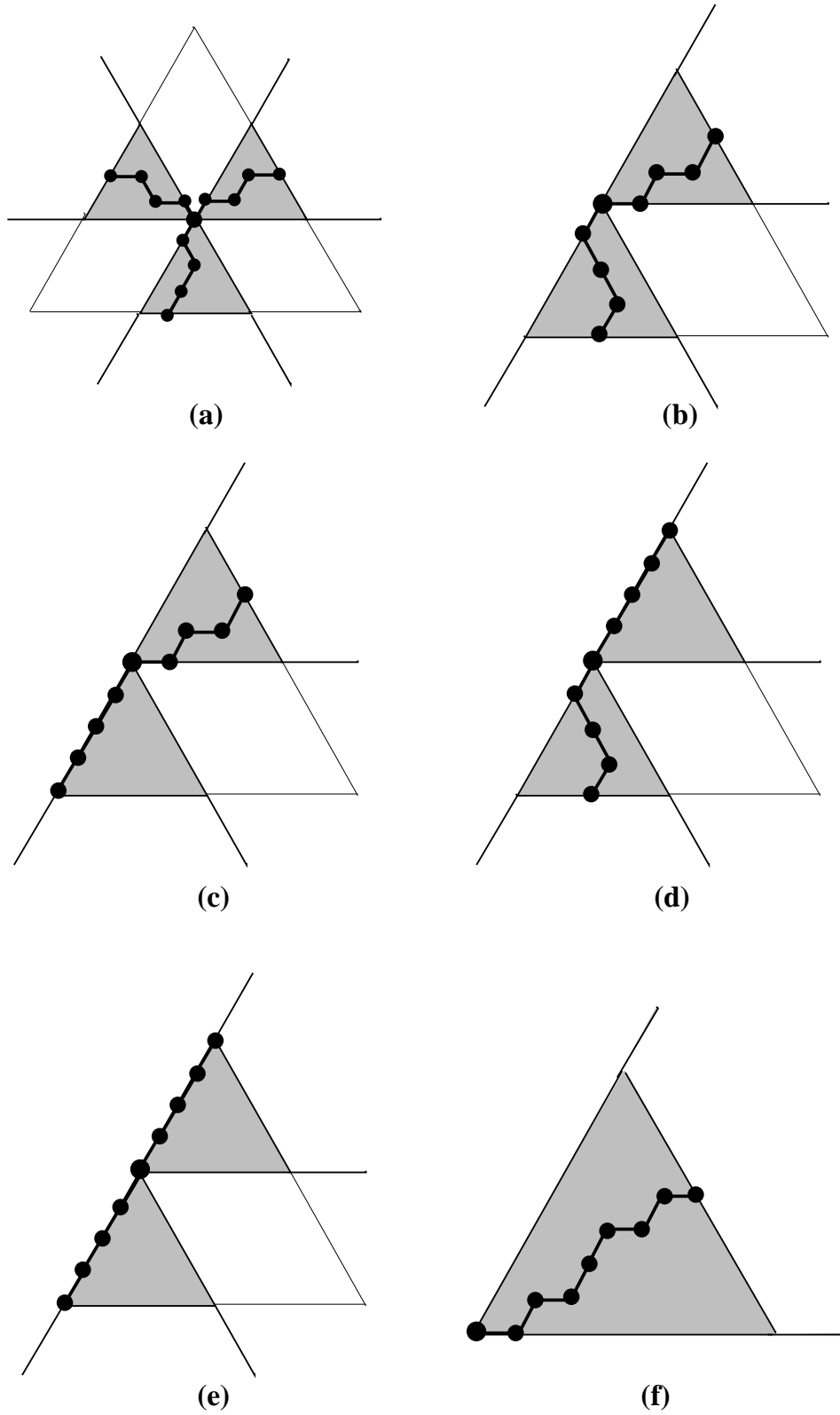


Figure 5.4 Examples of quorums in DTM: (a)-(f)

account: whether quorum x will contain all nodes of one side of the triangular mesh or not. In the affirmative case, quorum y will intersect quorum x at least one node since quorum y must get at least one node from every side. In the negative case, quorum x will divide the triangle mesh into two regions with each region contains nodes from at most two different sides. \square

5.4 An Extension

When the number of nodes in the system, N , can not be expressed exactly as $\frac{k(k+1)}{2}$, for some $k \in N$, we express N as $\frac{k(k+1)}{2} + r$ where $k, r \in N$ and $1 \leq r \leq k$. We first arrange $\frac{k(k+1)}{2}$ nodes into a k -triangular mesh, then each of the remaining r nodes is added next to the node on side 2 of the k -triangular mesh from bottom up, one at a time. Figure 5.5 illustrates some cases when $N \neq \frac{k(k+1)}{2}$. Those shaded nodes in Figure 5.5 constitute side 2 of those triangular meshes. Note that in Figure 5.5-(c), node 0 is assumed to be residing at both side 0 and side 2, while node 19 resides at side 2 only.

5.5 Properties of the Protocol

Theorem 5.2 *Given a k -triangular mesh, the quorum size of the proposed protocol is k , which is proportional to \sqrt{N} , where N is the number of nodes in the system.*

Proof. This can be proved in the same way as in the TM and TTM protocols. \square

Theorem 5.3 *The DTM protocol dominates the TTM and TM protocol when $k \geq 4$, $k \geq 3$, respectively.*

Proof. We denote the set of all quorums defined by the DTM, the TTM and the TM protocol in a k -triangular mesh as C_{DTM} , C_{TTM} and C_{TM} , respectively. Apply the similar approach as in Theorem 4.3, we can show that C_{DTM} is also a coterie. From the definition of quorums, we observe that $C_{TM} \subset C_{DTM}$ when $k \geq 3$ and $C_{TTM} \subset C_{DTM}$ when $k \geq 4$.

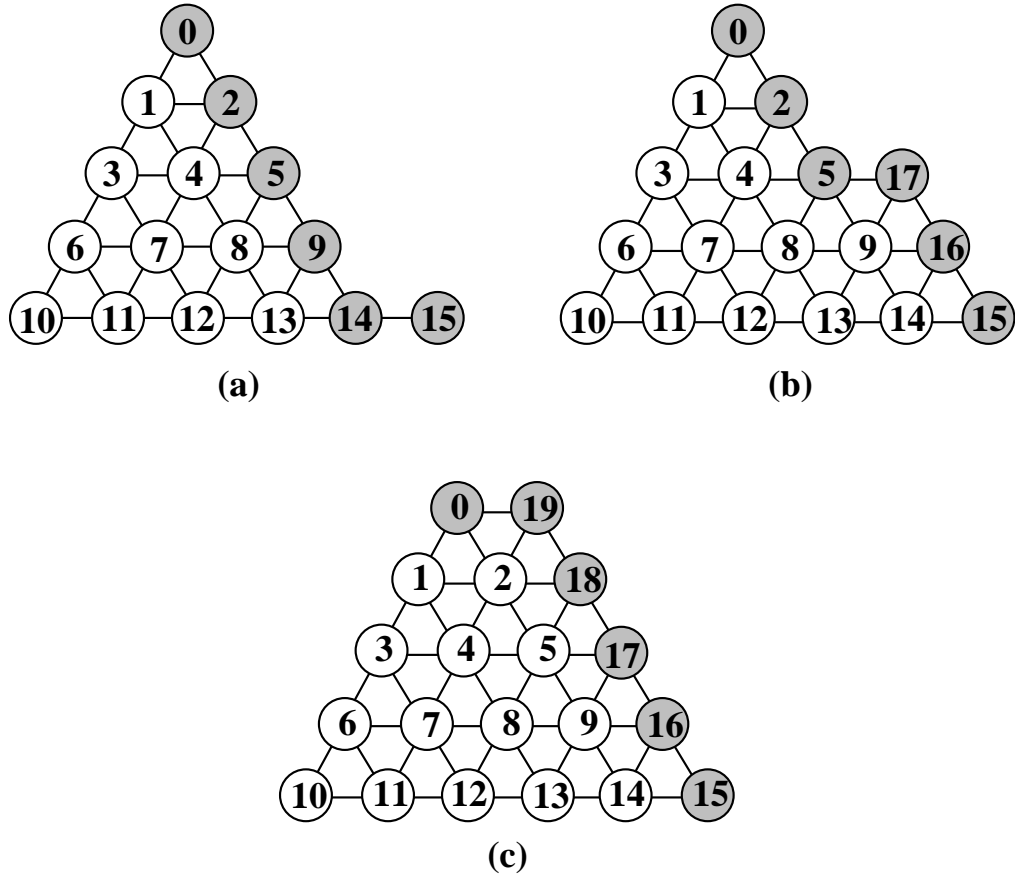


Figure 5.5 Examples when N can not be expressed as $\frac{k(k+1)}{2}$ exactly in DTM: (a) $N=16$ ($k=5, r=1$); (b) $N=18$ ($k=5, r=3$); (c) $N=20$ ($k=5, r=5$).

Apply Theorem 2.2, we know that the DTM protocol dominates the TM and the TTM protocol when $k \geq 3$ and $k \geq 4$, respectively. \square

Theorem 5.4 *The dynamic triangular mesh protocol does not have an equivalent vote assignment for $k \geq 3$.*

Proof. From the definition of quorums, we know that all the quorums in the TM protocol are also quorums in the DTM protocol. Since the TM protocol does not have an equivalent vote assignment as shown in Theorem 3.3, there is also no equivalent vote assignment for the DTM protocol. \square

CHAPTER VI

The Performance

In this Chapter, some aspects of distributed mutual exclusion protocols imposing logical structures are analyzed: quorum size, availability and fault tolerance. We compare these features of three triangular-mesh-based protocols, i.e., TM, TTM and DTM, with the tree [3], the HQC [20], and the grid [10] protocols. (Note that since the write-write intersection property holds, the write quorums of the grid protocol can be used to control accesses to a shared resource.)

6.1 Quorum Size

The number of messages required to construct a quorum is proportional to the size of the quorums. In the HQC protocol [20], it organizes nodes into a multi-level tree where each external node is mapped to a node in the system. A quorum at a level- i node is composed of a majority of its level- $(i + 1)$ children. To access a shared resource, the quorum at the root must be obtained. Therefore, the quorum size is $N^{0.63}$. In the tree protocol [3], it organizes nodes into a complete binary tree. A tree quorum consists of nodes on a path from the root to a leaf in the binary tree. If it fails to construct a quorum due to a node failure, the failed node is substituted by two paths each starting from one child of the failed node and ending at a leaf. Therefore, the quorum size is $\log_2 N$ and is increased up to $\lceil \frac{N+1}{2} \rceil$ as the number of node failures is increased. In the grid protocol [10], it organizes nodes into a $M_1 \times M_2 (= N)$ grid. A quorum contains one column of nodes and at least one node from each column of the grid. Therefore, the quorum size is $(M_1 + M_2 - 1)$, and is proportional to $O(\sqrt{N})$ when $M_1 = M_2$. Based on Theorem 2, the quorum size in our three triangular-mesh-based protocols is k , which is $\lceil \sqrt{2N} \rceil$.

Figure 6.1 shows a comparison of the quorum size of these six protocols, where nodes are organized into a $M_1 \times M_1 (= N)$ grid for the grid protocol. From these figure, we observe that the quorum size in our triangular mesh protocol is less than that in the grid protocol all the time, and is less than that of the HQC protocol when $N \geq 15$. Although the quorum size in the tree protocol is less than that in our protocol, the quorum size will be increased up to $\lceil \frac{N+1}{2} \rceil$ as the number of node failures is increased. Figure 6.2 shows a comparison of the quorum size of the triangular mesh protocol and the tree protocol as a function of the number of failed nodes, where we consider the worst case and $N = 15$. That is, the node fails starting from the root to the leaf, and from the left to the right. (Note that when $N = 15$, both the tree protocol and the three triangular-mesh-based protocols have a similar topology.) From this figure, we observe that the quorum size in our protocols will be less than that of the tree protocol when node failure occurs in the worst case. Moreover, when $N = 15$, and the number of failed node is greater than 7 in the worst case, there is no quorum that can be constructed in the tree protocol.

6.2 Availability

The availability is defined as the probability that a quorum can be constructed. We assume that each node is assumed to be independent available with probability p . Since given a number of node failures, the number of nodes prevented from constructing their associated quorums depends on the relative positions among those failed nodes, it is difficult to get a close form of availability in our triangular-mesh-based protocols. Therefore, the availability of the TM, TTM, and DTM protocols is computed by first generating all possible combinations for nodes to be available or unavailable, then we check each case to see whether a quorum can be constructed. If a quorum can be constructed, we add the possibility of occurrence of this case into the availability. The availability of the tree protocol and the HQC protocol are computed in the same way. For a $M_1 \times M_2$ grid, we use the formula: $(1 - (1 - p)^{M_1})^{M_2} - (1 - p^{M_1} - (1 - p)^{M_1})^{M_2}$ to get its availability [10].

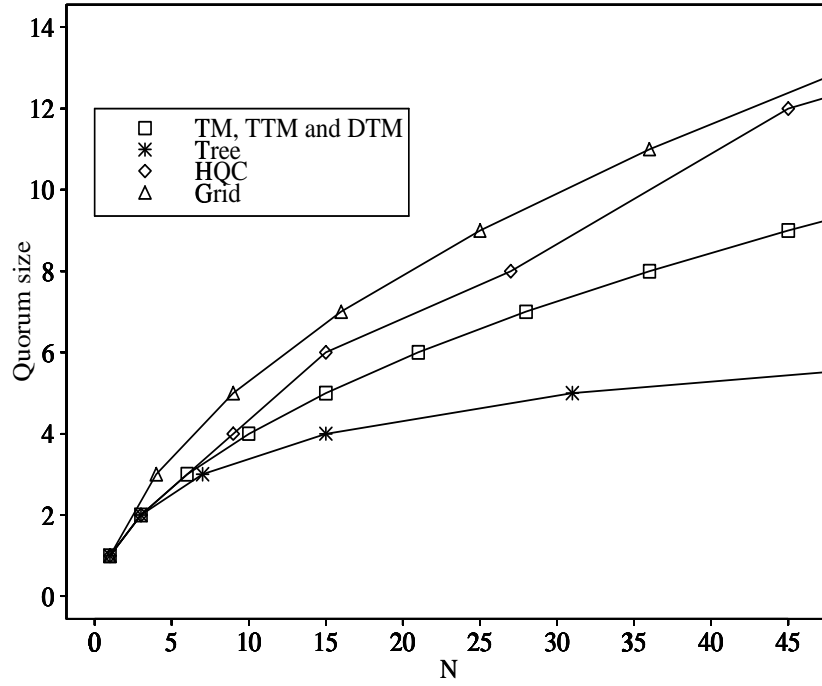


Figure 6.1 A comparison of the quorum size when no node failure occurs

The simulation results of the proposed protocols for $N = 6, 10, 15, 21$ and 28 are depicted in Table 6.1. Figure 6.3, 6.4 and 6.5 show the simulation results of availability with different N of TM, TTM and DTM protocols, respectively.

Figure 6.6 shows the availability of these four protocols when $N = 15$. From this figure, we observe that our triangular-mesh-based protocols have higher availability than the grid protocol all the time and the curve of the tree protocol comes close to that of the DTM protocol. Moreover, the TM, TTM and DTM protocols also can have higher availability than the tree protocol when $p \geq 0.93$, $p \geq 0.92$ and $p \geq 0.582$, respectively. Since in a k -triangular mesh of N nodes, the size of the coterie associated with TM, TTM and DTM is of $O(k^2)$, $O(k^2)$, and $O(k^2 2^{k-1})$, respectively, the DTM protocol has higher availability than TM and TTM.

| No. Failed Nodes | 0 | 1 | 2 | 3 | 4-6 |
|------------------|---|---|----|----|-----|
| DTM | 1 | 6 | 15 | 10 | 0 |
| TTM | 1 | 6 | 15 | 10 | 0 |
| TM | 1 | 6 | 15 | 9 | 0 |

(a)

| No. Failed Nodes | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7-10 |
|------------------|---|----|----|-----|-----|-----|----|------|
| DTM | 1 | 10 | 45 | 120 | 178 | 126 | 32 | 0 |
| TTM | 1 | 10 | 45 | 120 | 171 | 105 | 23 | 0 |
| TM | 1 | 10 | 45 | 120 | 168 | 93 | 17 | 0 |

(b)

| No. Failed Nodes | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11-15 |
|------------------|---|----|-----|-----|------|------|------|------|------|-----|----|-------|
| DTM | 1 | 15 | 105 | 455 | 1365 | 2907 | 4261 | 4050 | 2319 | 724 | 96 | 0 |
| TTM | 1 | 15 | 105 | 455 | 1362 | 2823 | 3742 | 3015 | 1449 | 388 | 45 | 0 |
| TM | 1 | 15 | 105 | 455 | 1362 | 2808 | 3569 | 2613 | 1107 | 261 | 27 | 0 |

(c)

| No. Failed Nodes | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|------------------|---|----|-----|------|------|-------|-------|--------|--------|--------|--------|--------|-------|-------|
| DTM | 1 | 21 | 210 | 1330 | 5985 | 20349 | 53992 | 112788 | 183468 | 226823 | 206994 | 135435 | 61546 | 18501 |
| TTM | 1 | 21 | 210 | 1330 | 5985 | 20325 | 53200 | 105264 | 152508 | 159516 | 120465 | 65511 | 25195 | 6540 |
| TM | 1 | 21 | 210 | 1330 | 5985 | 20325 | 53117 | 103488 | 143304 | 138970 | 95604 | 47172 | 16533 | 3942 |

| No. Failed Nodes | 14 | 15 | 16-21 |
|------------------|------|-----|-------|
| DTM | 3327 | 272 | 0 |
| TTM | 1032 | 75 | 0 |
| TM | 576 | 39 | 0 |

(d)

| No. Failed Nodes | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|------------------|---|----|-----|------|-------|-------|--------|---------|---------|---------|----------|----------|----------|
| DTM | 1 | 28 | 378 | 3276 | 20475 | 98280 | 376740 | 1183304 | 3094053 | 6781978 | 12440924 | 18928839 | 23570297 |
| TTM | 1 | 28 | 378 | 3276 | 20475 | 98280 | 376576 | 1176741 | 3005769 | 6199505 | 10188252 | 13273335 | 13737600 |
| TM | 1 | 28 | 378 | 3276 | 20475 | 98280 | 376576 | 1176166 | 2988780 | 6047134 | 9550216 | 11729835 | 11302856 |

| No. Failed Nodes | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22-28 |
|------------------|----------|----------|----------|---------|---------|--------|--------|-------|-----|-------|
| DTM | 23675703 | 18928401 | 11895930 | 5796959 | 2148540 | 586750 | 111583 | 13221 | 736 | 0 |
| TTM | 11341827 | 7483728 | 3934446 | 1631490 | 523158 | 125370 | 21163 | 2247 | 113 | 0 |
| TM | 8643071 | 5283687 | 2582720 | 1000785 | 301500 | 68225 | 10923 | 1104 | 53 | 0 |

(e)

Table 6.1 The simulation results of TM, TTM and DTM protocols: (a) $N = 6$; (b) $N = 10$; (c) $N = 15$; (d) $N = 21$; (e) $N = 28$.

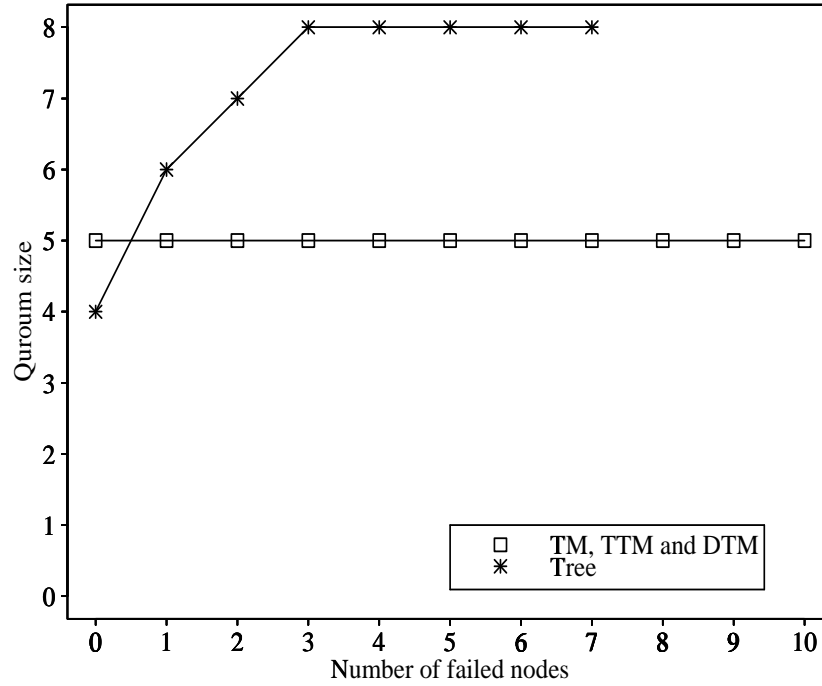


Figure 6.2 A comparison of the quorum size when node failures occur in the worst case ($N = 15$)

6.3 Fault Tolerance

Our simulation reveals that in the worst case, the TM and TTM protocols can tolerate up to $(k - 1)$ node failures when $k \leq 4$, and up to $(k - 2)$ node failures when $k \geq 5$, where $N = \frac{k(k+1)}{2}$. This is because that there exist some patterns of $(k - 1)$ node failures which disable all quorum constructions. For example, in a k -triangular mesh, $k \geq 5$, the set consists of failed nodes whose (x, y) -tuples satisfy one of the following conditions will make all quorums unavailable:

for $i = 1, \dots, k - 1$,

- (1) $(\frac{i+1}{2}, \frac{i-1}{2})$, i is odd;
- (2) $(\frac{i-2}{2}, \frac{i+2}{2})$, i is even.

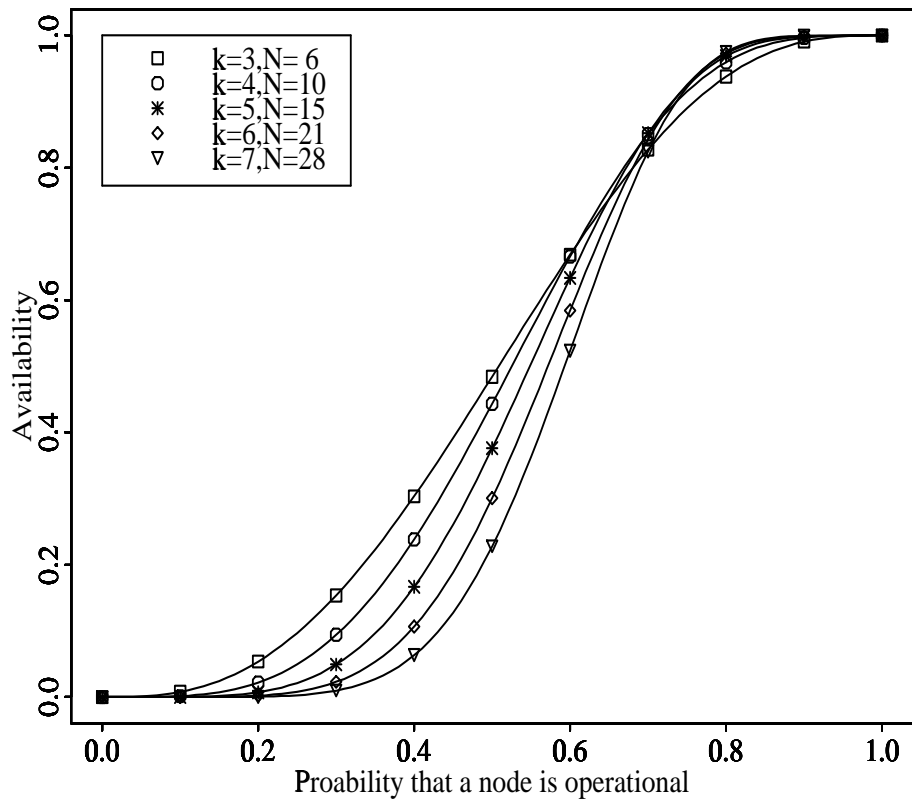


Figure 6.3 The availability of TM protocol with different N

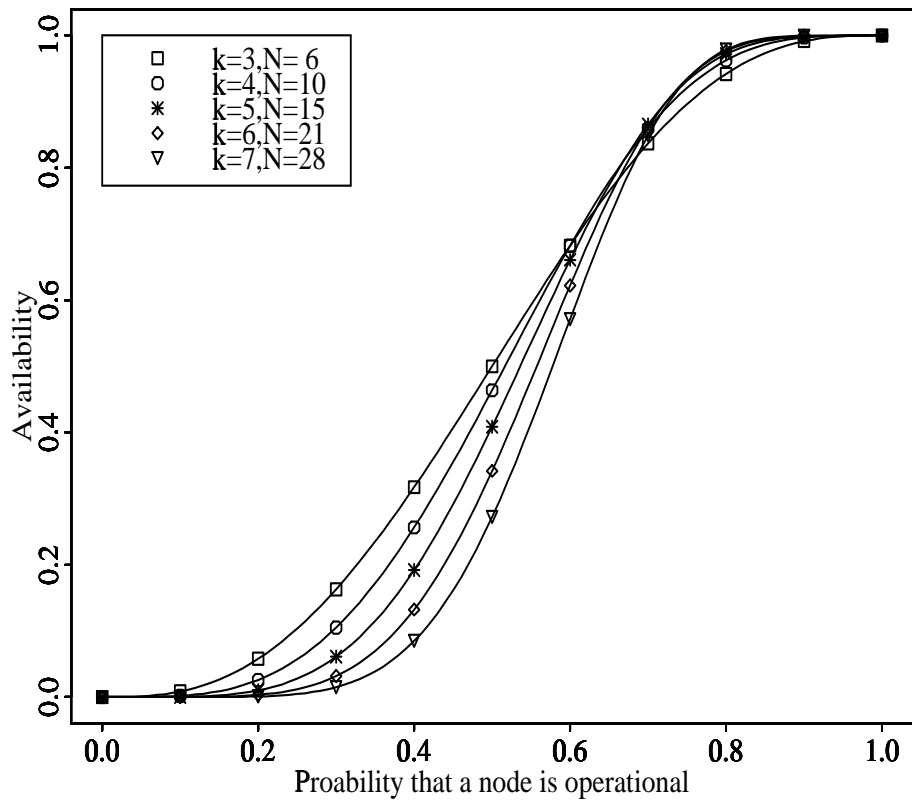


Figure 6.4 The availability of TTM protocol with different N

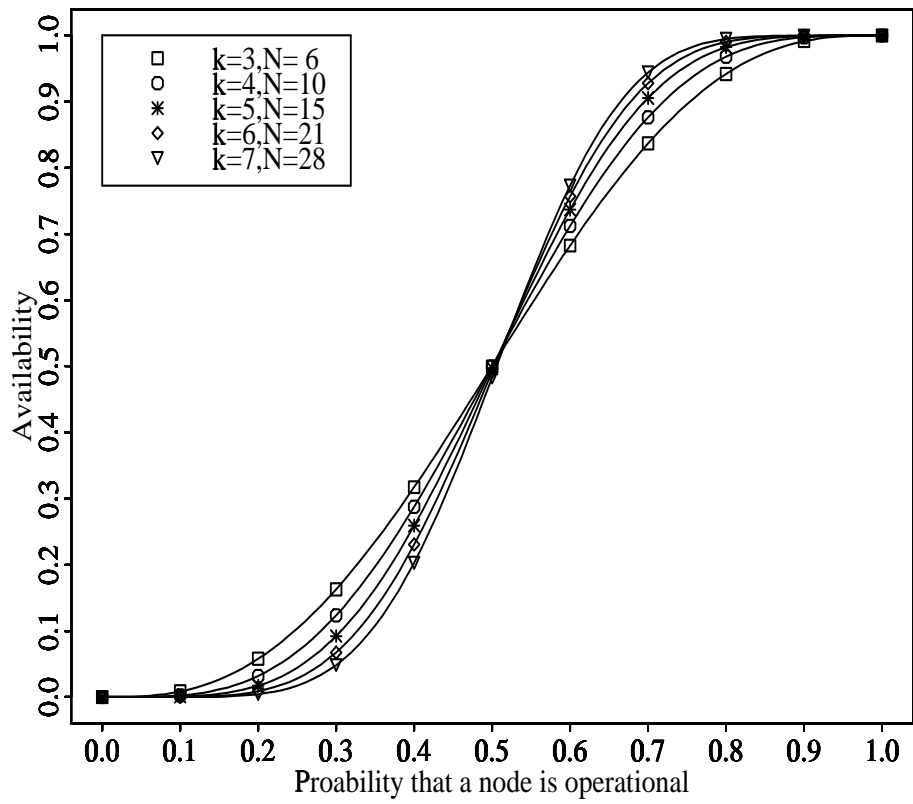


Figure 6.5 The availability of DTM protocol with different N

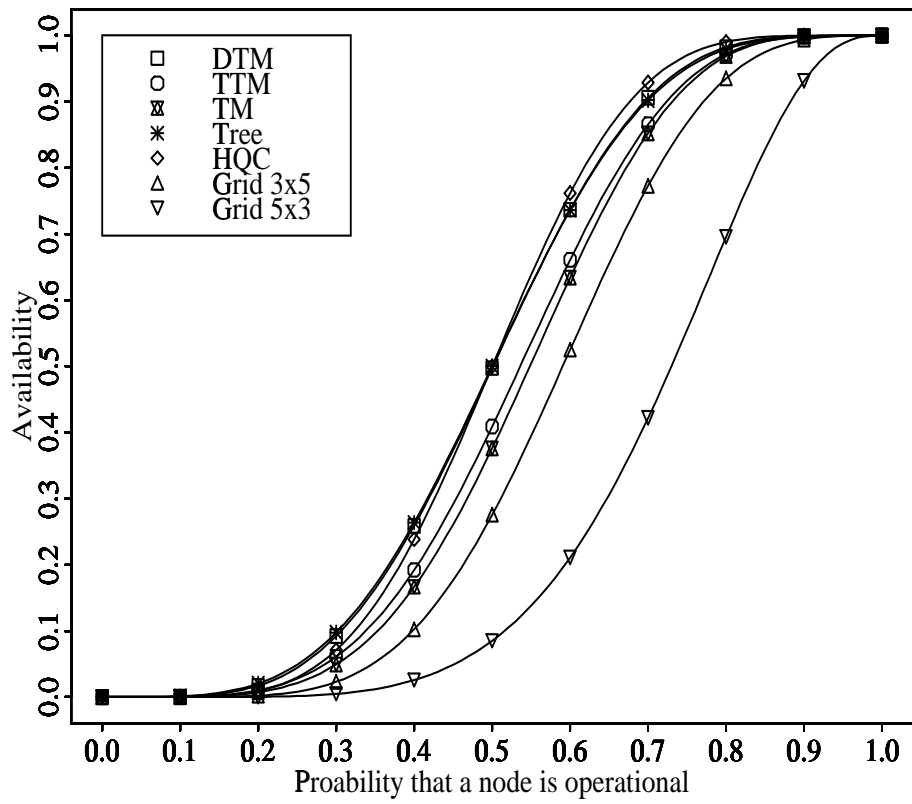


Figure 6.6 A comparison of availability when N=15

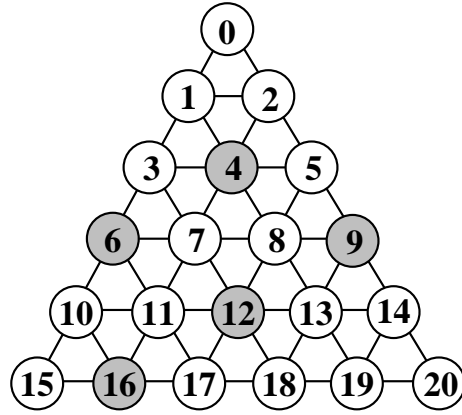


Figure 6.7 An example of node failures which make all quorums of TM and TTM protocols unavailable

Figure 6.7 shows a set of 5 failed nodes: $\{ 4, 6, 9, 12, 16 \}$ which makes all quorum constructions impossible. However, in the worst case, the DTM protocol can always tolerate up to $(k - 1)$ node failures, which has better fault tolerance than TM and TTM protocols.

The above discussion can be summarized in Table 6.2 based on six criteria, where we consider a $M_1 \times M_2 (= N)$ grid. The first two criteria are the quorum sizes in the best and worst cases, respectively. The quorum size of the tree protocol varies from $\log_2 N$ to $\lceil \frac{N+1}{2} \rceil$ as the number of node failures is increased. The triangular-mesh-based, the HQC, and the grid protocols have constant quorum size. The third criterion is the impact that a single node failure would have on the size of a quorum. In the grid protocol, if the failed node belongs to a column included in the quorum, then another $(M_1 - 1)$ nodes are needed to form another quorum. In the tree protocol, the failure of the root will result in a requirement of another $\log_2 N$ nodes to construct a new quorum. In the triangular-mesh-based protocols, when $k \geq 5$, in the worst case, there are at least $\lceil \frac{k-3}{4} \rceil + 1$ nodes reusable, therefore, we need another $(k - \lceil \frac{k-3}{4} \rceil - 1)$ nodes to construct another quorum. The fourth criterion is whether the protocol is a fully distributed one. The tree protocol assigns greater burden to the nodes with smaller level numbers than those with larger level numbers. Therefore, the tree protocol is not a fully distributed protocol while the triangular-mesh-based protocols, and the other two protocols are fully distributed ones. The last two criteria are the number of failed nodes which does not halt the system in the best and worst case, respectively. While in the best case, all these six protocols can be fault-tolerant up to all node failures

except those nodes which have already constructed a quorum. While in the worst case, the tree and the HQC protocol can be fault-tolerant up to (*the quorum size* $- 1$) failures. The grid protocol can be fault-tolerant to $(\min\{M_1, M_2\} - 1)$ failures. According to the simulation results, the TM, TTM and DTM protocols can tolerate up to $(k - 2)$, $(k - 2)$ and $(k - 1)$ node failures, respectively.

| | HQC | Tree | Grid | TM | TTM | DTM |
|----------------------------------------------|----------------|-------------------------------|------------------------|--------------------------------------------------------------------|--------------------------------------------------------------------|--------------------------------------------------------------------|
| (1) quorum size (best case) | $N^{0.63}$ | $\log_2 N$ | $M_1 + M_2 - 1$ | $\lfloor \sqrt{2N} \rfloor$ | $\lfloor \sqrt{2N} \rfloor$ | $\lfloor \sqrt{2N} \rfloor$ |
| (2) quorum size (worst case) | $N^{0.63}$ | $\lceil \frac{N+1}{2} \rceil$ | $M_1 + M_2 - 1$ | $\lfloor \sqrt{2N} \rfloor$ | $\lfloor \sqrt{2N} \rfloor$ | $\lfloor \sqrt{2N} \rfloor$ |
| (3) cost of one node failure (worst case) | 1 | $\log_2 N$ | $M_1 - 1$ | $\lfloor \sqrt{2N} \rfloor - \lfloor \frac{[2N]-3}{4} \rfloor - 1$ | $\lfloor \sqrt{2N} \rfloor - \lfloor \frac{[2N]-3}{4} \rfloor - 1$ | $\lfloor \sqrt{2N} \rfloor - \lfloor \frac{[2N]-3}{4} \rfloor - 1$ |
| (4) fully distributed? | yes | no | yes | yes | yes | yes |
| (5) fault tolerance (best case) | $N - N^{0.63}$ | $N - \log_2 N$ | $N - (M_1 + M_2) + 1$ | $N - \lfloor \sqrt{2N} \rfloor$ | $N - \lfloor \sqrt{2N} \rfloor$ | $N - \lfloor \sqrt{2N} \rfloor$ |
| (6) fault tolerance (worst case) | $N^{0.63} - 1$ | $\log_2 N - 1$ | $\min\{M_1, M_2\} - 1$ | $\lfloor \sqrt{2N} \rfloor - 2$ | $\lfloor \sqrt{2N} \rfloor - 2$ | $\lfloor \sqrt{2N} \rfloor - 1$ |

Table 6.2 A comparison of six mutual exclusion algorithms imposing logical structures

CHAPTER VII

Conclusion

In this Chapter, we give a summary of the thesis and point out some future directions.

7.1 Summary

In the problem of mutual exclusion, concurrent access to a shared resource or the Critical Section (CS) must be synchronized such that at any time only one process can access the CS. In a distributed system, due to the lack of both in shared memory and a global clock, and due to unpredictable message delay, the design of a distributed mutual exclusion protocol that is free from deadlock and starvation, is much more complex than that in a centralized system.

To make distributed mutual exclusion protocols fault-tolerant to node and communication failures, many researches apply the replica control strategies to achieve mutual exclusion. The majority voting protocol [40] and the quorum consensus protocol [15] are such examples. However, these protocols require high communication cost which is $O(N)$ due to the large quorum size.

To reduce the overhead of achieving mutual exclusion while supporting fault tolerance, many protocols imposing a logical structure on the network are proposed [3, 10, 20]. The hierarchical quorum consensus protocol (HQC) [20] requires $O(N^{0.63})$ messages, the tree quorum protocol [3] requires $O(\log N)$ messages in the best case and degrades gracefully, and the grid protocol [10] requires $O(\sqrt{N})$ messages. All the quorums constructed from these protocols can be used to replace the set of nodes in Maekawa's protocol [24] to achieve mutual exclusion. (Note that in Maekawa's protocol, only one set is associated with each node, which makes the protocol non-tolerant to failures. While these three protocols

[3, 10, 20] can support a node with several alternative sets.) In Chapter 2, we have given a survey of the replica control protocols described above.

Next, we have presented the three proposed triangular-mesh-based protocols, i.e., the triangular mesh protocol, the triple triangular mesh protocol and the dynamic triangular mesh protocol in Chapter 3, Chapter 4 and Chapter 5, respectively. In the triangular mesh protocol, we associate with each node two special patterns. The nodes in each special pattern constitute a quorum. In the triple triangular mesh protocol, a quorum contains nodes from some side of each of three subtriangles in the triangular mesh. In the dynamic triangular mesh protocol, three dynamic paths constitute a quorum in the given triangular mesh. The quorum size of these proposed protocols is K that is $O(\sqrt{N})$, where where $N = \frac{K*(K+1)}{2}$ is the number of nodes in the system. We also have proved that none of these three triangular-mesh-based protocols has an equivalent vote assignment. Moreover, the TTM protocol dominates the TM protocol and the DTM protocol dominates the TM and the TTM protocols.

In Chapter 6, some aspects of distributed mutual exclusion protocols imposing logical structures have been analyzed: quorum size, availability and fault tolerance. We have compared these features of three triangular-mesh-based protocols, i.e., TM, TTM and DTM, with the tree [3], the HQC [20], and the grid [10] protocols. From our simulation study, we have shown that these three proposed protocols can have higher availability than the grid protocol. Moreover, the quorum size of the proposed protocols will be less than that in the HQC protocol when N is greater than or equal to 15 and less than that in the tree quorum protocol when node failures of some sort exist. We also have observed that when $N = 15$, the curve of the tree protocol comes close to that of the DTM protocol. Moreover, the TM, TTM and DTM protocols also can have higher availability than the tree protocol when $p \geq 0.93$, $p \geq 0.92$ and $p \geq 0.582$, respectively. In the worst case, the triangular mesh protocol, the triple triangular mesh protocol and the dynamic triangular mesh protocol are fault-tolerant up to $(K - 2)$, $(K - 2)$ and $(K - 1)$ site and communication failures, respectively.

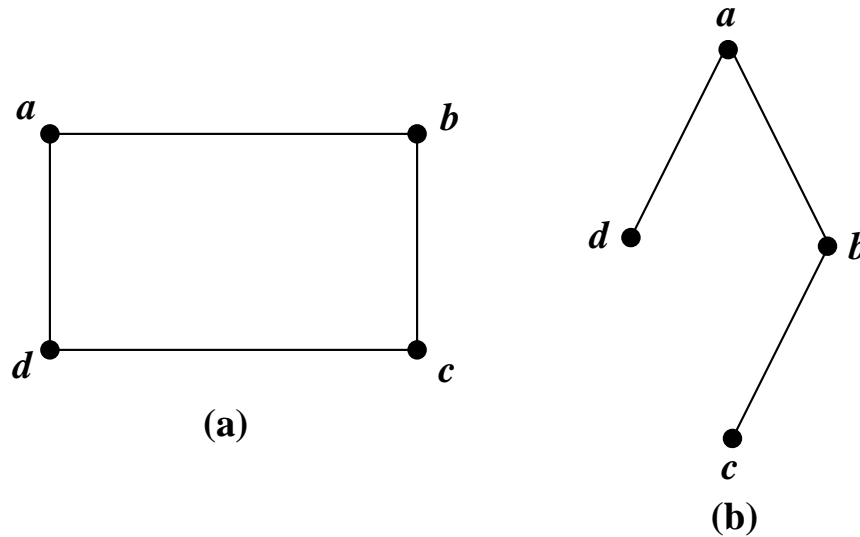


Figure 7.1 Systems used to illustrate how the topology impacts the design of distributed mutual exclusion protocols: (a)-(b)

7.2 Future Work

In [14], Barbara and Garcia-Molina show that the topology of the system can have a definite influence over the coterie or vote assignment. They use two measures to compare different vote assignments:

- (1) *Node Vulnerability* of a system with vote assignment V is the minimum number of crashed nodes that produce a halted state.
- (2) *Edge Vulnerability* of a system with vote assignment V is the minimum number of link failures that produce a halted state.

To illustrate the influence of the assignment over these two measures, let's consider the system of Figure 7.2-(a). If every node is assigned one vote, it is enough to remove two edges ((a, b), (c, d) for instance) to halt the system operation. On the other hand, if node a is given two votes while the rest receive one vote, it is necessary to remove 3 edges to stop the operation. Thus, the edge vulnerability of the first assignment is 2 and the second is 3. Consequently, from the point of view of edge vulnerability, the second assignment is superior.

Consider the system shown in Figure 7.2-(b). If we assign the coterie $R = \{\{a, b\}, \{a, c\}, \{b, c\}\}$, We notice that the group $\{a, c\}$ cannot exist by itself, so we replace it by $\{a, b, c\}$ giving $R_1 = \{\{a, b\}, \{a, b, c\}, \{b, c\}\}$. This is not a coterie; purging it renders $R_2 = \{\{a, b\}, \{b, c\}\}$. This in turn is dominated by $S = \{\{b\}\}$. We observe that

- (1) S has the same node vulnerability as R , since in both cases it is enough to crash node b to disrupt system operation. Since S is a singleton coterie, it has higher edge vulnerability than either R_1 or R_2 (which have 2).
- (2) The group that substituted $\{a, c\}$ ($\{a, b, c\}$) did not survive the purging; so R_2 has no group representing it.
- (3) The 2-partition ab/cd disrupts the system.

The above discussion shows how the *topology* impacts the design of distributed mutual exclusion protocols. To design a protocol which takes into consideration the topology of the underlying system is the future research work.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] D. Agrawal and A. EL Abbadi, "An Efficient Solution to the Distributed Mutual Exclusion Problem," in *Proc. of the 8th ACM Symposium on Principles of Distributed Computing*, pp. 193-200, 1989.
- [2] D. Agrawal and A. EL Abbadi, "Exploiting Logical Structures in Replicated Databases," *Information Processing Letters*, Vol. 33, No. 5, pp. 255-260, Jan. 1990.
- [3] D. Agrawal and A. EL Abbadi, "An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion," *ACM Transactions on Computer Systems*, Vol. 9, No. 1, pp. 1-20, Feb. 1991.
- [4] P. A. Alsberg and J. D. Day, "A Principle for Resilient Sharing of Distributed Resources," in *Proc. of the Sixteenth International Conference on Very Large Data Bases*, pp. 562-570, 1976.
- [5] D. Barbara, H. Garcia-Molina, and A. Spauster, "Increasing Availability Under Mutual Exclusion Constraints with Dynamic Vote Reassignment," *ACM Transactions on Computer Systems*, Vol. 7, No. 4, pp. 394-426, Nov. 1989.
- [6] O. S. F. Carvalho and G. Roucairol, "On Mutual Exclusion in Computer Networks, Technical Correspondence," *Communications of the ACM*, Vol. 26, No. 2, pp. 146-148, Feb. 1983.
- [7] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing Vote and Quorum Assignments for Reading and Writing Replicated Data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 3, pp. 387-397, Sep. 1989.
- [8] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Multi-Dimensional Voting: A General Method for Implementing Synchronization in Distributed Systems," in *Proc. of the 6th Int. Conf. on Data Engineering*, pp. 362-369, 1990.
- [9] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data," in *Proc. of the 6th International Conference on Data Engineering*, pp. 438-445, 1990.

- [10] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6, pp. 582-592, Dec. 1992.
- [11] D. Davcev and W. Burkhard, "Consistency and Recovery Control for Replicated Data," in *Proc. 10th Symp. Operating System Principles*, pp. 87-96, 1985.
- [12] D. Eager and K. C. Sevcik, "Achieving Robustness in Distributed Database Systems," *ACM Transactions on Database Systems*, Vol. 8, No. 3, pp. 354-381, Sep. 1983.
- [13] H. Garcia-Molina and D. Barbara, "How to Assign Votes in a Distributed System," *Journal of the Association for Computing Machinery*, Vol. 32, No. 4, pp. 841-860, Oct. 1985.
- [14] H. Garcia-Molina and D. Barbara, "The Vulnerability of Vote Assignments," *ACM Transactions on Computer Systems*, Vol. 4, No. 3, pp. 187-213, Aug. 1986.
- [15] D. K. Gifford, "Weighted Voting for Replicated Data," in *7th Symposium on Operating Systems Principles*, pp. 150-159, 1979.
- [16] M. Herlihy, "Dynamic Quorum Adjustment for Partitioned Data," *ACM Transactions on Database Systems*, Vol. 12, No. 2, pp. 170-194, Jun. 1987.
- [17] T. Ibaraki and T. Kameda, "A Theory of Coterics: Mutual Exclusion in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 7, pp. 779-794, Jul. 1993.
- [18] S. Jajodia and D. Mutchler, "Dynamic Voting," in *Proc. of 1985 ACM-SIGMOD*, pp. 227-238, 1985.
- [19] S. Jajodia and D. Mutchler, "Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database," *ACM Transactions on Database Systems*, Vol. 15, No. 2, pp. 230-280, Jun. 1990.
- [20] A. Kumar, "Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data," *IEEE Transactions on Computers*, Vol. 40, No. 9, pp. 996-1004, Sep. 1991.
- [21] A. Kumar and A. Segev, "Optimizing and Evaluating Algorithms for Replicated Data Concurrency Control," in *Proc. of 9th Symposium on Distributed Computing Systems*, pp. 101-109, 1989.
- [22] L. Lamport, "Time, Clocks and Ordering of Events in Distributed Systems," *Communications of the ACM*, Vol. 21, No. 7, pp. 558-565, July 1978.

- [23] L. Lamport, "The Implementation of Reliable Distributed Multiprocess Systems," *Computer Networks*, Vol. 2, pp. 95-114, 1978.
- [24] M. Maekawa, "A \sqrt{N} Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp. 145-159, May 1985.
- [25] M. Raynal, "A Simple Taxonomy for Distributed Mutual Exclusion Algorithms," *ACM Operating System Review*, Vol. 23, No. 2, pp. 47-51, 1991.
- [26] J. Misra, "Detecting Termination of Distributed Computations Using Markers," in *Proc. of the Second ACM Symposium on Principles of Distributed Computing*, pp. 290-94, 1983.
- [27] M. Naimi and M. Trehel, "An Improvement of the $\log(N)$ Distributed Algorithm for Mutual Exclusion," in *Proc. of the 7th International Conference on Distributed Computing Systems*, pp. 371-375, Sept. 1987.
- [28] M. L. Neilsen and M. Mizuno, "Coterie Join Algorithm," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 5, pp. 582-590, Sep. 1992.
- [29] S. Nishio, K. F. Li, and E. G. Manning, "A Resilient Mutual Exclusion Algorithm for Computer Network," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 344-355, July 1990.
- [30] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in The Presence of Faults," *Journal of the Association for Computing Machinery*, Vol. 27, No. 2, pp. 228-234, Apr. 1980.
- [31] K. Raymond, "A Tree-Based Algorithm for Distributed Mutual Exclusion," *ACM Transactions on Computer Systems*, Vol. 7, No. 1, pp. 61-77, February 1989.
- [32] G. Ricart and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," *Communications of the ACM*, Vol. 24, No. 1, pp. 9-17, Jan. 1981.
- [33] B. A. Sanders, "The Information Structure of Distributed Mutual Exclusion Algorithms," *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 284-299, August 1987.
- [34] D. Shou and S. Wang, "A New Transformation Method for Nondominated Coterie Design," *Information Sciences*, Vol. 74, pp. 223-246, 1993.
- [35] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, Mass., 1982.
- [36] M. Singhal, "A Heuristically-Aided Algorithm for Mutual Exclusion in Distributed Systems," *IEEE Transactions on Computers*, Vol. 38, No. 5, pp. 651-662, May 1989.

- [37] M. Singhal, "A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 1, pp. 121-125, Jan. 1992.
- [38] M. Singhal, "A Taxonomy of Distributed Mutual Exclusion," *Journal of Parallel and Distributed Computing*, Vol. 18, pp. 94-101, 1993.
- [39] I. Suzuki and T. Kasami, "A Distributed Mutual Exclusion Algorithm," *ACM Transactions on Computer Systems*, Vol. 3, No. 4, pp. 344-349, Nov. 1985.
- [40] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," *ACM Transactions on Database Systems*, Vol. 4, No. 2, pp. 180-209, Jun. 1979.