2.1. Introduction

In this chapter, we present an overview of various permission-based distributed mutual exclusion algorithms (PBDME algorithms) and their comparative performance. We have tried to include a good number of algorithms that have contributed to the development of PBDME algorithms. We have not discussed the algorithms in detail. Instead, we have tried to bring out the essential features, which make a particular algorithm distinct from others. In fact, we have laid more emphasis on the technique being used to achieve mutual exclusion. But before we discuss PBDME algorithms in detail, we take some time to identify the characteristics that a DME algorithm is expected to possess in general. We begin with a historical background of mutual exclusion.

2.1.1. History of Mutual Exclusion

The origin of the mutual exclusion problem can be traced back to 1965 when Dijkstra [D-65] described and solved the mutual exclusion problem. Dijkstra
stated that any solution to the mutual exclusion problem must satisfy the constraints:

i) No assumption is made about the instructions or the numbers of processors supported by the machine except that basic instructions such as read, write or test a memory location are executed atomically.

ii) No assumption is made about the execution speed of the competing processes, except that it is non-zero.

iii) When a process is in the non-critical section, it cannot prevent another from entering the critical section.

iv) It must not be possible for a process seeking access to a critical section to be delayed indefinitely.

Dijkstra's algorithm guaranteed mutual exclusion and was deadlock free but it did not guarantee fairness. That is, it was possible that a process seeking access to critical section waited indefinitely while others entered and exited their critical sections frequently. In other words, the algorithm ran the risk of starvation. Knuth [K-66] proposed the first fair solution to the mutual exclusion problem. In addition to the four constraints specified by Dijkstra, Knuth's algorithm also satisfied

v) A process attempting to enter its critical section will be able to do so within a finite time.

Thereafter, a number of algorithms were proposed, which guaranteed mutual exclusion and were deadlock-free and fair. Each of these algorithms aimed at improving performance in terms of synchronization delay. Some of these
algorithms are De Bruijn's algorithm [DB-67], Eisenberg and McGuire's algorithm [EM-72] and Peterson's algorithm [P-81]. All these algorithms were designed for centralized systems, the systems possessing a central memory that all processes can access simultaneously for reading and writing. Moreover, their basic principle was the same: they all used one or more shared variables to achieve mutual exclusion, just similar to the case of single-processor systems, where critical regions are protected using semaphores, monitors and similar constructs. An elegant discussion on the solutions of mutual exclusion problem in centralized systems and a lucid account of the development of the subject can be found in [R-86].

2.1.2. Distributed Mutual Exclusion Algorithms

Due to lack of a common shared memory, the problem of mutual exclusion becomes much more complex in the case of distributed systems as compared to the centralized systems and needs special treatment. Of course, when we use the term 'distributed system', we mean message-passing networks in which every process has a local memory and there is no central controlling unit. The systems in which there exists a central control unit (process) to which all sites submit request messages for entry into CS and send release messages after exiting CS, are not included because, in effect, such systems are closer to the centralized systems. Moreover, in such systems, the failure of a single process (the centralized king) leads to the total failure of the system.

A number of solutions have been provided to the mutual exclusion problem in distributed systems. Algorithms, which ensure mutual exclusion in
distributed systems, are called **Distributed mutual exclusion algorithms (DME algorithms)**. A good DME algorithm, besides providing mutual exclusion, should take care that there are no deadlocks and starvation does not occur. A **deadlock** is a state of the system when no node is in its CS and yet no requesting node can proceed to its CS. **Starvation** is said to occur when some node waits indefinitely to enter its CS, even though other nodes are entering and exiting their critical sections.

### 2.1.2.1. Performance Metrics of Distributed Mutual Exclusion Algorithms

The performance of a distributed mutual exclusion algorithm can be evaluated in terms of a number of metrics. Message Complexity (MC) and Synchronization Delay (D) are two parameters which can be used to compare the performance of various DME algorithms. **Message complexity** of a DME algorithm is the number of messages exchanged by a process per CS entry. **Synchronization delay** is the average time delay in granting CS, which is the period of time between the instant a site invokes mutual exclusion and the instant when it enters CS. The execution time of the instructions in the algorithm is assumed to be negligible as compared to the message transition time. Other criteria that can be used to compare the performance of DME algorithms are fault tolerance and availability. The **availability** of a DME algorithm is the probability that the critical section can be entered in the presence of failures. Its **fault-tolerance** is the maximal number of nodes that can fail before it becomes impossible for any node to enter its CS. Thus, a good DME algorithm must be safe (it shall ensure mutual exclusion), live (the system should make progress towards the execution of CS and a deadlock situation shall not occur) and fair (it shall not be biased against or in favour of a
node and each request shall eventually be satisfied). It should have low message complexity and synchronization delay. High fault-tolerance and availability add to the qualities of a good DME algorithm.

2.1.2.2. Classification of Distributed Mutual Exclusion Algorithms

A number of DME algorithms have been proposed, all aiming at improved performance with respect to one performance measure or the other. Based on the technique used, DME algorithms can be classified as Token-based algorithms and Permission-based algorithms as suggested by Raynal [R-91] or as Token-based algorithms and Non-token-based algorithms as suggested by Singhal [S-93]. Moreover, these algorithms can be static or dynamic, logical structure based or broadcast based. A DME algorithm is static if it does not remember the history of CS execution. In a static algorithm, nodes need not keep information about the current state of the system. On the other hand, in the case of a dynamic algorithm, nodes keep track of the current state of the system and the algorithm has an inbuilt mechanism to make decision on the basis of this knowledge. In logical structure based algorithms, the sites in the system are assumed to be arranged in a logical configuration like tree, ring etc. and messages are passed from one site to another along the edges of the logical structure imposed. In the case of broadcast based algorithms, no such structure is assumed and the requesting site sends messages to other sites in parallel, i.e., the message is broadcasted.

Token-based algorithms achieve mutual exclusion through a privilege message called a token, which is shared among the sites. A site can enter CS if and only if it is in possession of the token. Since the token is unique, mutual
exclusion is guaranteed. Fair scheduling of token among competing sites, detecting the loss of token and regenerating a unique token are some of the major design issues of the token-based mutual exclusion algorithms. Although token-based algorithms are generally faster than the non-token-based algorithms, produce lesser message traffic and are not deadlock prone, their resiliency to failures is poor [G-98]. If the site holding the token fails, complex token-regeneration protocols have to be executed and when this site recovers, it has to undergo a recovery phase during which it is informed that the token, it was holding prior to its failure, has been invalidated. Token-based algorithms can be broadcast based or logical structure based, static or dynamic. Some examples of Token-based algorithms are Helary et al.’s [HPR-88] and Suzuki-Kasami’s [SK-85] algorithms (broadcast based, static), Singhal’s [S-89] and Yan et al.’s [YZY-96] algorithms (broadcast based, dynamic), Raymond’s [R-89a] and Neilsen-Mizuno’s [NM-91] algorithms (logical structure based, static) and Chang et al.’s [CSL-90], Helary et al.’s [HMR-94] and Naimi et al.’s [NTA-96] algorithms (logical structure based, dynamic). Saxena and Gupta in [SG-99] give an algorithm that optimizes synchronization delay. A performance comparison of token- and tree- based distributed mutual exclusion algorithms can be found in [SG-98]. A simulation study for token-based DME algorithms has been carried out by Chang in [C-96].

Permission-based algorithms require rounds of message exchange among the nodes to obtain the permission to execute CS. They can be further subdivided into voting-based algorithms and coterie-based algorithms. The basic idea on which permission-based algorithms are based is as follows:
When a process wants to enter its CS, it asks other nodes for their permission. A process on receiving a request, grants its permission if it is not interested in CS. If it is interested in CS, the priority of the incoming request is established against its own request. Generally, priority decisions are made using timestamps.

In voting-based algorithms, each site in the system is assigned a vote (a non-negative integer). A site seeking access to CS must obtain permission from an appropriate number of nodes, i.e., a number of nodes whose total votes constitute a majority of the total number of votes assigned to the system. In coterie-based algorithms, a collection of sets (of the nodes of the system), called a coterie, is attached to the system. A node seeking access to CS must obtain permission from each and every node of a set from the coterie. Each of the two categories - voting-based algorithms and coterie-based algorithms - may further be subdivided into static and dynamic algorithms. Figure 2.1 depicts the classification of DME algorithms that has emerged during nearly two decades of the development of the subject. We shall organize our discussion on various DME algorithms as per this classification in the sections to follow. We discuss vote-assignments and coteries and related concepts in detail, in Section 2.4. But before we proceed further, we give, in the next two sections, the system model which most of the DME algorithms use and the concepts of logical clocks and timestamps as given by Lamport for ordering messages.
2.2. System Model

We present, here, a model of the system, which, in general, has been used by almost every DME algorithm. A distributed system of $N$ nodes consists of $N$ geographically dispersed autonomous sites connected via a communication network. The sites communicate only through message passing and do not share a common memory. The communication network is assumed to be logically fully connected, i.e., every node can communicate with every other node. The communication medium is reliable, i.e., all messages sent are eventually delivered.
and sites do not crash. Message propagation delay is finite but unpredictable. Messages between any two processes are assumed to arrive in the order they are sent. There is no global clock. Byzantine failures [LSP-82] do not occur and failures are \textit{fail-stop} [SS-83].

2.3. Message Ordering

The order of events occurring in different processes can be very important in certain distributed applications. But clocks in different computers may count time at different rates and may thus diverge. Although many synchronization techniques exist, clocks across a distributed system cannot be perfectly synchronized. Therefore, we cannot, in general, use physical time to find out the order of various events occurring in a distributed system. To overcome this problem, Lamport introduced the concept of logical clocks and used it to generate timestamps. A \textbf{timestamp} is a non-negative integer, which accompanies every message. Since almost every mutual exclusion algorithm has used timestamps to order messages, to decide priorities and to resolve conflict between processes seeking concurrent access to CS, we find it appropriate to put here in detail, the concept of timestamps as given by Lamport [L-78]. The timestamps evolve as follows:

Each process $P_i$ maintains a logical clock $C_i$. A logical clock $C_i$ for a process $P_i$ is a function which assigns a number $C_i(a)$ to any event $a$ occurring in the process $P_i$. The entire clock system is represented by $C$. 
For any event $b$ occurring in the system $C(b) = C_j(b)$ if $b$ is an event in the process $P_j$. For any events $a$ and $b$, the condition that if $a$ occurs before the occurrence of $b$, then $C(a) < C(b)$, is called the **clock condition**. The entire system of clocks satisfies the clock condition and to ensure this, the following procedure is adopted: Each process $P_i$ increments $C_i$ between any two successive events, i.e., if an event $a$ occurs before an event $b$ in $P_i$, then $C_i(a) < C_i(b)$. Every message $m$ generated from the process $P_i$ contains a time-stamp $T_m$. A time-stamp, $T_m$, is a non-negative integer whose value equals $C_i(a)$, where event $a$ is the sending of message $m$ by the process $P_i$. On receiving a message $m$, a process $P_j$ sets its clock $C_j$ to a value, which is greater than or equal to the maximum of its present value and $T_m$, the timestamp on the message received.

This induces a partial ordering on the set of all events occurring in the system. Events are ordered using logical clocks as per the procedure described above. Ties are broken using process identities. That is, an event $a$ is given precedence over an event $b$ if and only if $C_i(a) < C_j(b)$ or if $C_i(a) = C_j(b)$ and $P_i < P_j$.

In the section that follows, we discuss vote assignments and coteries, two important tools that are generally used to achieve mutual exclusion in distributed systems and some related concepts.
2.4. Techniques for Mutual Exclusion in PBDME Algorithms

Mutual exclusion in permission-based DME algorithms can be achieved either by using voting or through coteries. Thomas [T-79] was the first to introduce the concept of voting. A non-negative integer \( v_i \) is attached to each node \( i \) of the set \( U = \{1, 2, ..., N\} \). The collection \( \zeta = \{v_1, v_2, ..., v_N\} \) is called a vote assignment for the system. The simplest way to assign votes is to assign one vote each to a process. This is called uniform voting. But different votes can be assigned to different processes. This is known as weighted voting and was introduced by Gifford [G-79]. If all the votes are assigned to a single node, the assignment is said to be singleton. For a vote assignment \( \zeta \), \( \text{Tot}\zeta = \sum_{i=1}^{N} v_i \) gives the total number of votes in the system. The number of majority votes in the system is given by

\[
\text{Maj}\zeta = \begin{cases} 
\frac{\text{Tot}\zeta}{2} + 1, & \text{if } \text{Tot}\zeta \text{ is even} \\
\frac{\sqrt{2}(\text{Tot}\zeta + 1)}, & \text{if } \text{Tot}\zeta \text{ is odd}
\end{cases}
\]

A process can enter its critical section only if it has obtained permission from a number of nodes whose total number of votes is at least \( \text{Maj}\zeta \). Since a node grants permission to only one node at a time, only one node can gather a majority of votes at a time and mutual exclusion is ensured.

In [GB-85], Garcia-Molina and Barbara introduced the concept of a coterie as a powerful tool for enforcing mutual exclusion in distributed systems. A
coterie $C$, on a non-empty set $U$, is a collection of non-empty subsets of $U$ satisfying:

i) Minimality condition: $S, T \in C \Rightarrow S \not\subset T, T \not\subset C$

ii) Intersection property: $S \cap T \neq \emptyset \ \forall \ S, T \in C$

Elements of a coterie are called quorum-groups or quorum-sets or just quorums. If a node wants to perform a restricted operation, it must obtain permission from each and every node of some quorum in the assigned coterie. Since a node grants permission to only one node at a time and since any two quorums in a coterie have at least one node in common, mutual exclusion is guaranteed. A number of techniques have been proposed to construct coteries ([AA-91], [AA-92], [AEA-97a], [BDS-89], [BSW-79], [BS-96], [CAA-92]) and to obtain new coteries from the ones already known to us [NM-92].

A coterie $C$ on a set $U$ is said to be dominated by a coterie $D (\neq C)$ on $U$ if every quorum of $C$ has a quorum of $D$ as its subset, i.e.,

$$\forall G \in C, \exists H \in D : H \subseteq G$$

A coterie $C$ is non-dominated (ND) if there does not exist any coterie that dominates it. Suppose that a coterie $D$ dominates a coterie $C$. Then, if a quorum-group of $C$ survives in a partition due to node or link failures, a quorum-group in $D$ will certainly survive. Thus, we can say that an ND coterie provides more protection against partitions as compared to the coteries it dominates.

Thus, in short, we can say that, in permission-based DME algorithms, a node that wishes to access CS, seeks permission from other nodes of the system. Mutual exclusion is enforced either using voting schemes or through coteries.
Based on the technique used, permission-based algorithms can be divided into two categories:

- Voting-based algorithms
- Coterie-based algorithms

While comparing the two strategies of achieving mutual exclusion Garcia-Molina and Barbara [GB-85] observed that the two approaches are not equivalent. Whereas every vote-assignment gives rise to a coterie, there are coteries that cannot be obtained from vote-assignments. A coterie, which has a corresponding vote-assignment, is called a vote-assignable coterie. The authors prove that for systems with five or fewer nodes, the two concepts turn out to be equivalent but for systems with more than five nodes, there exist coteries, which are not vote-assignable. Thus, coteries are more general than vote-assignments as they provide us with choices, which would not have been obtained using vote-assignments (the number of distinct vote assignments is not greater than $2^n$, while the number of distinct coteries is greater than $2^{2^m}$ for some constant $c$ [GB-85].

Though at the outset, these two approaches for obtaining mutual exclusion in distributed systems may seem to be similar, they are different and the difference between the two is quite important. In algorithms based on coteries, in order to perform a restricted operation, a node must seek permission from a group of nodes, which form a quorum-group of a pre-assigned coterie. Permission from any arbitrary set of nodes will not do. On the other hand, in algorithms based on voting, a node is not concerned about which nodes contribute to its required number of votes. Its only concern is the number of votes it has to collect. From
which nodes do these votes come is immaterial. It sends requests and as soon as it gets permission from nodes whose total votes constitute majority, it enters CS. Thus, though (as observed above) coteries are more general than vote-assignments, vote-assignments are more flexible as compared to coteries. This is so because in algorithms based on vote-assignments, a node is free to obtain permission from any set of nodes, which fulfill its quorum requirement in contrast to algorithms based on coteries, where nodes have to seek permission from particular combinations of nodes – the nodes which form a quorum in the coterie. Therefore, the difference between the two approaches is quite significant and an attempt to classify permission-based algorithms into voting-based algorithms and coterie-based algorithms is justified.

In this section, we have restricted ourselves to the discussion of some very basic concepts, which are crucial for developing a fair understanding of the subject. A number of concepts may still be required to make our discussion meaningful and understandable. We shall define them as and when required. We now proceed to discuss various PBDME algorithms in the next two sections. Section 2.5 has been devoted exclusively to voting-based algorithms and Section 2.6, to coterie-based algorithms.

2.5. Voting-based Algorithms

Voting-based algorithms use majority voting to achieve mutual exclusion. A process seeking entry into CS sends messages and on receiving replies (permissions) from various nodes, sums up the number of votes of the nodes
granting permission and its own votes. As soon as the number of votes gathered becomes more than or equal to \( Maj_\zeta \), the process can enter its CS.

Voting can be static or dynamic. In **static voting**, the nodes do not keep information about the current state of the system and the votes once assigned, are not changed as the algorithm evolves. In **dynamic voting**, the nodes keep track of the current state of the system and in the case of network partitioning, due to link or node failures, new votes may be assigned so as to make at least one partitioned group of nodes active and to keep the system working. We first take up static algorithms.

### 2.5.1. Static Algorithms

The earliest voting-based distributed mutual exclusion algorithms were given by Thomas [T-79] and Gifford [G-79]. These algorithms were static in nature. The votes were fixed a priori. While Thomas used uniform voting, Gifford used weighted voting. In uniform voting, each node is assigned a single vote (or the same number of votes) and a node, which obtains permission from a majority of nodes, is allowed to enter CS. The technique used by Thomas is known as the **majority voting**. In weighted voting, different votes may be assigned to different nodes. Gifford used weighted voting for managing replicated data. A process is permitted to perform a write (read) operation only if it has collected a pre-specified number of votes called write (read) quorum. For a vote-assignment \( \zeta \), the read quorum \( r \) and the write quorum \( w \) are chosen so as to satisfy \( r + w > Tot_\zeta \) and \( 2w > Tot_\zeta \). This ensures mutual exclusion as simultaneous occurrence of two write operations or a read and a write operation is ruled out. The technique used
by Gifford is called the **quorum consensus method**. In fact, the majority voting of Thomas is a special case of the quorum consensus method of Gifford.

Agrawal and Abbadi [AA-92] impose a logical structure on the set of copies of an object and develop a protocol that uses the information available in the logical structure to reduce the communication requirements for read and write operations and call it **Tree Quorum Protocol**. It is a generalization of static voting protocol with two degrees of freedom for choosing quorum. Instead of requiring each write operation to write copies at all levels of the tree, it is required to write only a quorum $w$ of levels. As a result, read operations are required to access a quorum of levels so that there is an intersection of a level between a read and a write operation. Also, instead of requiring each operation to access a majority of children, operations are required to access a quorum of children such that read and write operations have an intersection of at least one child. These generalizations result in quorums with two degrees of freedom: the number of levels and the number of children.

Akhil Kumar in [A-91] observed that a major problem with the quorum consensus method of Gifford [G-79] for synchronizing access by read and write operations to copies of a replicated object, is that the number of copies required for a quorum increases linearly with the total number of copies present of the object as the minimum quorum size is $\left\lfloor \frac{n+1}{2} \right\rfloor$. He proposed that the copies of an object in a database be organized logically into a multilevel tree of depth $m$ with root at level 0. The physical nodes be identified by the leaves of this tree and higher level nodes of the tree shall correspond to logical groups. Thus, a node at
level $i$, $i = 0, 1, 2, ..., m - 1$ is viewed as a logical group which, in turn, consists of $l_{i+1}$ groups at level $i+1$. The quorum is defined by collecting a majority of votes at each level. This reduces the quorum size required for synchronizing operations on $N$ copies to $N^{0.63}$. The hierarchical quorum consensus algorithm (HQC algorithm) turned out to be more general than ordinary voting, as there are quorums, based on this algorithm, which do not have an equivalent single-level representation.

In [AAC-91], the concept of multidimensional voting was introduced. In multidimensional voting, the vote-assignment to each node and the quorum are $k$-dimensional vectors of non-negative integers. For a distributed system of $N$ nodes, numbered $1, 2, ..., N$ the multidimensional vote-assignment $V_{N,k}$ is a $N \times k$ matrix, where $v_{ij}$ represents the vote assigned to the node $i$ in the $j^{th}$ dimension. Clearly, $v_{ij} \geq 0$ for $i = 1, 2, ..., N$ and $j = 1, 2, ..., k$. The votes assigned in various dimensions are independent of each other. The quorum-assignment $\bar{q}_k = (q_1, q_2, ..., q_k)$ is a $k$-dimensional integer vector where $q_j > 0$ for $j = 1, 2, ..., k$.

In addition, a number $l$, $1 \leq l \leq k$, is defined. It is the number of dimensions of vote-assignment for which the quorum must be satisfied. Thus, there are two levels of requirement: the vote level and the dimension level. At the vote level, the number of votes must be greater than or equal to the quorum requirement in that direction and at the dimension level, the number of dimensions for which a quorum is collected must be greater than or equal to $l$. Multidimensional voting with quorum requirement in $l$ out of $k$ dimensions is denoted by MD$(l,k)$-voting. In this paper, Ahmad et al. show that though a coterie may not be vote-assignable
in general, every coterie can be represented by a MD($l,k$) vote and quorum assignment. To support their result by examples, they obtain a MD(1,5) vote and quorum assignment for tree based coterie of Agrawal-Abbadi [AA-91] for a tree of depth three with 7 nodes and give an example of a coterie for $N = 6$ which is not vote-assignable ordinarily but has a corresponding multidimensional vote-assignment MD(1,4). The authors show that multidimensional voting can also be applied for reading and writing replicated data and to partially replicated data and obtain a multidimensional vote and quorum assignment for the 3x2 grid protocol of Cheung et al. [CAA-92]. They also introduce the concept of nested multidimensional voting and apply it to obtain a generalization of the Hierarchical Quorum Consensus method given by Akhil Kumar in [A-91].

2.5.2. Dynamic Algorithms

Dynamic algorithms use the knowledge of the current state of the system to choose new vote-assignments. Though, voting is resilient to partitions in the sense that the system keeps on working despite partitioning as long as there is at least one group whose nodes have a majority of votes, there may arise situations when none of the partitioned groups can collect the required number of votes and the system may enter a halted state. Halted states are undesirable as they reduce system availability. Dynamic algorithms help in improving performance by reducing the incidence of halted states.

In [BGS-89], Barbara et al. suggest two methods of dynamic vote reassignment: the group consensus method and the autonomous reassignment method. In the group consensus method, the nodes in the majority partitioned-
group agree upon the new vote-assignment, using either a distributed algorithm or by electing a coordinator to perform the task. In the **autonomous reassignment method**, each node makes decision about changing its votes and picks up a new vote value on its own. But before the change is made final, the node must collect a majority of votes. While reassignment by the group consensus method is more accurate and more resilient to future failures, its implementation is complicated. On the other hand, the autonomous reassignment method is quicker, simpler and more flexible although the global vote-assignment achieved by this method may not be as good as the one achieved through group consensus [BGS-89].

Jajodia and Mutchler in [JM-90] propose two generalizations to voting schemes: dynamic voting and dynamic voting with linear ordered copies (dynamic-linear voting, in short). In **dynamic voting**, the number of sites necessary for carrying out an update is a function of the number of up-to-date copies in existence at the time of the update and not of the total number of copies, as in the case of static algorithms. Changes to the quorum occur dynamically, without any manual intervention. Each copy of the file maintains two variables: a version number, \(VN\), and an update sites cardinality, \(SC\), which is the number of sites that participated in the most recent update to the file. When a site wants to commit an update, it sends messages to all the sites in the partition to which it belongs. On receiving replies, it learns about the \(VN\) (the most recent version number of the copies available in the partition) and \(SC\), the number of copies in the partition with that version number. It commits an update if the partition, to which it belongs, is the distinguished partition. A partition \(P\) is the **distinguished** partition if and only if it contains more than half of \(SC\) sites with version number
**Dynamic-linear voting** extends dynamic voting by adding a third variable, the distinguished site, $DS$, which is used to break the tie when a partition contains exactly half of the sites with the up-to-date version of the file. When the number of sites participating in an update is even, each site sets its $DS$ entry to one of the participating sites. If, in a partition, the number of sites with the most recent $VN$ is exactly $SC/2$, an update is permitted if and only if the site contains the distinguished site $DS$. In such a case, an update would not have been permitted in dynamic voting. Moreover, dynamic-linear voting accepts an update whenever dynamic voting does.

### 2.5.3. Availability Analysis for Vote-assignments

Availability is a very important performance measure for distributed mutual exclusion algorithms and needs special attention. Due to node or communication link failures, a network of computers may partition into disjoint partitions. In the case of voting-based DME algorithms, a node seeking entry to its CS has to collect a majority of votes. But due to network partitioning, it may so happen that no partition is available so that the required quorum of votes can be collected. Availability of a vote-assignment tells us about the possibility of finding an active partition (with total votes more than or equal to the required quorum). In a highly available vote-assignment, the probability of finding such a partition, and thus the probability of a restricted operation being performed, is very high. Therefore, the relative performance of various vote-assignments can be compared on the basis of their availability. A lot of work has been done to find the availability of various
vote-assignments and to find vote-assignments optimal with respect to availability.

In [BG-87], Barbara and Garcia-Molina study the impact of the network topology and the number of nodes on the availability of a system. They address the problem of selecting vote-assignments in order to maximize the probability that the critical operation can be performed at a given time by some group of nodes. Heuristics to assign votes, so that the probability of having an active group is maximized, are suggested in this work. These heuristics take into consideration the reliabilities of nodes and links and the network topology and give good results in most cases. The authors use a probabilistic metric to study vote-assignments and derive analytical expressions for system reliability in the case of three homogeneous topologies - fully connected, Ethernet and ring network and find that, in the case of these three topologies, for different link and node reliabilities, different vote-assignments (uniform, non-uniform or singleton) are viable.

Also in [BG-86], the same authors investigate the influence of vote-assignments on the node and edge vulnerability and show that the vulnerability of a distributed system can be improved by the use of good assignments. The node vulnerability of a system is the minimum number of crashed nodes that produce a halted state and its edge vulnerability is the minimum number of link failures that produce a halted state. The highest node vulnerability for a system with \( N \) nodes is achieved by a \((N+1)/2\)-uniform vote-assignment if \( N \) is odd, and by a \( \frac{N}{2} \)-pseudo-uniform vote-assignment if \( N \) is even. In the case of edge vulnerability, the best vote-assignment cannot be determined without referring explicitly to a
topology. For a ring of \( N \) nodes \( (N > 2) \) and any non-dominated coterie (vote-assignment) other than the singleton coterie assigned to it, its edge vulnerability is 3. For a completely connected graph of \( N \) nodes, the best coterie to assign in terms of edge vulnerability is a \([\frac{N}{2}]\)-uniform or pseudo-uniform coterie \((N > 2)\). The authors also suggest several heuristics for assigning votes, which produce vote-assignments with reasonable edge and node vulnerability.

Jajodia and Mutchler [JM-90] compare the availabilities of various voting schemes under two models: the site model and the link model. In the site model, only sites fail and links are infallible. Any site that has not failed can communicate with every other site that is up. In the link model, the network is modeled as a connected graph. Only the links are subject to failures and repairs, the sites do not fail. Under the site model, the availability of the dynamic-linear algorithm turns out to be greater than that of the dynamic voting and the static voting algorithms, when the number of sites in the model is more than three. Under the link model also, the availability of the dynamic-linear voting turns out to be larger than the best static algorithm. Thus, the authors deduce that the dynamic-linear voting has greater availability than any static algorithm. Moreover, it is as easy to implement as static voting and its message complexity is slightly more than any static algorithm.

In [CAA-89], Cheung et al. address the problem of optimizing performance for reading and writing replicated data and provide a method to enumerate vote and quorum assignments in the general read/write case when the read and the write quorums may be different. They give an efficient method for
finding the availability for any vote and quorum assignment when the availability of the nodes and the read/write transactions are given and use system availability to illustrate how the set of vote and quorum assignments can be used to find the optimal assignment for reading and writing replicated data.

Spasojevic and Berman [SB-94] prove that though coteries are more general than vote assignments, voting is an optimal static pessimistic scheme for fully connected networks with negligible link failures and for Ethernet systems. But, it may not be optimal for general systems. They actually present a network for which no voting scheme yields optimal availability. The authors also propose an efficient algorithm for computing optimal vote assignment for fully connected networks when relative frequencies of read and write operations are known.

Thus, we have seen that vote-assignments are quite useful in enforcing mutual exclusion in distributed systems. A number of voting-based algorithms are available and one may choose among these, the one that best suits the system requirement. Voting-based algorithms are easy to implement and provide high degree of flexibility. Availability can be taken as a criterion to compare various voting-based algorithms. Other things being equal, a vote-assignment with higher availability shall be chosen. One may choose between static and dynamic algorithms. Dynamic algorithms provide higher availability by adapting to changing system conditions. System availability in the case of dynamic vote-assignments is, certainly, better than that in the case of static vote-assignments and of the two dynamic vote reassignment methods given by Barbara et al. in [BGS-89], the autonomous reassignment method is a viable alternative to the group
consensus method since it yields almost as much availability as the group consensus approach and is much faster [BGS-89]. Prior knowledge of the reliabilities of nodes in a system can be helpful in making a decision about the selection of the algorithm to be implemented. For example, of the various static algorithms available, the HQC algorithm has better availability than that of the majority voting for \( p < 0.5 \) (where \( p \) is the probability that a node of the system is up) and for \( p > 0.8 \), the majority voting algorithm does better than the HQC algorithm [A-91]. Network topology may also play an important role in selecting an algorithm. For example, Barbara and Garcia-Molina in [BG-87] show that for the homogeneous, fully connected systems with perfect links and for Ethernet systems, if \( p = 0.5 \) then the uniform assignment is better than any non-uniform assignment whereas for the ring network, there may be non-uniform assignments that are superior to the uniform assignment even if \( p > 0.5 \). They observe that distribution of votes may not be advisable (i.e. assigning all the votes to a single node is preferable) in the ring networks or other networks with low connectivity but may prove to be advantageous in improving system availability if a few reliable links are added to these networks. Even while dealing with replicated databases, high availability is desirable because an algorithm with a high availability is more likely to satisfy a request for an update. The number of copies replicated may be important. For example, for large number of copies, the Tree Quorum Protocol provides higher availability of operations at substantially lower rates [AA-92].
2.6. Coterie-based Algorithms

In coterie-based algorithms, a node trying to enter CS has to seek permission from a group of nodes and for mutual exclusion to hold such a group is picked from a coterie. A number of coterie-based algorithms have been proposed. Different algorithms have used different methods to construct quorum-groups, i.e., their coterie construction techniques are different. As a result, they show different performances. These algorithms differ from each other on various counts but their general principle is the same.

The algorithm of Ricart and Agrawala [RA-81] is one of the earliest known DME algorithms and needs special mention. In this algorithm, a node $i$ seeking entry to CS would send requests to all the remaining nodes. On receiving a request, a node $j$ sends a reply immediately if it is not interested in CS. If the node $j$ is itself interested in CS, it compares the priority of the incoming request with its own priority and the process with higher priority gets the right to enter its CS. The process granting permission to a requesting process looks for the conflict of the incoming request with itself only. By sending a reply message to a node $i$, the node $j$ does not lock itself for the node $i$; it may send reply messages to other nodes as well. It is based on self-conflict while most of the algorithms developed later look for the conflict among various requesting sites. This puts Ricart-Agrawala’s algorithm in a class apart from other algorithms. The algorithm is fully distributed, symmetric and static in nature. It has message complexity $2(N+1)$ and synchronization delay, $D = 2$. This is an improvement over
Lamport's algorithm [L-78] which is perhaps the first DME algorithm in message passing models and has message complexity $3(N+1)$.

Carvalho et al. [CR-83] observed that the message complexity of the Ricart-Agrawala's algorithm could be improved by a slight modification. A node seeking entry into CS for the first time sends request messages to all the remaining nodes as in the case of Ricart-Agrawala's algorithm. But subsequently, it sends request messages only to those nodes to which it had sent back its permission (owing to a request message from them). The permission of those nodes from which it had received a reply message, and has not sent yet back its permission, is taken for granted. This algorithm is dynamic in nature and its message complexity varies from 0 to $2(N+1)$.

These two algorithms in which nodes use 'self-conflict only' criteria for granting permission to other nodes, are closer to the token-based algorithms. The only difference being that, while in the token-based algorithms requests are queued along with the token, in the case of these algorithms, requests are distributed. We now go through various coterie-based algorithms in which nodes look beyond self-conflict while granting permission to other nodes. As in the case of voting-based algorithms, coterie-based algorithms can also be static or dynamic, depending on the fact whether the nodes keep track of the current state of the system or not.
2.6.1. Static Algorithms

The first ever coterie-based algorithm was given by Maekawa [M-85]. It can be regarded as a stepping-stone in the development of DME algorithms for a number of reasons:

- For the first time, it was assumed that the nodes are logically arranged.

- Unlike Ricart-Agrawala’s algorithm, Maekawa’s algorithm goes beyond self-conflict. On receiving a request message, a node would send its permission to the requesting node only if it has not already granted permission to some other node. It grants permission to only one node at a time, locks itself exclusively for that node and only after receiving a RELEASE message from the node, it grants permission to another node: the node with the highest priority among the requesting nodes.

- The logical grid used by Maekawa and its modifications have been frequently used in a number of algorithms developed later.

The quorums in Maekawa's algorithm are obtained as follows: Each node \( i \) obtains permission from a set of nodes \( S_i \). For mutual exclusion to hold, the condition

\[
a) \ S_i \cap S_j \neq \emptyset \ \forall \ i, j, 1 \leq i, j \leq N
\]

is a must. In addition, Maekawa assumed that

\[
b) \ i \in S_i, \ \forall \ i \in \{1, 2, ..., N\}
\]
c) \( |S_i| = k, \ \forall i \in \{1, 2, \ldots, N\} \)

d) For every \( j, \ j \in S_i \), for exactly \( m \) values of \( i \).

With these assumptions, the algorithm becomes truly **fully distributed**. Every node has to do the same amount of work (each node has to send and receive the same number of messages for achieving mutual exclusion due to the condition c)) and bears equal responsibility (each node is an arbitrator for exactly \( m \) nodes, as a result of the condition d)). Using results from projective geometry, Maekawa showed that the existence of quorums \( S_i \) satisfying a) to d) is guaranteed and the size of each \( S_i \) is equal to \( \sqrt{N} \). He also gave a simpler method to obtain quorums \( S_i \). If the set of nodes is logically arranged in the form of a grid, \( S_i \), the quorum-group for the node \( i \), is the set of all the nodes contained in the column and the row through the node \( i \). This gives the size of each \( S_i \) as \( 2\sqrt{N} - 1 \). The algorithm has message complexity \( c\sqrt{N} \) where \( c \) varies from 3 to 5 and synchronization delay \( D = 2 \). Ricart-Agrawala's and Thomas's algorithms have the same delay.

Though claimed to be deadlock free, Maekawa's algorithm turned out to be deadlock prone [S-91]. Mukesh Singhal in [S-91] observed that the reason for deadlock in Maekawa's algorithm is that in this algorithm, a site granting permission to another site locks itself for that site in an exclusive mode. He suggested that the locks on sites must be made mutable to avoid deadlocks. It is quite possible that a higher priority request arrives at a site after the site has been locked by a lower priority request. The higher priority request converts the exclusive lock into a shared lock. An INQUIRE message is sent to the lower
priority request, which preempts the lock if it has not yet been able to lock all the sites in its request set: the set of nodes to which it is supposed to send request messages. A request succeeds in locking a site, if its priority is higher than the priority of all the requests, which currently hold the lock on it. This makes the algorithm deadlock free but increases the message complexity from $o(\sqrt{N})$ to $o(N)$. The synchronization delay is equal to 1. A technique for the formation of optimal request sets is also discussed in this paper.

Agrawal-Abbadi [AA-91] gave another logical structure based algorithm called the Tree Quorum Algorithm. The sites are logically arranged into a tree with a well-defined root. A quorum is constructed by selecting any path starting from the root and ending with any of the leaves. If a path is not available as a result of the failure of a node $i$, then $i$ is substituted with two paths ($m$ paths, when the algorithm is extended to a tree in which every node has degree $m$), both (all) starting with the children of $i$ and terminating with leaves. Quorums (quorum-groups) obtained using this algorithm are called tree quorums. The collection of all the tree quorums gives the coterie generated by this algorithm. In a failure free environment, the message complexity of tree quorums algorithm is $\lceil \log N \rceil$ which is an improvement over Maekawa's algorithm ($MC = \sqrt{N}$). The algorithm exhibits the property of graceful degradation: in a failure-free environment, message complexity is low; as the number of node-failures increases, the message complexity goes on increasing. The penalty for failures closer to the root is more severe than the failures in the vicinity of the leaves. This is a quorum-construction
algorithm and other measures need to be taken to avoid deadlock and starvation [AA-91].

Another advantage of Agrawal-Abbadi's algorithm over Maekawa's algorithm is that, while in Maekawa's algorithm, each node has only one fixed request set attached to it, a node in the Agrawal-Abbadi's algorithm may choose its request set from amongst a number of quorums in the coterie. The failure of a single site in its request set, $R_i$, may render node $i$ unable to form a quorum-group in the case of Maekawa's algorithm; the tree quorum algorithm always guarantees the formation of a quorum-group as long as the number of failures is less than $\lceil \log N \rceil$ sites.

Since all coterie-based algorithms require that a node seeks permission from some nodes before entering CS and informs some set of nodes after completing the execution of CS, processes need to keep information about other processes. Sanders [S-87] observed this and introduced the concept of information structure as a unifying principle behind various DME algorithms. The information structure of a DME algorithm consists of $R_i$, the request set for the process $i$ and $I_i$, the inform set for the process $i$. The request set for the process $i$ consists of those nodes of the system from which it is supposed to receive permission before it enters the critical section whereas its inform set consists of those nodes of the system, to which it must inform whenever it changes its state from in-CS to not-in-CS or vice versa. Also, $S_i$ denotes the status set that is maintained by the node $i$ and is the set of those nodes about which the node $i$
maintains information. Obviously, $i \in I_j \Rightarrow j \in S_l$. Sanders gave a generalized DME algorithm (GEN ISDME algorithm) based on information structure and showed that for a DME algorithm to be correct, its information structure must satisfy the conditions:

a) $i \in R_i \ \forall i \in U$

b) $I_i \subset R_i \ \forall i \in U$

c) $\forall i, j \in U, i \neq j, \{I_i \cap I_j \neq \emptyset\} \ OR \ \{i \in R_j \land j \in R_i\}$

(2.6.1.1)

where $U$ is the set of all nodes in the system. The performance of a DME algorithm can be studied as a function of its information structure. Moreover, the information structure of an algorithm can be static or dynamic.

Although, in principle, the GEN ISDME algorithm is supposed to resolve all conflicts, there may arise situations when it is deadlocked. This may be caused by unpredictable communication delays. To overcome this difficulty, Sanders proposed a deadlock detecting general information structure based distributed mutual exclusion algorithm (DDGEN ISDME algorithm), which detects and avoids potential deadlocks using a deadlock recovery scheme, similar to the one used by Maekawa [M-85]. The modified algorithm has the same necessary and sufficient conditions on the information structures in order to ensure mutual exclusion.

Bonollo and Sonenberg [BS-96] showed that by restricting the class of information structure used by Sanders, a deadlock free ISDME algorithm
(DFGEN ISDME) can be obtained which does not require any deadlock recovery scheme. The information structure conditions that must hold are

\begin{align}
\text{a)} \quad & i \in I, \ \forall i \in U \\
\text{b)} \quad & I_i \subset R_i, \ \forall i \in U \\
\text{c)} \quad & (i \in I_j \lor j \in I_i) \lor \{i \in R_j \land j \in R_i\} \ \forall i, j \in U, i \neq j
\end{align}

First part of condition c) requires that either \(i\) or \(j\) is in \(I_i \cap I_j\) whereas in the case of Sanders' theorem, it was sufficient to have some \(k \in I_i \cap I_j\). The class of information structure allowed by these conditions is a subset of the class of information structure allowed by Sanders' ISDME theorem. The algorithm has \(D = 1\) and \(MC = \frac{(2a + 3b)}{N}\). In the best case, the smallest value of MC is \(3(N - 1)/2\) which is the same as that of Singhal's algorithm [S-91] and in the worst case, the largest value of MC is \(2(N - 1)\) which is the same as that of Ricart-Agrawala's algorithm [RA-81].

Before we proceed further, let us take a look at some of the algorithms mentioned above, their information structures and their inter-relationship. Both Maekawa's algorithm and Sander's ISDME algorithm were deadlock prone and needed deadlock recovery schemes. Mukesh Singhal provided a deadlock free Maekawa-type algorithm (an algorithm in which request sets satisfy \(i \in R_i, \ \forall i \in U\) and \(R_i \cap R_j \neq \emptyset, \ \forall i, j \in U\)), which did not require a deadlock recovery scheme. Using almost similar technique, Bonollo and Sonenberg gave DEADLOCK FREE ISDME algorithm, which did not require a deadlock recovery scheme. In terms of information structures, conditions for Maekawa's algorithm are
\[ R_i \cap R_j \neq \emptyset, \forall i, j \in U \] (2.6.1.3)

and for Singhal's deadlock free Maekawa-type algorithm are

\[ i \in R_j \lor j \in R_i, \forall i, j \in U, i \neq j \] (2.6.1.4)

The constraint (2.6.1.4) imposed by deadlock free Maekawa-type algorithm is stronger than the constraint (2.6.1.3) required for Maekawa's algorithm and is weaker than what is required in Ricart-Agrawala's algorithm, Viz.,

\[ i \in R_j \land j \in R_i, \forall i, j \in U, i \neq j \] (2.6.1.5)

We have already seen that the class of information structures (2.6.1.2), allowed by the DFGEN ISDME algorithm of Bonollo and Sonenberg, is a subset of the class of information structures (2.6.1.1) allowed by the DDGEN ISDME algorithm of Sanders. Moreover, the class of information structures (2.6.1.2) is a superset of the class of information structures (2.6.1.4) admitted by the deadlock-free Maekawa-type algorithm of Singhal. Thus, the DFGEN ISDME algorithm of Bonollo and Sonenberg may be regarded as representing a continuum from Singhal's deadlock free Maekawa-type algorithm on one hand and to Ricart-Agrawala's algorithm on the other [RA-81].

Maekawa's logical arrangement of nodes into a grid came out to be quite handy for constructing quorums (and thus coteries) and a number of quorum construction techniques have been proposed based on this. Cheung et al. in [CAA-92] present the grid protocol for synchronizing reads and writes in a replicated database. The nodes storing copies of data are assumed to be logically arranged into a rectangular grid. Read and write operations are required to lock groups of nodes (quorum-groups) in such a way that two write operations or a read and a
write operation cannot be executed simultaneously. This is achieved as follows.
Choosing one node from each column of the grid forms a read quorum-group. Such a group of nodes is called a C-cover (column cover) as it intersects with every column of the grid. A write quorum-group is formed by choosing a C-cover along with all the nodes in any one column of the grid. This ensures that any two write quorum-groups and a write and a read quorum-group always have a node in common. For even distribution of load among the nodes, the choices of read and write quorums are made using random permutations of rows and columns. The protocol provides high data availability and low response time (of course, this protocol and many others that follow were designed for managing replicated data. So, for each of these protocols, both read and write quorums have to be defined and the issues involved may be different. But as far as the write quorums are concerned, mutual exclusion has to be guaranteed. Therefore, in this and any other replica control protocol we mention, we will restrict ourselves only to write quorums). The algorithm dynamically searches for a quorum but all possible quorums are fixed a priori. A number of quorums are available for each node.

In [AC-91], Akhil Kumar et al. combined the ideas in [CAA-92] and [A-91] and organized nodes in a H-grid (hierarchical grid) using a multilevel hierarchy with physical nodes at level 0 and logical groups at higher levels. The quorums can be obtained by collecting all objects in any one column of the grid at the level $L$. At the level $i$, this procedure translates into collecting level 0 objects, which are physical nodes. The set of nodes so collected gives a quorum of size $o(\sqrt{N})$. 
Agrawal et al. [AEA-97a] used a modified grid to construct what they called ‘Billiard Quorums’. Instead of using horizontal and vertical line segments of rows and columns in the grid scheme of Maekawa, they used paths that resemble billiard paths. This reduced quorum size to $\sqrt{2N}$. However, the size of each quorum has to be an odd integer and the method does not satisfy the equal responsibility property of Maekawa.

Luk and Wong [LW-97] suggest that if, instead of using a square grid, we organize the nodes logically into a triangular grid, then any two quorums will meet only in one node. They constructed quorums using this triangular grid and the quorum size turned out to be approximately equal to $\sqrt{2N}$ which is better than that of Maekawa and almost as good as that of Agrawal et al. [AEA-97a]. The quorums can be row-based or column-based and if the two schemes are used alternately for each new request, the equal responsibility property is satisfied and each requesting site can have two quorums.

Rangarajan et al. [RST-95] divided $N$ sites into $N/G$ groups of $G$ sites each and called each such group, a subgroup. They constructed $N/G$ quorum-groups such that each quorum-group is made up of $\sqrt{N/G}$ subgroups with each subgroup containing $G$ sites. The quorum-groups are constructed using finite projective planes just as was done in Maekawa’s algorithm. The only difference being that now, instead of sites, subgroups are used to construct quorum-groups. There is only one quorum-group per site. In fact, each quorum-group caters to $G$ sites. The intersection between any pair of quorum-groups is exactly one subgroup. The algorithm has a message complexity $O(\sqrt{N\log N})$ and achieves
asymptotically high resiliency to failures for a much lower message overhead than the majority voting.

Cao et al. [CSDRS-98] suggest an algorithm, which reduces synchronization delay and still has low message complexity. The basic idea is as follows: A site $i$ can enter its CS only if it has got permission from all the sites in its quorum-group. But, instead of first sending a RELEASE message to unlock the arbiter site, which in turn sends a REPLY message to the next site to enter the CS, a site exiting the CS directly sends a REPLY message to the site which will next enter CS. This reduces synchronization delay from 2 to 1. This is a coterie-based algorithm but is independent of the coterie being used, i.e., we can use any coterie we like. The message complexity is $o(ck)$, where $c$ is a constant lying between 3 and 6 and $k$ is the average size of the quorum. For example, $k$ is $\sqrt{N}$ if we use quorums formed by Mackawa’s algorithm and it is $\log N$ if we use those formed by Agrawal-Abbadi’s tree quorum algorithm. Use of the logical structure depends on the coterie formation scheme being used. Moreover, the resiliency to site and communication link failures can be increased if a fault-tolerant coterie is used.

Neilson and Mizuno [NM-92] gave a method of producing new coteries from known coteries with the help of a function, which they called the Coterie Join Algorithm. The method is as follows:

For any set of nodes $U_1, U_2$ with $U_1 \cap U_2 = \emptyset$ and for $x \in U_1$, define $U_3 = U_1 - \{x\} \cup U_2$. For any coterie $C_1$ on $U_1$ and $C_2$ on $U_2$, we can obtain a coterie $C_3$ on $U_3$ by retaining all quorums of $C_1$ not containing $x$ and by
replacing each occurrence of $x$ in a quorum of $C_1$ by a quorum of $C_2$. They proved that the new coterie $C_3$ thus obtained is non-dominated whenever both $C_1$ and $C_2$ are non-dominated. They emphasize that since it may be cumbersome to compute and store all the quorums in the composite coterie, what needs to be stored is: i) each of the input coterie and ii) the information about how the composite coterie is to be constructed.

### 2.6.2. Dynamic Algorithms

Most of the distributed mutual exclusion algorithms are static in nature as their information structures do not change during the execution of the algorithm. These algorithms do not take advantage of the dynamic nature of the system. Dynamic ISDME algorithms can adapt to fluctuating system conditions and exploit them to optimize the performance. Mukesh Singhal in [S-92] presents one such algorithm whose information structure evolves with time as sites learn about the system through messages. The algorithm assigns an initial information structure and the rules for changing the information structure are such that the mutual exclusion conditions are always satisfied. It gets rid of the queue (in Maekawa's and other similar algorithms), which stores blocked requests. Instead, the algorithm uses dynamically changing inform sets which lead to more elegant presentation of dynamic ISDME algorithms. Initially, the request set for the node $i$ is set as $R_i = \{1,2,\ldots,i\}$. The cardinality of $R_i$ decreases in stepwise manner from left to right. The rules for information exchange and for updating request sets and inform sets are such that the staircase pattern is preserved in the system even after the
sites have executed CS a number of times. Deadlocks are avoided by assigning a globally unique timestamp to each request, which determines its priority. In this paper, a simulation study is also carried out to compare the algorithm with that of Ricart and Agrawala. The message complexity of the algorithm is obtained as $N-1$ for low to medium traffic and its average message complexity for heavy traffic is $3(N-1)/2$. This improvement over the Ricart-Agrawala’s algorithm is achieved without sacrificing delay. Moreover, the dynamic nature of its information structure does not complicate the algorithm’s recovery from various types of failures.

Rabinovich and Lazowska [RL-92] also observed that the conventional structured coterie protocols, though efficient, incur a penalty of reduced availability in exchange for the performance gain because, in most of the existing protocols, the operations rely on their knowledge of statically pre-defined logical structure of the network. They propose a mechanism that allows dynamic adjustment of quorum-sets when quorums are defined based on a logical network structure. This improves the availability of the replica control protocols. The basic principle is as follows: Given an ordered set of nodes, one can devise a rule that unambiguously imposes a desired logical structure on this set. The read and the write operations rely on this rule for determining quorum-groups and not on the knowledge of the static structure of the network. In this protocol, each node is assigned a name and all the names are linearly ordered. Among all the nodes replicating a data item, a set of nodes is identified as the current epoch and at any time the data item may have only one current epoch associated with it. Originally,
all replicas of the data item form the current epoch and at least a write quorum of nodes from an existing epoch, is required to be included in the new epoch. The authors use this technique to obtain a dynamic grid protocol from the grid protocol of Cheung et al. [CAA-92] and show that the dynamic grid protocol increases the availability manifolds as compared to the static grid protocol while preserving its good load sharing and message traffic characteristics.

2.6.3. Coteries on Graphs

Ibaraki et al. [INK-95] generalized the concept of coteries to apply it to a family of subgraphs of a graph. An arbitrary family $\mathcal{H}$ of connected subgraphs of a graph $G = (V, E)$ is called a $G$-coterie if

\begin{enumerate}
\item $V_H \neq \emptyset$, $\forall H = (V_H, E_H) \in \mathcal{H}$.
\item $H_i, H_j \in \mathcal{H} \Rightarrow H_i \not\subset H_j, H_j \not\subset H_i$ \hspace{1cm} (Minimality Condition)
\item $V_{H_i} \cap V_{H_j} \neq \emptyset \forall H_i = (V_{H_i}, E_{H_i}), H_j = (V_{H_j}, E_{H_j}) \in \mathcal{H}$ \hspace{1cm} (Intersection Property)
\end{enumerate}

Let $G = (V, E)$ be a graph and let $C$ be a coterie on $V$. For $S \subseteq C$, let $H = (V_H, E_H)$ be a connected subgraph of $G$ satisfying $S \subseteq V_H$. $H$ is minimal with respect to $S$ if there is no connected proper subgraph $H_1 = (V_{H_1}, E_{H_1})$ of $H$ satisfying $S \subseteq V_{H_1}$. Let $H_G(S)$ denote the family of all minimal connected subgraphs, $H = (V_H, E_H)$, of $G$. Define

$$H^*_G(S) = \bigcup_{S \subseteq C} H_G(S) - \{ H : \exists H_1 \in \bigcup_{S \subseteq C} H_G(S) \text{ and } H_1 \text{ is a proper subgraph of } H \}$$
For coteries $C$ and $D$ on the node set $V$ of graph $G$, $C$ is said to $G$-dominate $D$ if

$$H'^*_G(C) \neq H'^*_G(D) \text{ and } H'^*_G(C) > H'^*_G(D)$$

where $H'^*_G(C) > H'^*_G(D)$, if for any graph $H \in H'^*_G(D)$, there is a graph $H' \in H'^*_G(C)$ such that $H'$ is a subgraph of $H$. A coterie $C$ is $G$-nondominated if there is no coterie that $G$-dominates it.

### 2.6.4. Availability of Coteries

Availability is an important criterion, which can be used to test the quality of a coterie. The availability of a coterie is the availability of at least one of its quorum-group when required. When a node needs to enter CS, it should be able to find at least one quorum-group, all of whose vertices are operational and connected. It is defined as the probability that the set of operational nodes has a connected subgraph consisting only of fault-free nodes such that some quorum of the coterie is included in the subgraph. Mathematically, if $p_i$ denotes the probability that a node $i$ operates correctly and $r_j$, the probability that an edge $e_j$ operates correctly, where $0 < p_i, r_j \leq 1$, for a graph $G = (V, E)$, then the availability of a coterie $C$ with respect to $G$ is denoted by $A_G(C)$ and is defined as the probability that there is a connected subgraph $G_1 = (V_1, E_1)$ of $G$, consisting only of operational nodes and edges such that $Q \subseteq V_1$ for some $Q \in C$. Higher availability of a coterie exhibits greater ability to tolerate node and link failures. A coterie $C$ on a graph $G$ is optimal with respect to availability if $A_G(C) \leq A_G(C_1)$, for any coterie $C_1$ on the node set of $G$. 
Attempts have been made to find coteries optimal with respect to availability. Ibaraki et al. [INK-95] used the concept of $G$-domination (domination with respect to a graph) of coteries to calculate the availability of coteries on rings and trees and obtained the following results:

- An optimal coterie for a graph $G$ is always an optimal coterie for one of its biconnected component.
- A singleton coterie is optimal if $G$ is a tree.
- A singleton coterie or a 3-majority coterie maximizes availability if $G$ is a ring.

In [DKKMP-94], Diks et al. prove that for arbitrary networks with low node reliability ($p = \frac{1}{2}$), the singleton coterie is always optimal. They also obtain optimal coteries for complete multipartite networks with $p = \frac{1}{2}$ and vote-assignable optimal coteries for $(1,m)$-, $(2,m)$- bipartite networks for node reliability $p = \frac{1}{2}$. A graph $G$ is multipartite if it can be decomposed into a number of disjoint subsets $V_1, V_2, \ldots, V_k$ such that no edge in $G$ joins the vertices in the same subset. If $V_i$ has $m_i$ points, $i = 1, 2, \ldots, k$, it is said to be $(m_1,m_2,\ldots,m_k)$-multipartite. It is said to be bipartite if $k = 2$.

In [PW-95], Peleg and Wool show that if $p$ denotes the probability that a node fails, then for $0 < p < \frac{1}{2}$, the least available ND coterie is the singleton coterie and the most available coterie is the majority coterie whereas if the nodes are fail-prone ($\frac{1}{2} < p < 1$), the best strategy is to pick a single centralized king.
If $p_i$ is different for different nodes and $p_i < \frac{1}{2}$ for every $i$, then voting defines the optimal availability quorum system. If $0 < p_i < \frac{1}{2}$ for every $i$, then the optimal weights are given by $w_i = \log_2(1 - p_i)/p_i$ [SB-94, TK-88]. Amir and Wool [AW-98] complete this line of research when they consider the most general case in which for some or all values of $i$, $p_i \geq \frac{1}{2}$. The authors

- Prove that any node with $p_i \geq \frac{1}{2}$ is dummy and can be assigned weight $w_i = 0$.
- Show that if $p_i \geq \frac{1}{2}$ $\forall i$, then a monarchy is an optimal quorum system with one of the least unreliable processor as the king.
- Present an algorithm for calculating the optimal weights for a given quorum system.

### 2.6.5. Communication Delay: An Important Performance Metric for Coteries on Graphs

Communication delay is another important metric on the basis of which we can test the quality of a coterie. Fu et al. [FWLN-94], Fu [F-97] observed that generally quorum size has been used to estimate the communication cost of a coterie without considering the actual distance between different sites. Not much attention has been paid to minimizing communication delay, which can be an important factor for the system response time. They define two metrics to measure the communication delay (or the communication cost) of coteries, which take into
account the network topology. The metrics are \textit{max\_delay} and \textit{mean\_delay} and are defined as follows:

Let $G = (V, E)$ be a network and let $C$ be a coterie on $G$ (on its node set $V$). Let $\text{dist}(a, b)$ denote the distance between any two nodes $a$ and $b$ in $G$. It is equal to the minimum number of hops necessary for moving from $a$ to $b$ through the edges in $G$. For any $s \in G$ and for any $Q \in C$, we find the distance of $s$ from every point of $Q$ and denote the maximum of these distances by $\alpha(s, Q)$. That is,

$$\alpha(s, Q) = \max\{\text{dist}(s, v) : v \in Q\}$$

We use the symbol $\alpha(s, Q)$ instead of the symbol $\text{dist}(s, Q)$, used by Fu et al. [FWLN-94], because in this work we intend to exploit the metric space structure on graphs and in a metric space $\text{dist}(x, A)$ denotes the distance of a point $x$ from a set $A$, which is obtained by finding the distance of the point $x$ from each and every member of the set $A$ and then taking the infimum (minimum, in case the set is finite) of all these distances. Here, in our work, whenever we say the distance of a point $x$ from a set $A$, we mean $\alpha(x, A)$.

We obtain the value of $\alpha(s, Q)$ for every quorum $Q$ in $C$, take minimum of these values and call it the \textbf{delay of the node $s$ from the coterie $C$}. It is denoted by $\text{delay}(s, C)$ and its value is

$$\text{delay}(s, C) = \min\{\alpha(s, Q) : Q \in C\} = \min_{Q \in C} \max_{v \in Q} \text{dist}(s, v)$$

A quorum $Q$ in $C$ for which $\alpha(s, Q)$ is minimum is called an \textbf{optimal quorum} for the node $s$. Therefore, if a quorum $Q$ in $C$ is an optimal quorum for
the node \( s \), then \( \text{delay}(s, C) = \alpha(s, Q) \). For the coterie \( C \), its \textit{max\_delay} is defined as

\[
\text{max\_delay}(C) = \max\{\text{delay}(s, C) : s \in V\}
\]

and its \textit{mean\_delay} as

\[
\text{mean\_delay}(C) = \frac{1}{|V|} \sum_{s \in V} \text{delay}(s, C)
\]

A coterie on \( G \) with a minimal value of the delay taken over all coteries on \( G \) is called a delay optimal coterie on \( G \). Thus, if \( C(G) \) denotes the set of all coteries on \( G \), coterie \( C^* \) is said to be a \textit{max\_delay optimal coterie} on \( G \) if

\[
\text{max\_delay}(C^*) = \min\{\text{max\_delay}(C) : C \in C(G)\}
\]

It is said to be \textit{mean\_delay optimal} if

\[
\text{mean\_delay}(C^*) = \min\{\text{mean\_delay}(C) : C \in C(G)\}
\]

\text{Fu et al.} [FWLN-94] and \text{Fu} [F-97] use these metrics to obtain delay optimal (cost optimal) coteries for a number of network topologies like trees, rings, hypercubes and grids. They also investigate cost optimal bicoteries/wr-coteries for replicated databases on a clustered network. The definitions given in this subsection are of particular interest to us because communication delay is the central idea around which a major part of this thesis has been developed. We find delay optimal coteries on the \( k \)-dimensional folded Petersen Graph in Chapter 3 and on the Constant Degree 4 Cayley graph in Chapter 4. Also, the communication cost of a \( k \)-coterie, which we define in Chapter 5, derives inspiration from the communication cost of a coterie defined above.
2.6.6. Non-dominated Coteries

As defined earlier, a coterie is non-dominated if there is no coterie which dominates it. The importance of non-dominated coteries can be seen from the fact that they provide more protection against partitions as compared to the coteries they dominate. Also, the message complexity of a non-dominated coterie is expected to be less than the coterie it dominates because most of its quorums are subsets of the quorums of the coterie it dominates. Moreover, there is a very strong chance of finding among non-dominated coteries, the coteries that are optimal in terms of availability. This is so because if a coterie $D$ dominates a coterie $C$, then the probability, that there is a quorum all of whose nodes are operational, is higher in $D$ than in $C$. This explains, to some extent, the reason why so much of importance has been attached to ND coteries.

Attempts have been made to find all possible coteries and ND coteries. While Garcia et al. [GB-85] used hypergraphs to study coteries; Ibaraki and Kameda [IK-91] used Boolean functions. While Ibaraki and Kameda [IK-91] used the self-duality of a function associated with a coterie to characterize ND coteries, Bioch et al. [BI-95] used the concept of almost self-duality of Boolean functions to check whether or not a given positive function is Boolean and gave an algorithm to generate all positive self-dual functions and thus all ND coteries.

Harada and Yamashita [HY-97] characterize $G$-ND coteries in graph theoretic terms and derive a necessary and sufficient condition for the majority coterie on a graph to be $G$-ND and show that if $G=(V,E)$ is a bi-connected
graph with minimum degree $\delta(G) = \frac{3|V|}{4}$, then the majority coterie $C$ on $G$ is $G$-ND.

Kuo and Huang [KH-98] used availability to determine whether a given coterie is ND or not. They showed that

- A coterie $C$ is ND if and only if $A_c(p) = 0.5$ when $p = 0.5$.

- The majority coterie is ND when $N$ is odd.

- The tree coteries are all ND.

- The grid coterie in an $m \times n$ grid is dominated if $m = 1, n = 1$.

So, if we have a general method of finding availability of a given coterie, we will be able to decide whether the given coterie is non-dominated or not. Finding all possible ND coteries is a major research issue.

Thus, we have seen that, quite a number of techniques have been suggested for obtaining coteries. Different coteries have different quorum sizes and thus varying message complexities. Selecting an appropriate coterie from which to choose quorums is an important issue in designing a distributed system. Certainly, non-dominated coteries are preferred as they have better performance in terms of message complexity and availability. Two given coteries can be compared on the basis of a number of metrics like message complexity, synchronization delay, fault-tolerance, availability, communication delay etc. In
Table 2.1, we present the performance comparison of some of the well-known coteries in terms of message complexity.

Table 2.1. Message Complexity of some Coterie-based DME Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ricart-Agrawala</th>
<th>Carvalho et al.</th>
<th>Maekawa</th>
<th>Aggrawal-Abbadi</th>
<th>Bonollo-Sonenberg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Complexity</td>
<td>2*(N-1)</td>
<td>0 to 2*(N-1)</td>
<td>√N</td>
<td>log N</td>
<td>(2a+3b)N</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Akhil Kumar</td>
<td>Billiard</td>
<td>Luk-Wong</td>
<td>Rangarajan-Setia</td>
<td>Cao-Singhal</td>
</tr>
<tr>
<td>Message Complexity</td>
<td>√N</td>
<td>√2√N</td>
<td>√2√N</td>
<td>√(N log N)</td>
<td>o(ck)</td>
</tr>
</tbody>
</table>

2.7. Conclusion

Having gone through a number of permission-based DME algorithms, we find that different algorithms show different performance capabilities with respect to different parameters. These algorithms can be evaluated on the basis of performance measures like message complexity, synchronization delay, availability, communication delay etc. Which algorithm to choose for a particular application is an important decision, a system designer has to make. It is difficult to single out one of these algorithms as the best one with respect to all the performance metrics. An algorithm may be optimal with respect to one of these parameters but may show poor performance as regards another. Besides this, a designer, while choosing an algorithm has to cope with its implementation aspect as well. One can choose between static and dynamic algorithms. Dynamic algorithms optimize performance by adapting to fluctuating system conditions.
One would, certainly, like to choose an algorithm, which is reasonably reliable and has high availability. The higher the availability, the better the algorithm.

A choice can be made between voting-based algorithms and coterie-based algorithms. Vote-assignments and coteries have their respective advantages and disadvantages. Coteries are more general and easy to enumerate. But, the implementation of the algorithms based on coteries is complex because a coterie can be exponential in size. Implementation of coteries requires that the processes keep a list of groups in the coterie. A comparison of responses against this list is required in order to determine whether a process can proceed or not. Addition and removal of nodes may cause all the quorum-sets to be replaced by new ones. On the other hand, voting is more flexible and easier to implement as each node has to maintain only its own vote-assignment and an operation can proceed if the number of votes collected by it is more than or equal to the required quorum. In this case, addition and removal of nodes will only require a change in the quorum.

Also, while choosing an algorithm based on voting, the choice of a vote-assignment becomes an important issue. Different vote-assignments can be compared in terms of likelihood of their halted states. Besides, vote-assignments can have a very significant effect on the node and edge vulnerability and hence on the reliability of the system. Choosing dynamic voting schemes may be useful as these schemes increase resiliency to failures. They help in avoiding halted states and thus provide improved availability. Multi-dimensional voting is another viable alternative as it has the power of coteries and the flexibility and the ease of implementation of voting. It is more powerful than ordinary weighted voting.
because it is equivalent to the method of coteries. A coterie may not be vote-assignable ordinarily but always has a corresponding MD vote and quorum assignment.

Moreover, network topology may play an important role in selecting the mutual exclusion technique to be employed. For some networks like fully connected networks and for Ethernet systems, voting gives optimal availability as shown by Spasojevic and Berman in [SB-94]. But for general networks, coteries may give better availability. At times, actual distances between sites in a network may also be important as suggested by Fu et al. [FWLN-94] and Fu [F-97].

Thus, in a nutshell, we can say that, it is not possible to single out an algorithm as the best algorithm for achieving mutual exclusion. A number of algorithms are available from which we can choose the one which best suits the application in hand, keeping an eye on the system requirement, the network topology and the performance measure that needs to be optimized. In this chapter, we have presented an overview of permission-based DME algorithms. We do not claim our survey to be exhaustive but we have tried to include a good number of algorithms that have contributed to the development of PBDME Algorithms. We have refrained from getting involved in the intricacies of these algorithms. Rather, we have tried to concentrate on the comparative study of these algorithms, bringing out their common features and listing out their differences. In fact, we have laid more emphasis on the technique used (to achieve mutual exclusion). We have made an attempt to bring forth various choices a system designer may have, before he decides in favour of a particular algorithm. Wherever possible, we have
listed out important results that may be useful to those working in this area of research. This, we hope, will come out to be quite handy for a quick reference to permission-based distributed mutual exclusion algorithms and for developing reasonable understanding on the subject.