Chapter 1
Introduction

1.1 Problem Description

Mutual exclusion is one of the first problems met in parallel processing. The implementation of a Mutual Exclusion mechanism is a major task facing every designer of Operating Systems. It is of value to Application Programmers also because they have to use services provided by computer systems built around several processing units, or even several computers linked by a network. The problem of Mutual Exclusion is: to define fundamental operations that make it possible to resolve conflicts resulting from several concurrent processes sharing the resources of a computer system. The whole complex structure of synchronization and communication between the elements of a parallel or distributed system depends upon the existence of such operations. In a Centralized system, all processes share a common memory. Therefore, the synchronization of the accesses to shared data may be achieved through simple primitives operating on a common integer variable called semaphore. A process that needs shared data executes a test on the value of the semaphore (state of the shared resources) to see if it is greater than zero. If so, it decrements the value and continues. Upon finishing the use of the shared data, the process increments the value of the semaphore. If the value of the semaphore is zero, the process is put to sleep and will be woken up when the semaphore's value is greater than zero. Hence, the correctness of the protocol for the management of the accesses to shared data, the ordering of the primitives and a
centralized control are necessary for the correctness of the algorithm (neither deadlock nor starvation). In distributed systems, many problems including concurrency control of replicated data, atomic commitment, distributed shared memory and others require the execution of operations in a mutually exclusive way. Unfortunately, in distributed systems, neither shared memory nor a common clock are available. Therefore, the protocol for the management of the accesses to a shared resource will be more difficult since the ordering of the activities must be done by means of communication messages whose delays are unpredictable. Two types of control are possible for distributed mutual exclusion algorithms:

- **Centralized control**: In this the state of the resource is always in the private data space of one co-ordinator process and each process sends requests to the coordinator and waits for a grant.

- **Distributed control**: In this the state of the shared resources is replicated in the private data space of all or some processes, or it is exchanged among the processes. In both the cases, the decision of the allocation of the resources is an agreement among more than one process.

One of the main advantages of the distributed control is that it is resilient to failures. The centralized control is not. Moreover, centralized control needs two different process codes, one for the coordinator and one for the requesting process. This complicates recovery after failure. The drawback of the distributed control is more complex algorithms.
1.2 Characteristics of Distributed systems

A distributed computing system consists of multiple autonomous processors that do not share common memory but cooperate by sending messages over a communication network. We will also refer to software model with processes running in parallel and that communicate through message passing, as a distributed computing system. In this case a software model of a distributed computing system can be simulated on a centralized architecture. We will use the terms process, processor, site and node as synonyms. In addition to the absence of common memory, in a distributed computing system a global state does not exist and there is no common clock. Therefore, each process has a partial view of the global state and the only external events that it can observe are those of reception of messages. The order of the events occurring in different processes is not obvious and it is provided by means of a logical clock that is synchronized through messages. Before talking about mutual exclusion we need to define the terminology and some basic assumptions that would be used throughout the text.

- **Sites and Processes**: A site is a computer linked to a network. In each site there may be more than one process running in parallel. We only consider sequential processes all executing the same program with the only difference among them being the process identifier (Textual Symmetry).

- **Critical Section (CS)**: The shared resource to which exclusive access is sought would be referred to as critical section (CS).

- **Communication Structure**: The Processes communicate through communication channels. The topological configuration of the communication structure is defined by
a connection graph among the processes in which nodes correspond to processes and edges to channels.

- **Communication Channels**: Each algorithm we will present is based on some of the following communication network assumptions:
  1. Channels are two-way.
  2. Channels do not duplicate messages.
  3. Channels are error free.
  4. Channels are FIFO (the messages are received in the order they were sent to the channel).
  5. The delay of each channel is unpredictable but finite.

If the communication system does not guarantee any of these assumptions software layers must be interposed. The more the number of assumptions the communication subsystem guarantees, the less the communication software support we need to implement it. Therefore, the assumptions on the communication subsystem are an important design parameter.

- **Ordering of Requests**: The ordering of events in distributed system is a key point to ensure control in distributed mutual exclusion algorithms. In particular, such algorithms need a rule that provides a total order of requests. Most algorithms use the timestamp method proposed by Lamport [Lam78] to obtain a total order of requests. Each process i has an integer value that represents its logical clock. Each time i sends a message, it increments by one its logical clock and the new value of the latter is attached to the message. Each time i receives a message from process j, it updates its logical clock according to the following rule:
If \( \text{timestamp}_i < \text{timestamp}_j \) then \( \text{timestamp}_i = \text{timestamp}_j + 1 \)

This rule induces an ordering on the communication events since it implies that the event of the transmission of a message occurs before its reception. Any way, this rule gives only a partial order of events since more than one message may have the same timestamp.

In order to obtain a total order, we say that a message of \( i \) precedes a message of \( j \) iff:

\[
(\text{timestamp}_i < \text{timestamp}_j) \lor ((\text{timestamp}_i = \text{timestamp}_j) \land (i < j))
\]

### 1.3 Issues in Distributed Mutual Exclusion Algorithms

#### 1.3.1 Indices of Comparison

The Performance of Distributed Mutual Exclusion algorithms is compared based on the following indices:

1. **Message Traffic** : This index gives the number of messages exchanged by a process per CS entry. Usually it consists of messages of resource request and other messages to allocate and release the resources. In token based algorithms, it comprises of the number of hops request message make to reach the token and the number of hops token message takes to reach the requesting site.

2. **Time Delay** : It is the average time delay in granting the CS, which is the period of time between the instant a site invokes mutual exclusion and the instant when the site enters the CS.

3. **Resilience to failure** : This index gives a measure of the vulnerability to site or communication line failures of a mutual exclusion algorithm.

Usually these indices are in conflict. Reducing one, others may increase. For example, reducing the time delay, message traffic will increase and vice versa.
1.3.2 Correctness of Distributed Mutual Exclusion algorithm

In order to maintain the integrity of shared resources, a distributed mutual exclusion algorithm works correctly iff the following properties are satisfied:

1. **Safety**: Each resource may be used by at most one process at any given time, i.e. mutual exclusion property.

2. **Liveness**: The system makes progress towards the execution of one CS, i.e. a deadlock situation may not occur.

3. **Fairness**: Each request is eventually satisfied, i.e. no starvation occurs. The algorithm should not be biased towards any particular site.

A key point to obtain safety is to detect all conflicts that may arise. A conflict arises when a process tries to allocate a number of resources greater than those available at that given time. Liveness is preserved by applying the same deterministic rule to resolve each conflict. Fairness is obtained not only by ensuring the detection and resolution of all conflicts, but also by ensuring that conflicts are not always resolved only against some particular process.

1.4 Classification of Mutual Exclusion Algorithms

Over the last decade, the problem of mutual exclusion has received considerable attention and several algorithms to achieve mutual exclusion in distributed systems have been proposed. Recently, there has been a burgeoning growth of distributed mutual exclusion algorithms. These have implied common sense tricks [RA81, SK85] to elegant theories [Mae85] to improve the performance. They have assumed wide variety of communication
topologies (e.g. tree, ring, mesh, hypercubes and arbitrary graphs) and a widely varying amount of information is maintained by each site about other sites. Many algorithms adapt to changing System State to enhance the performance, while others do not. Taxonomy of these algorithms is given by Singhal [Sin93] which gives a common terminology, ground to compare these algorithms and performance metrics to evaluate these algorithms. A complete survey and performance analysis of mutual exclusion algorithms is given by Chang[Cha96]. Mutual exclusion algorithms have employed two approaches to achieve mutual exclusion and can be divided into two broad classes: Token based and non-token based. A hybrid algorithm was proposed by Chang [CSL90b] which combines the technique of both token based and non-token based algorithms. A broad classification of mutual exclusion algorithms is shown in fig. 1.1.

**Fig. 1.1**

1.4.1 **Token based Algorithms**

Fig.1.2 shows the classification of token based algorithms. In token based algorithms, a unique token is shared among the sites. A site is allowed to enter its CS if it possesses the
token. Therefore, the mutual exclusion is trivially guaranteed. However, difficult tasks are (i) to insure fair scheduling of token among competing sites without excessive message overhead and (ii) to detect loss of token and regenerate a unique token. Token based algorithms have proved to be faster than the non-token based algorithms [Cha96]. Token based algorithms also produce lesser message traffic and are not deadlock prone like the non-token based algorithms. But one major drawback is that token based algorithms are less failure resilient. To detect the loss of token and generate a unique token is a major design issue of token based mutual exclusion algorithms. Examples of token based algorithms are [CSL90a, HMR94, HPR88, NM91, NTA96, Ray89b, SK85, YZY96]. Token based algorithms are further classified into two categories: Broadcast based and logical structure based.
1.4.1.1 **Broadcast based Algorithms**

In broadcast based mutual exclusion algorithms, no structure is imposed on sites and the site sends token request messages to other sites in parallel. Broadcast based token based algorithms are further divided into two classes: Static and dynamic.

- **Static algorithms** are memoryless because they do not remember the history of CS executions. In these algorithms, a requesting site sends token to all other sites. Examples of such algorithms are [HPR88, SK85, YZY96, NLM90]. Ricart & Agrawala proposed an algorithm [RA81] in which token is initially held by a site and moves to different sites depending upon the request messages received. A requesting node generates a new sequence number in its own series and sends a request message with this sequence number to all other nodes. When a node which has the token but does not need it notices that the request sequence number for some other node is higher than that the token remembers from its last visit to that node, it sends the token to the requesting node. A similar algorithm was proposed by Suzuki and Kasami [SK85]. In this algorithm the token carries with it the queue which contains the waiting requests. This simplifies the determination of the next site to get the token. Nishio, Li and Manning have developed a resilient mutual exclusion algorithm [NLM90] which is the extension of Suzuki and Kasami’s algorithm. This algorithm checks whether the token is lost during network failure, and regenerates it if necessary. The mutual exclusion requirement is satisfied by guaranteeing that only one token is regenerated in the network. Failures that are taken into account are: processor failure, Communication controller failure and communication link failure.
Helary, Plouzeau and Raynal have proposed a distributed algorithm \cite{HPR88} for mutual exclusion in an arbitrary network which uses flooding broadcast technique to locate the token in the network.

- **Dynamic Algorithms**, on the other hand, remember a recent history of the token locations and send token request messages only to dynamically selected sites which are likely to have the token. An important consideration in these algorithms is that the requesting site must send a token request to a site which either has the token or is going to get it in near future. In Singhal’s algorithm \cite{Sin89}, data structures at sites are initialized and updated such that for any two sites, when any one of them requests the token, that site does send a request message to the other site. Thus, no site is isolated from another site. The key idea behind the algorithm is that a site’s request reaches a site which has the token even though request messages are not sent to all sites. Yan et al have proposed a fast token chasing algorithm which makes use of dynamic state information and topology of the system to achieve optimal delay and message complexity \cite{YZY96}. In this algorithm, each processor sends out a unique request message to chase the token. The path of this message may be modified onroute and it reaches the token faster by making good use of both dynamic state information and network topology information. This algorithm can also tolerate message loss, link crash and processor crash.

1.4.1.2 **Logical structure based Algorithms**

In logical structure based mutual exclusion algorithms, sites are weaved into a logical configuration like tree, ring etc.
These mutual exclusion algorithms can be static or dynamic:

- **Static Algorithms**: In this the logical structure remains unchanged and only the direction of edges changes. Static algorithms have used three logical configurations, namely, tree, graph and ring. In ring based algorithms, sites are arranged in a ring fashion [Lela77, Mar85, NM91]. A token circulates on the ring serially from site to site. A site must wait to capture the token before entering its CS. In tree based algorithms, sites are arranged as a directed tree whose root hold the token [Ray89b, Snep87]. In Raymond's algorithm, a request for token propagates serially in the tree from a requestor node to the root node. Token is passed serially from root to the requestor and as token traverses the edges, the direction of the edges is reversed so that the requestor becomes the new root of the directed tree. Neilson and Mizuno [NM91] have presented an improved version of Raymond's algorithm that requires fewer messages because it passes token directly between processors instead of sequentially through the tree. In graph based algorithms, sites are arranged as a directed graph with a sink node which holds the token [HPR88, Snep87]. A request for token and token propagation is handled in the same way as in tree based algorithms. An advantage of graph based algorithms is that they are fault tolerant to communication link and site failures (because a graph has multiple paths between node while a tree does not). However, a cost for this fault tolerance is increased message traffic because additional messages need to be exchanged to prevent cycles in the graph structure.

- **Dynamic algorithms**: In dynamic algorithms [CSL90a, HMR94, NTA96], a dynamic logical tree is maintained such that the root is always the site which will
hold the token in near future (i.e. the root is the last site to get the token among the current requesting sites) when no message is in transit. The network is assumed to be fully connected. The token is directly sent to the next requesting site to execute the CS, but a request is sequentially forwarded along a virtual path to the root. The logical structure changes as sites request and execute the CS. The shape of the structure, and connectivity all undergo changes. An interesting feature of the dynamic logical structure algorithms is that the logical structure can remember the history of CS execution (i.e. recent token locations) and adjust itself to maximize the performance. In Naimi et al's algorithm [NTA96] each site maintains a pointer "father" which indicates the site to which request for CS access should be forwarded, and the pointer "next" which indicates the site to which token should be forwarded after a site leaves its CS. If next is nil, token is held with the site itself. The next variables at each site form a queue, the head is the process processing the token. The tail is the last process which has requested the CS. A path is so organized that when a process requests entering the CS, the request message is transmitted to the tail. Chang et al [CSL90a] presented an improved algorithm which follows the idea similar to the above discussed Naimi et al's algorithm. This algorithm reduces the height of the logical tree by keeping track of the latest information about the site possibly holding the token; Therefore it reduces message traffic. Moreover, the algorithm does not expand the size of the queue in the token message or at each site. Helary et al have proposed a general information structure and the associated generic algorithm for token- and tree based distributed mutual exclusion algorithms [HMR94]. Their information structure contains a dynamic rooted tree structure logically connecting
the sites involved in the system and a behavior attribute (Transit or Proxy) dynamically assigned to each site. A variable $father_i$ indicates according to site $i$'s knowledge, the site through which token can be reached. It is same as the variable $holder$ in Raymond’s algorithm and the variable $last$ in Naimi–Trehel’s algorithm [TM87a]. The possibility of changing the behavior attribute dynamically allows for development of algorithms better fitted to particular physical support.

1.4.2 Non-token based Algorithms

Fig.1.3 shows the classification of non-token based algorithms. These algorithms require one or more successive rounds of message exchanges among the sites to obtain the permission to execute CS. In these algorithms, a site enters its CS only after an assertion defined on its local variables becomes true. Mutual exclusion is achieved because the assertion becomes true only at one site at any given time. Examples of non-token based algorithms are [Mae85, San87, Sin91, Sin92, Lam86a, Lam86b, RA81]. In non-token based algorithms, a site can be in one of the following three states: requesting the CS, executing the CS, or idling (i.e. neither requesting nor executing the CS). A request for CS is assigned a time-stamp which is generated according to Lamport’s scheme [Lam78]. Timestamps of requests are used to prioritize requests and to resolve conflicts among concurrent requests. Sanders have presented a generalized non-token based mutual exclusion algorithm for Distributed Systems [San87]. The concept of information structures has been introduced to unify different non-token based algorithms in DS.
The information structure of non-token based algorithms

The information structure of a non-token based mutual exclusion algorithm defines the data structure needed at a site to record the status of other sites. The information structure at a site $S_i$ consists of the following three sets: (a) Request set $R_i$, (b) Inform set $I_i$, and (c) Status set $St_i$. These sets consist of the ids of the sites of the system. A site must acquire permission from all the sites in its request set before entering the CS. When the state of a site changes (due to wait to enter the CS or due to exit from the CS), it informs about the change to all sites in its Inform set. The status set $St_i$ contains the sites about which $S_i$ maintains status information.
The generalized non-token based algorithm

To simplify the design of a non-token based algorithm, the following two assumptions are usually made: (1) The network topology is logically fully connected. (2) Between any pair of sites, messages are delivered in the order in which they are sent. To ensure that every site resolves conflicting requests in the same way, a unique time stamp is included in each CS request message to order the requests. A site maintains a variable \text{CSSTAT} which indicates the site’s knowledge of the status of the CS. To execute the CS a site takes the following actions:

The site sends time-stamped request messages to all the sites in its request set. The site executes the CS only after it has received a Grant message from all the sites in its request set. On exit from the CS, the site sends a Release message to every site in its Inform set. Every site maintains a queue which contains request messages, in the order of their timestamps, for which no Grant message has been sent. On the receipt of a request message, a site takes the following actions: Place the request on its queue. If \text{CSSTAT} indicates that the CS is free, then send a Grant message to the site at the top of the queue and remove its entry from the queue. If the recipient of the Grant message is in its status set, then set \text{CSSTAT} to indicate that the recipient site is in the CS. On the receipt of a Release message, a site takes the following actions: \text{CSSTAT} is set to free. If the queue is non-empty, then send a Grant message to the site at the top of the queue and remove its entry from the queue. If the recipient of the Grant message is in its status set, then set \text{CSSTAT} to indicate that the recipient site is in the CS. Repeat the last two steps until \text{CSSTAT} indicates that a site is in the CS or the queue becomes empty. The correctness condition states that for every pair of sites, either they request each others permission or they
request permission of a common site (which maintains the status information of both and arbitrates conflicts between them).

Depending upon the information structure these algorithms can be classified as static and Dynamic non-token based algorithms:

- **In Static** information structure algorithms, the contents of request sets remain fixed and do not change as the sites execute the CS, examples of such algorithms are [CR83, Sin92].

- **In Dynamic** information structure algorithms, the contents of the request sets change as the sites execute the CS, design of such algorithms is much more complex because it requires coming up with rules for updating the information structures such that the correctness for mutual exclusion is satisfied dynamically.

Depending upon how a Request set is formed, the non-token based algorithms are further classified into the following two approaches: Ricart – Agrawala type and Maekawa type.

### 1.4.2.1 **Ricart – Agrawala type Algorithms**

In these algorithms, a site grants permission to a requesting site immediately if it is not requesting the CS or its own request has lower priority. Otherwise, it defers granting permission until its execution of the CS is over. That is, while granting a permission a site looks into only its own conflict. A site can grant permission to many requesting sites simultaneously. This approach must ensure that if site Y is executing the CS and site X is requesting the CS then request set of X should contain Y. The Ricart-Agrawala algorithm [RA81] is static because the contents of the Request sets do not change as the algorithm executes. Carvalho – Roucairol proposed an improved variation of the Ricart-Agrawala
algorithm where a site remembers the recent history of the CS execution and minimizes the number of messages exchanged [CR83]. Singhal has proposed a dynamic Ricart-Agrawala type algorithm [Sin92], which makes use of state information of the system to continuously update its information structure. Dynamic information structure algorithms are attractive because they can adopt to fluctuating system conditions to optimize the performance. Design of dynamic information structure algorithms like [Sin92] requires design of different initial information structures and rules for changing the information structures such that mutual exclusion conditions are always satisfied.

1.4.2.2 Maekawa type algorithms

In Maekawa type algorithms, a site can grant permission only to one site at a time [Mea85, AEA97b, LW97]. A site grants permission to a requesting site only if it has not currently granted permission to another site (site X is locked by site Y if site X grants permission to site Y). Otherwise, it delays granting permission until the currently granted permission has been released. The request set of a site X contains the identifiers of the sites which are exclusively locked by site X when requesting the CS. This approach must ensure that the intersection of the Request sets is non-null, such that two conflicting requests can be detected by a site Z, where Z is a element of the intersection of $R_x$ and $R_y$. Maekawa has proposed a symmetric algorithm [Mae85] which is based on the principle of “equal effort” and “equal distribution of responsibility” among all the sites i.e. size of all Request sets is same and each site is contained in exactly same number of Request sets. Maekawa established a relation between the cardinality of Request sets, $k$, and the number of sites, $N$, using the property of finite projective planes as
\[ N = k \times (k - 1) + 1 \]

which gives \( k \) as \( O(N^{0.5}) \).

One extreme of Maekawa type algorithm is Centralized mutual exclusion algorithm, in which a single site acts as the sole arbiter to resolve conflicts among all the sites. Another extreme of Maekawa type algorithms is the fully distributed mutual exclusion algorithms, where a site requests Maekawa type permission from all other sites. Maekawa has suggested a suboptimal scheme which avoids the construction of finite projective planes and is classified as grid based scheme. The sites are logically arranged in a grid in a shape of square. A quorum for a requesting site includes the union of the row and the column that the requesting site corresponds to. Therefore, the quorum size is roughly twice of the theoretical lower bound i.e. \( k = 2N^{0.5} \). The advantage of this algorithm is that it is simple and geometrically evident but it is not well optimized in the sense that \( R_i \) intersects with \( R_j \) in two sites for all \( i \neq j \). Agrawala et al proposed an alternative grid based method which brings down the quorum size from \( 2N^{0.5} \) to \( (2N)^{0.5} \) [AEA97b]. The idea is to resemble billiard ball paths on a modified grid. This algorithm does not satisfy the equal responsibility property and the size of each Request set has to be an odd integer. Luk & Wong [LW97] have given another suboptimal variation of Maekawa's grid based algorithm. In this the sites are logically arranged in the shape of a triangular grid and the size of the quorum is approximately \( (2N)^{0.5} \). Cheung et al have proposed another variation of Maekawa's algorithm in which the Request set consists of all sites in a column of the grid and exactly one site from each column of the grid [CAA92]. This algorithm exhibits better availability as compared to the above discussed Maekawa type algorithms. Agrawal & Abbadi proposed a fault tolerant Maekawa type algorithm
[AA91] in which the sites are logically organized as a binary tree and the Request set of a site contains all the sites on a path from root to the leaf, contains that site. This algorithm violates the "equal responsibility" property but it achieves fault tolerance to site failures and network partitioning by assigning multiple Request sets to sites. Maekawa type algorithms are in general prone to deadlocks because a site is exclusively locked by other sites and lock requests are not prioritized by their timestamps. Therefore, these algorithms require extra messages to handle deadlocks. Singhal has proposed a deadlock free scheme for Maekawa type algorithms [Sin91].

1.4.3 Hybrid Algorithms

In general, there is a trade-off between the speed and message complexity of distributed mutual exclusion algorithms. No single mutual exclusion algorithm can optimize both the speed and the message complexity. Concept of Hybrid mutual exclusion algorithms has been purported to simultaneously minimize both the time delay and the message complexity [CSL90b]. Hybrid algorithms are capable of combining the advantages of two mutual exclusion algorithms and offer potential of providing improved performance over an extended range of loads.

Sites in the system are divided among several groups and two different mutual exclusion algorithms are used to resolve intra-group and inter-group conflicts. Sites use one algorithm to resolve conflicts with sites in the same group and use a different algorithm to resolve conflicts with sites in different groups. By carefully controlling the interaction between the local and the global algorithms, one can minimize both message traffic and synchronization delay simultaneously. Chang et al [CSL90b] gave a hybrid algorithm
which combines Maekawa’s algorithm [Mae85] and Singhal’s dynamic algorithm [Sin92]. They conducted a performance study which showed that the hybrid algorithm improves message traffic over Singhal’s algorithm and synchronization delay over Maekawa’s algorithm over a wide range of loads.

1.5 K-Mutual Exclusion

In the case when more than one resource is available, two paradigms have been presented in the literature:

- M identical resources are shared among N processes; each process may get at most one resource. Such algorithms are called M-Mutex.

- The k-out-of-M identical resources allocation algorithms (i.e. M identical resources are shared among N processes, each process may require a number of resources that falls between one and M).

M-Mutex is a particular case of the K-out-of-M resources allocation problem. The first distributed solution to the K-out-of-M problem was proposed by Raynal [Ray91] and the basic problem can be described as follows:

In a distributed system there are M identical resources shared among N processes. At any given time each resource may be used by at most one process, and each process may have allocated any number of resources k (1 ≤k≤M). A process requests resources all at once and, to avoid deadlock phenomenon, the process is blocked until it gets all of its requested resources. After that the process starts using the k resources and then releases them all at once. A conflict arises whenever a process tries to allocate a number of
resources greater than those available at that given time. Correctness of k-mutual exclusion algorithms is similar to mutual exclusion algorithms with one shared resource. Raymond has proposed a distributed algorithm for multiple entries to a CS [Ray89a] which is an extension of Ricart & Agrawala’s algorithm. In this algorithm, out of N sites, k sites are allowed to access the CS simultaneously. A node wishing to enter the CS sends request messages to each of the other N-1 nodes, and may enter the CS as soon as (N-k) reply messages are received. Srimani and Reddy proposed another distributed algorithm for multiple entries to a CS [SR91] which achieves almost 50% savings in the number of message exchanges per CS invocation as compared to Raymond’s algorithm. This algorithm requires no message exchanges in the best case and (N+k-1) message exchanges in the worst case per CS invocation. These performance figures compare very favorably with those of Raymond’s algorithm which are (2N-k-1) and 2(N-1) respectively. Recently, there has been a significant interest in fault tolerant methods to solve the k-mutual exclusion problem based on the notion of quorums [KFYA94, CKSTH96, AEA97a]. Fujita et al [KFYA94] use partitioning approach in which the sites are partitioned into k clusters and quorums are constructed so that mutual exclusion is ensured within each cluster. Agrawal et al [AEA97a] have proposed a family of quorum-based protocols that generalize majority quorums for the k-mutual exclusion problem.

1.6 Thesis Contribution

In this thesis we investigate techniques for optimizing the performance specially the time delay in invoking the CS. The main contributions of this thesis are: 
• Effect of varying the connectivity of the network topology on token-and tree based algorithms is studied. A performance analysis reveals that static algorithms are delay optimal at lower connectivities and dynamic algorithms exhibit lower delay only at high connectivities.

• A new delay optimal token based algorithm is proposed which uses two novel techniques to reduce the average delay in accessing the CS. These are on-route servicing and use of token-loc propagators to disseminate token information faster and wider in the network. A performance comparison is made with two existing algorithms which are also token based. Further, on-route servicing technique is inserted in two existing algorithms and the performance gain is shown using simulation techniques.

• A general token- and tree based algorithm is developed which uses priority serialization discipline. Priority serialization disciplines such as Shortest-Job First (SJF), Head-of-Line, Shortest Remaining Job First etc. are useful in several applications to optimize some performance indices. For example, if minimization of the average response time of a request is the main goal of the algorithm then an SJF serialization discipline can be used.

1.7 Thesis Outline

The Thesis is organized as follows:

• Chapter 2 investigates the effects of varying the connectivity of the network topology on the performance of token- and tree based algorithms
• Chapter 3 presents a novel token based algorithm using broadcast approach, which makes use of dynamic state information and network topology information to optimize the delay in accessing the CS. Further, two existing algorithms are modified and a performance gain in terms of lower time delay per CS invocation is achieved.

• Chapter 4 discusses the insertion of fair priority based serialization discipline in token based mutual exclusion algorithms. A broadcast based prioritized algorithm is presented. Further a general algorithm is developed for inserting priority serialization in token- and tree based algorithms.

• Chapter 5 contains the conclusion of the work presented in the thesis and proposes some future work.

Bibliography contains, not only the references cited in the text, but it provides many other references related to the topic of mutual exclusion.