

Chapter 2

State-of-the-Art Review

A distributed system is a collection of autonomous computers connected via a communication network. There is no common memory and processes to communicate through message passing. In many applications, there are critical operations that should not be performed by different processes concurrently. It may sometimes be required that a typical resource is used by only one process at a time. This gives rise to the problem of mutual exclusion. Mutual exclusion (ME) is crucial for the design of distributed systems.

Due to lack of a common shared memory, the problem of ME becomes much more complex in the case of distributed systems as compared to the centralized systems. It needs special treatment. A number of algorithms have been introduced to solve the ME problem in distributed systems. A good distributed mutual exclusion (DME) algorithm, besides providing ME, should take care that there are no deadlocks and starvation does not occur (Fig. 2.1).

The existing DME algorithms typically follow either a token-based [1–4] or a permission-based [1, 5, 6] approach.

2.1 Definitions of Terminologies

Let us first define the terminologies that we have used in this book. The following definitions clearly mark the scope of each keyword in the context of this book.

Fairness of token-based Algorithms

The definition of fairness that we follow here implies that if the token holder P_{hold} receives a token request from some arbitrary process A ahead of the same from some other process B , then the algorithm ensures that after P_{hold} releases the token, process A gets it ahead of process B .

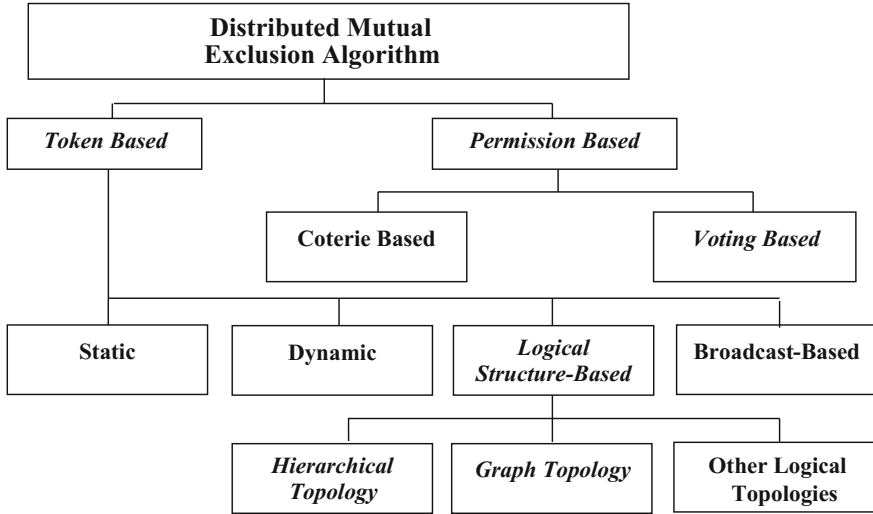


Fig. 2.1 Classification of distributed mutual exclusion algorithm

Liveness

The liveness property implies that any process A that requests for the token must get it eventually.

Safety

A mutual exclusion algorithm is safe if it ensures that no two processes would enter the critical section simultaneously.

Correctness

The correctness of a control algorithm is a combination of the liveness along with safety. A control algorithm is correct if it confirms to both liveness and safety aspects. Correctness of an algorithm implies that it should perform correct action and should avoid performing wrong actions. In case of a distributed control algorithm even the former aspect may be difficult to ensure because processes at which control actions are performed lack a global view of the system and its control data.

Conflict Between Priority Processing and Liveness

The definitions of priority-based fairness and liveness are directly in conflict with one another. Let's assume that a process P_A , with a low priority, requests for the token first. After P_A 's request, other processes P_B, P_C, \dots, P_N all with priorities higher than that of P_A , place their requests for the token. In order to ensure the priority-based fairness defined above, all the processes P_B, P_C, \dots, P_N would receive the token ahead of P_A . In fact, process P_A will never get the token if there is some pending token request from a higher priority process, irrespective of its time of the request. In this situation, the property of liveness is violated. The same had been the major limitation for one of our earlier works, MRA-P [7]. In our attempt to propose a solution that would strike the balance between priority-based fairness and liveness

as defined above, we introduce to dynamically upgrade priorities of token requesting processes.

Priority-based fairness

We assume that the processes have some pre-assigned priorities. It may be desirable to consider the priority of a process while handling the token requests from those. This implies that if the token request from a higher priority process A is pending, then the token must not be allotted to processes having priority lower than that of A . Taking into account the priority aspect, we therefore revise the above definition of fairness with a stronger definition below.

The revised definition of priority-based fairness implies that *the token must be allocated to some process A such that among all the processes having priority equal to that of A , the token request from A has reached P_{hold} ahead of others and there is no pending token request from any other process B having priority higher than that of A .*

Dynamic Process Priority

In our attempt to propose a solution that would strike the balance between priority-based fairness and liveness as defined above, we introduce to dynamically upgrade priorities of processes waiting in the queue for the token after placing the token request. The initial proposal had been somewhat like the following:

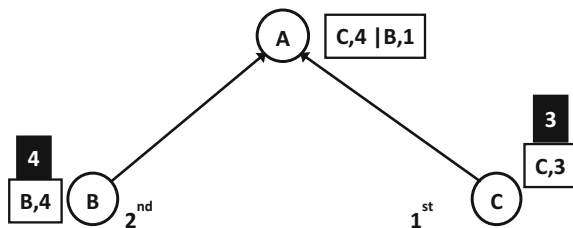
Example

When the token-holding process, say A , finds a token request from another process B with priority r and it is found that r is higher than the priority t for some previous process C whose token request is already stored in the request queue called path list of A , i.e., PL_A , then priority of process C is increased by 1 to $t + 1$. The list in A is re-constructed with this increased process priority for C .

The revised value of priority would be updated in the appropriate sorted sequence of pending requests maintained in the P_{hold} , the token-holding process. The motivation behind such a solution was to elevate the priority of a process A every time some higher priority process B supersedes A by virtue of its higher priority value. This would make sure that even a token request from the otherwise lowest priority process would eventually be satisfied as the priority itself changes dynamically. After a process gets the token, it enters the critical section. As it comes out of the critical section, the priority of a process is reset to its original value.

We explain this solution in Fig. 2.2 with three processes A , B , and C with priority values 4, 4, and 3, respectively. Process C of priority 3 places the first request

Fig. 2.2 Working principle of dynamic priority



for the token. It stores the 2-tuple in PL_C , i.e., $PL_C = \langle C, 3 \rangle$. This request goes to root process A from C using 2-tuple $\langle C, 3 \rangle$. Process A in root stores the 2-tuple in PL_A as $\langle C, 3 \rangle$. Process B now places the second token request. Here $PL_B = \langle B, 4 \rangle$. Process B sends the request to A . Root process A finds that process B has a higher priority and hence places $\langle B, 4 \rangle$ ahead of $\langle C, 3 \rangle$ in PL_A . However, as introduced in rule 1 earlier in this section, the priority of process C would increase by 1, i.e., the 2-tuple modifies as $\langle C, 4 \rangle$. According to priority, process A arranges the 2-tuples $\langle C, 4 \rangle$ and $\langle B, 4 \rangle$ into PL_A . The 2-tuple for C is placed ahead of that for B as the updated priority of C and that of last requesting process B are same while the request from C is older than that of B . The PL_A becomes $\{\langle C, 4 \rangle, \langle B, 4 \rangle\}$.

In order to improve our solution to ensure liveness and also to minimize the data structure maintained in the processes, the dynamic update of the process priority is done only on the request list maintained in the token-holding process.

2.2 Token-Based ME Algorithms

A token is an abstract object which represents a privilege to perform some special operation. Only a process possessing a token can perform these operations; other processes cannot. A token can be passed between processes during operation of the system. This way a privilege can be shared among processes. A single token is shared among all processes in the system and the process which holds it has the exclusive right of executing the critical section (CS). The token represents the privilege to enter a CS. Since single token exists in the system safety of the algorithm is obvious. A process wishing to enter a CS must obtain the token and the token is eventually transferred to the requesting process. The token-based approaches overcome this problem and offer solutions at a lower communication cost, faster than the non-token-based algorithms. But they are deadlock prone and their resilience to failures is poor [8]. If the process holding the token fails, complex token regeneration protocols have to be executed, and when this process 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. Fair scheduling of token among competing processes, detecting the loss of token and regenerating a unique token are some of the major design issues of the token-based ME algorithm.

There exist different ME algorithms for message passing distributed systems [1, 5, 9–13]. Some of these algorithms have been fine-tuned to suit the needs of real time systems in [2, 8, 14, 15]. One of the earlier algorithms for group mutual exclusion (GME) [9, 16–18] was given by Joung in [17]. Later, he proposed two algorithms RA1 and RA2 based on Ricart–Agrawala algorithm [14] to ensure ME for message passing systems [11]. There are several token-based algorithms for ME in distributed systems [19–23]. Mittal–Mohan algorithm [9] considers the concept of priority. In Mittal–Mohan algorithm a requesting process cannot assign priority to a request. Raymond has designed an efficient token-based ME algorithm. Raymond’s algorithm [8] assumes an inverted tree topology. One major limitation

of Raymond's algorithm is the lack of fairness in the sense that a token request that is generated at a later instance may be granted ahead of a previous request.

A DME algorithm is static if it does not remember the history of CS execution. In static algorithms, nodes need not keep information about the current state of the system. In Raymond's [12] token-based mutual exclusion algorithm, requests are set over a static spanning tree of the network, toward the token holder. This logical topology is similar to that of Housn and Trehel's approach [5].

In dynamic algorithms, processes keep track of the current state of the system and the algorithm has a built-in mechanism to make a decision on the basis of this knowledge. Using dynamic approach, a number of solutions have been proposed for prioritized mutual exclusion in a distributed system. Some of these approaches impose a higher message passing overhead.

In broadcast-based algorithms, no such structure is assumed and the requesting process sends message to other processes in parallel, i.e., the message is broadcasted. Suzuki and Kasami proposed a broadcast algorithm [11], in which each process S_i keeps an array of integers $R_{Ni}[1...N]$, where $R_{Ni}[j]$ is the largest sequence number received so far from S_j . In Suzuki-Kasami algorithm, 0 or N messages are sent per critical section executed and synchronization delay is 0 or T . In Naimi and Thiare [24] implemented the causal ordering in Suzuki-Kasami token-based algorithm in a distributed system. Based on Suzuki-Kasami algorithm, Singhal [21] proposed a heuristically aided algorithm that uses state information to more accurately guess the location of the token. The maximum number of messages required by these algorithms is of the order of the total number of nodes in the system.

An interesting algorithm has been proposed in [25] by extending Naimi-Trehel token-based algorithm [24] that reduces the cost of latency and numbers of messages exchanged between far hosts. The work is on hierarchical proxy-based approach for aggregation of request and permission for token by closed hosts. In [26], another hierarchical token-based algorithm is proposed to solve the GME [Group Mutual Exclusion] problem in cellular wireless networks. They claim that the algorithm is the first GME algorithm for cellular networks. In [27], a rooted tree named "open-cube" is introduced that has noteworthy stability and locality properties, allowing the algorithm to achieve good performances and high tolerance to node failures. In [28], a token-based mutual exclusion algorithm is proposed for arbitrary topologies. This algorithm makes use of the network topology and site information held by each node to achieve an optimal performance. It reduces the delay in accessing the CS by allowing the token to service the requesting site which falls en route its destination. In [29], an algorithm, called "Awareness", is proposed that aims at reducing the maximum response time whereas the number of priority violations remains low. The objective is to both postpone priority increments and prevent low priorities from increasing too fast. However, in this case, the response time of low priorities may considerably increase. In [30], the proposed algorithm

can reduce the number of the messages exchanged by 40% when the traffic is light compared with Suzuki and Kasami's algorithm [11]. In [31], a general algorithm is proposed which can represent non-token-based algorithms known as information structure distributed mutual exclusion algorithms (ISDME). The work also introduced a new deadlock-free ISDME algorithm (DF-ISDME) which operates on a restricted class of information structures. DF-ISDME allows deadlock-free solutions for a wider class of information structure topologies than Maekawa algorithms (DF-Maekawa). In [32], an algorithm has been designed which uses a distributed queue strategy and maintains alternative paths at each site to provide a high degree of fault tolerance. However, owing to these alternative paths, the algorithm must use reverse messages to avoid the occurrence of directed cycles, which may form when the directions of edges are reversed after the token passes through. In [33], research is done on a distributed mutual exclusion algorithm based on token exchange and is well suited for mobile ad hoc networks is presented along with a simulation study. A new algorithm is developed for a dynamic logical ring topology. The simulation study on a mobile ad hoc network identifies an effective reduction in the number of hops per application message. This can be achieved by using a specific policy to build the logical ring on-the-fly. In [34], they presented token based mutual exclusion algorithms to handle AND synchronization problems where each process can obtain an exclusive access to a set of resources rather than to a single resource. The message complexity of the algorithms is independent of the number of the requested resources. In [35], another hierarchical token-based algorithm is proposed to solve the GME problem in cellular wireless networks. Arguably, this could be the first GME algorithm for cellular networks. In this algorithm, a resource-starved mobile host requires very little data structure and the bulk of the computation is performed at the resource-rich base station level.

In [36], a simple protocol is proposed that ensures sequential consistency when the shared memory abstraction is supported by the local memories of nodes. It can communicate only by exchanging messages through reliable channels. In [37], a new token-based mutual exclusion algorithm is presented that uses quorum agreements. When a good quorum agreement is used, the overall performance of the algorithm compares favorably with the performance of other mutual exclusion algorithms. In [38], a token-based distributed mutual exclusion algorithm for multithreaded systems is proposed. Several per-node requests could be issued by threads running at each node. Their algorithm relies on special-purpose alien threads running at host processors on behalf of threads running at other processors. The algorithm uses a tree to route requests for the token. The work in [39] is another improvement of Suzuki and Kasami's algorithm [11]. This is rather an extension of the work in [30] that presents a dynamic token-based mutual exclusion algorithm for distributed systems. In this algorithm, a site invoking mutual exclusion sends token request messages to a set of sites possibly holding the token as opposed to all the sites as in the algorithm proposed by Suzuki and Kasami.

2.3 Hierarchical Topology Based ME Algorithms

The processes in the system are assumed to be arranged in a logical configuration like tree, ring, etc., and messages are passed from one process to another along the edges of the logical structure imposed. Token-based algorithms use abstract system models whose control edges form convenient graph topologies. Abstract ring and tree topologies are mostly used. We focus on only tree topology. Raymond's algorithm, MRA-P and FAPP use an abstract inverted tree as the system model. It is called an inverted tree because the control edges point toward the root, rather than away from it. P_{hold} designates the process in possession of the privilege token. Raymond's algorithm uses a spanning tree and the number of message exchanged per CS depends on the topology of the tree. An algorithm, based on a dynamic tree structure, was proposed by Trehal and Naimi [24]. The Naimi–Trehal algorithm takes into consideration a second pointer that is used when a node requests to enter CS before the previous requester leaves it. A major limitation of these two tree-based algorithms is in maintaining this strictly hierarchical logical topology. The algorithm proposed by Bernabeu-Auban and Ahamad in [12] uses path reversal. A path reversal at a node x is performed as the request by node x travels along the path from node x to the root node. Token based mutual exclusion algorithms provide access to the CS through the maintenance of a single token that cannot simultaneously be present at more than one process in the system. Requests for critical section entry are typically directed to whichever process is the current token holder. The token-based solutions for the mutual exclusion problems have an inherent safeness property, as the requesting process must get hold of the token before it enters the critical section. The MRA-P [7] is an improvement over Raymond's algorithm that not only overcomes the fairness problem but also handles the priority of the requesting processes [1]. The work on MRA-P has further motivated others to propose group mutual exclusion algorithms for real time systems [9]. The work, however, used too many control messages across the network. In [40], a service level agreement (SLA) aims to support quality of service (QoS) for services by a cloud provider to its client. Since applications in a cloud share resources, they build on two tree-based distributed mutual exclusion algorithms (Kanrar–Chaki algorithm [41] and Raymond's algorithm [8]) that support the SLA concept. In [42], a good survey work is reported. This analyzes several DME algorithms according to their relative characteristics like Maekawa-type approach [43], hybrid approach etc. In [44], a logical grouping of the sites is done to form a hierarchical structure. This structure is not rigid and can be modified to achieve different performance criteria. The hybrid algorithm needs only a maximum of $2\sqrt{N} + 1$ messages where N is the number of sites in the distributed system.

In [45], the work on GME deals with sharing a set of mutually exclusive resources among all n processes of a network. Three new group mutual exclusion solutions are designed for tree networks. In [46], the work on DME algorithm is for Grids. Two new DME algorithms are proposed based on Naimi–Trehal token-based

algorithm [24]. These take into account latency gaps, especially those between local and remote clusters of machines.

2.4 Graph Topology Based ME Algorithms

Solutions for strictly hierarchical topology, e.g., tree, may not be usable for many applications. Besides, if the communication media is wireless and/or nodes are mobile as in MANET, frequent link failures may cause performance problem. Graph-based solution could be more fault-tolerant for such applications and communication media. Raymond's algorithm [8] or MRA-P [7] assumes an inverted tree topology while Walter et al. [47] assume a dynamically re-configurable DAG structure. Raymond's work, MRA-P or its extension in [7] are, however, not quite suitable in a wireless environment, where the topology keeps on changing due to link instability. The solution proposed in [48] works on a dynamically changing DAG structure. Their algorithm handles link failures and new link formation. However, in a constantly changing network environment, absence of cycle in the topology may not be guaranteed. Hence, a DAG based solution may not be ideal for the highly dynamic environment. Moreover, Walter's work does not ensure fairness. In [49], a link failure resilient algorithm has been introduced for mutual exclusion on directed graph topology.

Token based algorithms for ME are proposed for the distributed environment assuming inverted tree topology [7, 8]. However, such a stable, hierarchical topology is quite unrealistic for mobile networks where link failures [10, 47, 49–54] are frequent due to node mobility. In this book, new ME algorithms have been introduced on graph topology. In [55], a new fault-tolerant distributed mutual exclusion algorithm is proposed which can tolerate an arbitrary number of node failure and the message complexity of the algorithm is relatively very low. They claimed that the algorithm functions properly when any of the cooperating nodes in the system fails. In [56], another algorithm is proposed to achieve mutual exclusion in distributed systems. This tolerates up to $t - 1$ arbitrary node failures without executing any recovery procedure and requires $O(\sqrt{m})$ messages in a network of n nodes. In [57], a dynamic triangular mesh protocol is proposed for mutual exclusion. Here the protocol is fault-tolerant up to $(k - 1)$ site failures and communication failures in the worst case, even when such failures lead to network partitioning. In [58], a mutual exclusion mechanism is developed to ensure that isolated groups do not concurrently perform conflicting operations, when a network is partitioned. The work formalizes these mechanisms in three basic scenarios: where there is a single conflicting type of action; where there are two conflicting types, but operations of the same type do not conflict; and where there are two conflicting types, but operations of one type do not conflict among themselves.

2.5 Permission-Based ME Algorithms

In this type of ME algorithms, a process P_k can enter in its critical section only after it obtains permission(s) from other processes in the system that share the same common resource in the respective critical sections. It requires rounds of message among the processes to obtain the permission to execute CS. In permission-based algorithm or symmetric algorithm, a process wishing to enter a critical section (CS) needs to consult other processes. Processes enter CS in FIFO order. Generally, priority decisions are made using time stamps. A process wishing to enter CS sends time-stamped request messages to all other processes and waits till it received a “go ahead” reply from each of them. A process receiving a request message immediately sends a “go ahead” reply if it is not interested in using the CS at present or if the received request precedes its own request in time. Otherwise, it delays its reply until it has itself used the CS. A number of solutions in both the approaches have been studied. Permission-based algorithms tend to increase the network traffic significantly.

Every process has a logical clock and all request messages are time-stamped with the current value of the sender’s logical clock. Before sending a message, a process increases its logical clock and, upon reception of a message, a process updates it with the maximum between the message time-stamp value and its current value. A total ordering of request messages can thus be established based on the value of their time stamps and, if necessary, the identity of the sender process in order to break ties. In other words, a request message whose time stamp is lower than a second one has priority over it. If the time stamps are equal, the request message sent by the process with the lowest identifier has priority. In both cases, the request message with higher priority has precedence over the other one. When a process receives a request, it answers either if it is not interested in using the resource (i.e., if the process is in the idle state (I)) or if it is interested, but the received request message has precedence over its own; otherwise it answers to all requests it had received upon executing the exit protocol (exiting the critical section). At that moment, the process also changes its state to idle.

Permission-based mutual exclusion algorithms exchange messages to determine which process can access the CS next. As for example, Lamports’ algorithm [5] is a symmetric, permission-based algorithm that requires $3 * (N - 1)$ messages to provide mutual exclusion. The permission-based algorithm proposed by Ricart and Agrawala, reduced the number of required messages to $2 * (N - 1)$ messages per critical section entry [14]. In another work, Carvalho and Roucairol [13] were able to reduce the number of messages to be between 0 and $2 * (N - 1)$ and Sanders [22] gave a theory of information structure to design mutual exclusion algorithms. An information structure describes which processes maintain information about what other processes, and from which processes a process must request information before entering the CS. Singhal’s heuristic algorithm [23] guarantees some degree of fairness but is not completely fair, as a lower priority request can execute CS before a higher priority request if the higher priority request is delayed. The

algorithm has defined different criteria for fairness. Mueller [2] introduced a prioritized token-based mutual exclusion algorithm. In many of the solutions, the fairness criterion is considered to be equivalent to the priority, i.e., a job that arrives early is defined as a higher priority job [24]. In [50], Sil and Das introduced a new energy efficient mutual exclusion algorithm for a mobile ad hoc network. The whole network is hierarchically clustered to get a logical tree structure of the network. They addressed the issue of mobility of nodes extensively. In [3], Saxena and Rai present a survey of various permission-based distributed mutual exclusion algorithms and their comparative performance. Permission-based algorithms can be further subdivided into coterie-based algorithms and voting-based algorithms.

Acoterie is a nonempty set with a collection of elements satisfying two conditions: the minimal condition and intersection property. The elements of a coterie form overlapping quorum groups or quorum sets or just quorums. If a process wants to perform an operation in the CS, it must obtain permission from each and every process of the quorum to which the process belongs. The candidate process may belong to two or more quorums. It has to get permission from each member of the overlapping quorums. Since a process grants permission to only one process at a time and since any two quorums in a coterie have at least one process in common, mutual exclusion, i.e., safety property is guaranteed [59]. In [60], the primary aim is to investigate the use of a gossip protocol to an ME algorithm to cope with scalability and fault-tolerant problems in cloud computing systems. They present a gossip-based mutual exclusion algorithm for cloud computing systems with dynamic membership behavior. In [61], an algorithm called prioritizable adaptive distributed mutual exclusion (PADME) is proposed to meet the need for differentiated services between applications for file systems and other shared resources in a public cloud. In [62], a hybrid distributed mutual exclusion algorithm is designed that uses Singhal's dynamic information structure algorithm [21] as the local algorithm to minimize time delay and Maekawa's algorithm [43] as the global algorithm to minimize message traffic. In [63], the author presents an N -process local-spin mutual exclusion algorithm, based on non-atomic reads and writes. Each process performs $\Theta(\log N)$ remote memory references to enter and exit its critical section. In [64], authors present a distributed algorithm for solving the GME problem based on the notion of surrogate-quorum. Intuitively, they use the quorum that has been successfully locked by a request as a surrogate to service other compatible requests for the same type of critical section. In [65], an interesting algorithm based on GME is for the philosophers to ensure that a philosopher attempting to attend a forum will eventually succeed, while at the same time encourage philosophers interested in the same forum to be in the meeting room simultaneously. Another approach, based on the work in [64] offers a high degree of concurrency of n , where n is the number of processes in the system [66]. The algorithm can adapt without performance penalties to dynamic changes in the number of groups. In [67], an FCFS group mutual exclusion algorithm is introduced that uses only $\Theta(N)$ bounded shared registers. They also present a reduction that transforms any "abortable" FCFS mutual exclusion algorithm M into a GME algorithm G . Thus, different group mutual exclusion algorithms can be obtained by

instantiating M with different abortable FCFS mutual exclusion algorithms. In [68], Vidyasankar introduced a situation where philosophers with the same interest can attend a forum in a meeting room, and up to k meeting rooms are available. In [69], a class of Maekawa-type mutual exclusion algorithms [43] is presented that is free from deadlocks and do not exchange additional messages to resolve deadlocks.

In [70], a new quorum generation algorithm has been presented. A symmetric quorum can be generated based on this algorithm and the size of the quorum is about $2\sqrt{N}$. This distributed mutual exclusion algorithm has reduced the message complexity of Maekawa-type distributed mutual exclusion algorithm to m messages, m is the quorum size. In [71], an analysis is done on the shared memory requirements for implementing ME of N asynchronous parallel processes in a model where the only primitive communication mechanism is a general test-and-set operation on a single shared variable. In [72], the concept of coterie is extended to k -coterie for k entries to a critical section. A structure named Cohorts is proposed to construct quorums in a k -coterie. The solution is resilient to node failures and/or network partitioning and has a low communication cost. In [73], a surficial quorum system for GME is presented. The surficial quorum system is geometrically evident and so it is easy to construct. It also has a nice structure based on which a truly distributed algorithm for GME can be obtained, and loads of processes can be minimized.

2.6 Voting-Based ME Algorithms

In voting-based algorithm, a decision is taken based on voting. Unlike coterie-based algorithms, each and every process in a system of n processes need not wait for permission in terms of votes from each of the remaining $n - 1$ processes in the system. Hence, the message complexity of voting-based ME algorithms is lower than the symmetric ME algorithms [74]. In voting-based algorithms, each candidate process seeking entry into CS sends request messages for votes to other processes. The votes are counted on receipt of the reply messages (votes) from various nodes. If the number of votes obtained is more than or equal to majority then the corresponding candidate process is granted access to enter its critical section. The voting schemes [75, 76] are often far from being correct as it may suffer from lack of liveness. Say, a network is partitioned and only 85% of the $n - 1$ processes are connected. If there are two requesting processes, it may lead to a scenario where both the requesting processes get more than 40% of the votes. However, none of these would achieve majority consensus. A similar situation may occur when there are more than two requesting processes even if the network is not partitioned and all 100% of the votes are cast.

Many simple majority voting schema [59, 76–83] are far from being correct as they may suffer from lack of liveness. Jajodia and Mutchler [84] proposed two generalizations of voting schemes: dynamic voting and dynamic-linear voting (i.e., dynamic voting with linear ordered copies). In dynamic voting, the number of sites necessary for carrying out an update is a function of the number of up-to-date

copies at the time of the update. In case of static algorithms, this depends on the total number of copies. Moreover, dynamic-linear voting accepts an update whenever dynamic voting does.

Rabinovich and Lazowska [78] proposed 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 to devise a rule that unambiguously imposes a desired logical structure on a given ordered set of processes. In this protocol, each node is assigned a name and all names are linearly ordered. Among all the nodes replicating a data item, a set of nodes is identified as the current epoch. At any instance, the data item may have only one current epoch associated with it.

Adam [79] also proposed a dynamic voting consistency algorithm, which is effective in environments where the majority of user requests are “read” types of requests. This algorithm also works on the majority partition would be available even if changes in the network topology take place at a higher rate than the update rate, as long as only simple partitioning takes place.

In [81], works on general voting protocol are discussed that reduces the vulnerability of the voting process to both attacks and faults. This algorithm is contrasted with the traditional two-phase commit protocols. The algorithm is applicable to exact and inexact voting in networks where atomic broadcast and predetermined message delays are present.

In [82], two dynamic voting algorithms are proposed. The “optimistic dynamic voting”, operates on possibly out-of-date information. On the other hand, “topological dynamic voting” works on the topology of the network on which the copies reside to increase the availability of the replicated data.

Paris and Long [83] proposed a dynamic voting protocol that does not need the instantaneous state information required by other dynamic voting algorithms. This algorithm also served low cost in traffic similar to that of static majority consensus voting.

In the existing literature as reviewed, no work is found that handles the situation when none of the candidate processes achieve majority consensus. The progress condition demands that at least one of the competing processes must be selected for CS execution even when majority consensus is not achieved by any process. On the other hand, the liveness property demands that all competing processes must eventually be allowed to enter the CS. Our work aims to address this gap in the existing literature and proposes a new voting-based algorithm that ensures both progress condition and safety even when no single process wins majority vote. An innovative solution introduced later in this text is also compatible with any voting mechanism that ensures liveness.

In static voting algorithm, the processes do not keep information about the current state of the system and the votes, once assigned, are not changed as the algorithm evolves [75]. Some of the earliest voting-based distributed mutual exclusion algorithms were given by Thomas [75] and Gifford [56]. This was static in nature in which the votes were fixed a priori and the distributed system is assumed to be fully connected by message passing. A process requesting to enter

the CS must obtain permission from majority of processes in distributed system; otherwise, it must not enter the CS and wait until it is allowed to enter the critical section. Xu and Bruck [80] proposed a deterministic majority voting algorithm for NMR (N Modular Redundant) systems where N computational modules execute identical tasks and they need to periodically vote on their current states. Agrawal and Abbadi [85] 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 communication requirements for read and write operations and call it “Tree Quorum Protocol”. This algorithm may be considered as a generalization of the static voting-based algorithms. However, when a distributed system is partitioned as a result of node or link failure, a static voting algorithm cannot adapt to the changing topology toward maintaining system availability. Thus, in order to avoid a hang-parliament status, it is necessary to change the paradigm to dynamic voting.

In dynamic voting algorithm, the processes keep track of the current state of the system and in the case of network partitioning, due to link or process failures, new votes may be assigned so as to make at least one partitioned group of processes active and to keep the system working [86, 87]. In this category, a distinguished partition may still have the majority of votes after next partitioning. In order to avoid data inconsistency, dynamic vote reassignment is only possible inside the distinguished partition and the votes of other partitions remain unchanged. It is due to the fact that partitions are unaware of each other. Group consensus and autonomous reassignment are two dynamic vote assignment techniques presented in [77]. In the first approach a process that firstly discovered partitioning, initiates itself as a candidate and establishes an election among the processes in its region. Then, if the majority of processes agreed on its coordination, it would announce itself as new coordinator and install a new vote reassignment such that only processes in majority partition can commit a request. Henceforth, all votes of processes will modify to new votes and mutual exclusion will be achieved in at most one partition. In the second approach, there is no centralized coordinator and each process is responsible for assigning its vote. In order to avoid each process to arbitrarily change its vote and probably compromise mutual exclusion, an approval of a certain number of processes must be obtained formerly. Autonomous reassignment is similar to group consensus except that each node is responsible for deriving its own vote. In symmetric algorithm a process wishing to enter a CS sends time-stamped request messages to all other processes and waits till it received a grant messages from each of them. The process executes CS whose time stamp is lowest like Ricart–Agrawala [14]. Singh and Bandyopadhyay [42] presents a work on mutual exclusion that ensures liveness, and correctness properties. Timed-buffer distributed voting algorithm (TB-DVA) [74] is a secure distributed voting protocol. It is unique for fault tolerance and security compared to several other distributed voting schemes. TB-DVA is a radical approach to distributed voting because it reversed the two-phase commit protocol: a commit phase (to a timed buffer) is followed by a voting phase. This conceptually simple change greatly enhances security by forcing an attacker to compromise a majority of the voters in order to corrupt the system. Recall that in the two-phase commit protocol only one voter must be compromised

to corrupt a system. In [88], a vote assignment algorithm is introduced toward maximizing the reliability. In this approach, the voting weight assigned to each node is readily determined if the link failure rate is negligible. Majority voting is commonly used in distributed computing systems to control mutual exclusion, and in fault-tolerant computing to achieve reliability.

2.7 Conclusions

Ensuring mutual exclusion among the processes is an important aspect. The existing algorithms follow either a permission-based or a token-based approach. Studies on a number of solutions in both the approaches reveal that the permission-based algorithms tend to increase the network traffic significantly. The token-based approaches perform better from this perspective and offer solutions at a lower communication cost.

References

1. Housini, A., Trehel, M.: Distributed mutual exclusion token-permission based by prioritized groups. In: Proceedings of ACS/IEEE International Conference, pp. 253–259 (2001)
2. Mueller, F.: Prioritized token-based mutual exclusion for distributed systems. In: Proceedings of the 9th Symposium on Parallel and Distributed Processing, pp. 791–795 (1998)
3. Saxena, P.C., Rai, J.: A survey of permission-based distributed mutual exclusion algorithms. *Comput. Stan. Interfaces* **25**(2), 159–181 (2003)
4. Helary, J.M., Mostefaoui, A., Raynal, M.: A general scheme for token and tree based distributed mutual exclusion algorithm. *IEEE Trans. Parallel Distrib. Syst.* **5**(11), 1185–1196 (1994)
5. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* **21**(7), 558–565 (1978)
6. Madria, S.K.: Timestamp based approach for the detection and resolution of mutual conflicts in real-time distributed systems. *Computer Science Technical Reports. Paper 1367*, pp. 1–16 (1997)
7. Karnar, S., Chaki, N.: Modified Raymond's algorithm for priority (MRA-P) based mutual exclusion in distributed systems. In: Proceedings of ICDCIT 2006. LNCS 4317, pp. 325–332 (2006)
8. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. *ACM Trans. Comput. Syst.* **7**, 61–77 (1989)
9. Mittal, N., Mohan, P.K.: A priority-based distributed group mutual exclusion algorithm when group access is non-uniform. *J. Parallel Distrib. Comput.* **67**(7), 797–815 (2007)
10. Walter, J.E., Welch, J.L., Vaidya, M.H.: Mutual exclusion algorithm for Ad hoc mobile networks. *Wirel. Network* **7**(6), 585–600 (2001)
11. Suzuki, Kasami, T.: An optimality theory for mutual exclusion algorithms in computer science. In: Proceedings of IEEE International Conference on Distributed Computing and System, pp. 365–370 (1982)
12. Bernabeu-Auban, J.M., Ahamad, M.: Applying a path-compression technique to obtain an efficient distributed mutual exclusion algorithm, vol. 392, pp. 33–44 (1989)

13. Carvalho, O.S.F., Roucairol, G.: On mutual exclusion in computer network. *Commun. ACM* **26**(2), 146–147 (1983)
14. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM* **24**(1), 9–17 (1981)
15. Lodha, S., Kshemkalyani, A.: A fair distributed mutual exclusion algorithm. *IEEE Trans. Parallel Distrib. Syst.* **11**(6), 537–549 (2000)
16. Swaroop, A., Singh, A.K.: A Distributed group mutual exclusion algorithm for soft real-time systems. In: *Proceedings of World Academy of Science, Engineering and Technology*, vol. 26, pp. 138–143 (2007)
17. Joung, Y.J.: Asynchronous group mutual exclusion. In: *Proceedings of the 17th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 51–60 (1998)
18. Manabe, Y., Park, J.: A quorum based extended group mutual exclusion algorithm without unnecessary blocking. In: *Proceedings of 10th International Conference on Parallel and Distributed Systems (ICPADS'04)* (2004)
19. Attreya, R., Mittal, N.: A dynamic group mutual exclusion algorithm using surrogate quorums. In: *Proceedings of the 25th IEEE Conference on Distributed Computing Systems (ICDCS'05)* (2005)
20. Lin, D., Moh, T.S., Moh, M.: Brief announcement: Improved asynchronous group mutual exclusion in token passing networks. In: *Proceedings of PODC'05*, pp. 275–275 (2005)
21. Singhal, M.: A heuristically-aided algorithm for mutual exclusion for distributed systems. *IEEE Trans. Comput.* **38**(5), 70–78 (1989)
22. Sanders, B.: The information structure of distributed mutual exclusion algorithm. *ACM Comput. Syst.* **5**(3), 284–299 (1987)
23. Singhal, M.: A dynamic information structure mutual exclusion in distributed system. *IEEE Trans. Parallel. Distrib. Syst.* **3**(1), 121–125 (1992)
24. Naimi, M., Trehel, M., Arnold, A.: A $\log(N)$ distributed mutual exclusion algorithm based on path reversal. *J. Parallel. Distrib. Comput.* **34**(1), 1–13 (1996)
25. Bertier, M., Arantes, L., Sens, P.: Hierarchical token based mutual exclusion algorithms. In: *IEEE International Symposium on Cluster Computing and the Grid*, pp. 539–546 (2004)
26. Swaroop, A., Singh, A.K.: A token-based group mutual exclusion algorithm for cellular wireless networks. In: *India Conference (INDICON)*, pp. 1–4 (2009)
27. Helary J.M., Mostefaoui A.: A $O(\log_2 n)$ fault-tolerant distributed mutual exclusion algorithm based on open-cube structure. In: *14th International Conference in Distributed System*, pp. 89–96 (1994)
28. Saxena, P.C., Gupta, S.: A token-based delay optimal algorithm for mutual exclusion in distributed systems. *Comput. Stan. Interfaces* **21**(1), 33–50 (1999)
29. Lejeune, J., Arantes, L., Sopena, J.: A $O(\log_2 n)$ fault-tolerant distributed mutual exclusion algorithm based on open-cube structure. In: *42nd International Conference of Parallel Processing (ICPP)*, pp. 290–299 (2013)
30. Chang, YI.: A dynamic request set based algorithm for mutual exclusion in distributed systems. *Operating Syst. Rev.* **30**(2), 52–62 (1996)
31. Bonollo, U., Sonenberg, E.A.: Deadlock-free information structure distributed mutual exclusion algorithms. *Aust. Comput. Sci. Commun.* **18**, 234–241 (1996)
32. Chang, Y.I., Singhal, M., Liu, M.: A fault tolerant algorithm for distributed mutual exclusion. In: *Ninth IEEE Symposium on Reliable Distributed Systems*, pp. 146–154 (1990)
33. Baldoni, R., Virgillito, A., Petrassi, R.: A distributed mutual exclusion algorithm for mobile Ad-hoc networks. In: *Seventh International Symposium on Computers and Communications (ISCC 2002)*, pp. 539–544 (2002)
34. Maddi, A.: Token based solutions to M resources allocation problem. In: *ACM Symposium on Applied Computing*, pp. 340–344 (1997)
35. Swaroop, A., Singh, A.K.: A token-based group mutual exclusion algorithm for cellular wireless networks. In: *India Conference (INDICON-2009)*, pp. 1–4 (2009)

36. Raynal, M.: Token-based sequential consistency in asynchronous distributed systems. In: 17th International Conference on Advanced Information Networking and Applications (AINA 2003), pp. 421–426 (2003)
37. Mizuno, M., Neilsen, M.L., Rao, R.: A token based distributed mutual exclusion algorithm based on quorum agreements. In: ICDCS, pp. 361–368 (1991)
38. Meza, F., Perez, J., Eterovic, Y.: Implementing distributed mutual exclusion on multithreaded environments: The alien-threads approach. In: Advanced Distributed Systems, pp. 51–62 (2005)
39. Chang, Y.I., Singhal, M., Liu, M.T.: A dynamic token-based distributed mutual exclusion algorithm. In: Tenth Annual International Phoenix Conference on Computers and Communications, pp. 240–246 (1991)
40. Lejeune, J., Arantes, L., Sopena, J., Sens, P.: Service level agreement for distributed mutual exclusion in cloud computing. In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 180–187 (2012)
41. Kanrar, S., Chaki, N.: FAPP: A new fairness algorithm for priority process mutual exclusion in distributed systems. Special issue on recent advances in network and parallel computing. *Int. J. Networks* 5(1), 11–18 (2010). ISSN: 1796-2056
42. Chang, Y.I.: A simulation study on distributed mutual exclusion. *J. Parallel Distrib. Comput.* 33(2), 107–121 (1996)
43. Maekawa, M.: A \sqrt{n} algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.* 3(2), 145–159 (1985)
44. Madhuram, S., Kumar, A.: A hybrid approach for mutual exclusion in distributed computing systems. In: Sixth IEEE Symposium on Parallel and Distributed Processing, pp. 18–25 (1994)
45. Beauquier, J., Cantarell, S., Datta, A.K., Petit, F.: Group mutual exclusion in tree networks. In: Ninth International Conference on Parallel and Distributed Systems, pp. 111–116 (2002)
46. Bertier, M., Arantes, L., Sens, P.: Distributed mutual exclusion algorithms for grid applications: A hierarchical approach. *J. Parallel. Distrib. Comput.* 66(1), 128–144 (2006)
47. Walter, J., Cao, G., Mohanty, M.: A k-mutual exclusion algorithm for ad hoc wireless networks. In: Proceedings of the First Annual Work Shop on Principles of Mobile Computing (POMC 2001) (2001)
48. Kanrar, S., Choudhury, S., Chaki, N.: A link-failure resilient token based mutual exclusion algorithm for directed graph topology. In: Proceedings of the 7th International Symposium on Parallel and Distributed Computing—ISPDC 2008 (2008)
49. Kanrar, S., Chaki, N., Chattopadhyay, S.: A new link failure resilient priority based fair mutual exclusion algorithm for distributed systems. *J. Network. Syst. Manage. (JONS)* 21(1), 1–24 (2013). ISSN 1064-7570
50. Sil, S., Das, S.: An energy efficient algorithm for distributed mutual exclusion in mobile Ad-hoc networks. *World. Acad. Sci. Eng. Technol.* 64, 517–522 (2010)
51. Panghal, K., Rana, M.K., Kumar, P.: Minimum-process synchronous check pointing in mobile distributed systems. *Int. J. Comput. Appl.* 17(4), 1–4 (2011)
52. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. *J. ACM* 43(4), 685–722 (1996)
53. Chen, W., Lin, S., Lian, Q., Zhang, Z.: Sigma: A fault-tolerant mutual exclusion algorithm in dynamic distributed systems subject to process crashes and memory losses. Microsoft Research Technical Report MSR-TR-2005-58 (2005)
54. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kouznetsov, P.: Mutual exclusion in asynchronous systems with failure detectors. Technical Report in Computer and Communication Sciences, id: 200227 (2002)
55. Reddy, R.L.N., Gupta, B., Srimani, P.K.: A new fault tolerant distributed mutual exclusion algorithm. In: ACM/SIGAPP Symposium on Applied computing: Technological Challenges of the 1990's, pp. 831–839 (1992)
56. Bouabdallah, A., Koenig, J.C.: An improvement of Maekawa's mutual exclusion algorithm to make it fault-tolerant. *Parallel Process. Let.* 02n03(2), 283–290 (1992)

57. Chang, Y.I., Chang, Y.J.: A fault-tolerant dynamic triangular mesh protocol for distributed mutual exclusion. *ACM SIGOPS Operating Syst. Rev.* **31**(3), 29–44 (1997)
58. Barbara, D., Garcia-Molina, H.: Mutual exclusion in partitioned distributed systems. *Distrib. Comput.* **1**(2), 119–132 (1986)
59. Lotem, E.Y., Keidar, I., Dolev, D.: Dynamic voting for consistent primary components. In: 16th ACM Symposium on Principles of Distributed Computing (PODC), pp. 63–71 (1997)
60. Lim, J.B., Chung, K.S., Chin, S.H., Yu, H.C.: A gossip-based mutual exclusion algorithm for cloud environments. In: *Advances in Grid and Pervasive Computing. Lecture Notes in Computer Science*, vol. 7296, pp. 31–45 (2012)
61. Edmondson, J., Schmidt, D., Gokhale, A.: QoS-enabled distributed mutual exclusion in public clouds. In: *On the Move to Meaningful Internet Systems: OTM 2011. Lecture Notes in Computer Science*, vol. 7045, pp. 542–559 (2011)
62. Chang, Y.I.: A hybrid distributed mutual exclusion algorithm. *Microproces. Microprogram.* **41**(10), 715–731 (1996)
63. Vidyasankar, K.: A highly concurrent group mutual L-execution algorithm. *Parallel Process. Let.* **16**(04), 467–483 (2006)
64. Atreya, R., Mittal, N., Peri, S.: A quorum-based group mutual exclusion algorithm for a distributed system with dynamic group set. *IEEE Trans. Parallel Distrib. Syst.* **18**(10), 1345–1360 (2007)
65. Vidyasankar, K.: A simple group mutual l-exclusion algorithm. *Inf. Process. Let.* **85**(2), 79–85 (2003)
66. Atreya, R., Mittal, N.: A distributed group mutual exclusion algorithm using surrogate-quorums. In: 25th IEEE International Conference on Distributed Computing Systems, pp. 251–260 (2005)
67. Jayanti, P., Petrovic, S., Tan, K.: Fair group mutual exclusion. In: Twenty-Second Annual Symposium on Principles of Distributed Computing, pp. 275–284 (2003)
68. Takamura, M., Altman, T., Igarashi, Y.: Speedup of Vidyasankar’s algorithm for the group k-exclusion problem. *Inf. Process. Let.* **91**(2), 85–91 (2004)
69. Singhal, M.: A class of deadlock-free Maekawa-type algorithms for mutual exclusion in distributed systems. *Distrib. Comput.* **4**(3), 131–138 (1991)
70. Li, M.A., Liu, X.S., Wang, Z.: A high performance distributed mutual exclusion algorithm based on relaxed cyclic difference set. *Dianzi Xuebao (Acta Electron. Sinica)* **35**(1), 58–63 (2007)
71. Burns, J.E., Jackson, P., Lynch, N.A., Fischer, M.J., Peterson, G.L.: Data requirements for implementation of n-process mutual exclusion using a single shared variable. *J. ACM (JACM)* **29**(1), 183–205 (1982)
72. Huang, S.T., Jiang, J.R., Kuo, Y.C.: K-coterie for fault-tolerant k entries to a critical section. In: *ICDCS*, pp. 74–81 (1993)
73. Joung, Y.J.: Quorum-based algorithms for group mutual exclusion. In: *Distributed Computing*, pp. 16–32 (2001)
74. Hardekopf, B., Kwiat, K., Upadhyaya, S.: A decentralized voting algorithm for increasing dependability in distributed systems. Join MEETING of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th International Conference on Information System Analysis and Synthesis (ISAS 2001) (2001)
75. Thomas, T.H.: A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans.Database Syst.* **4**(2), 180–209 (1979)
76. Ahmad, M., Ammar, M.H., Cheung, S.Y.: Multi-dimensional voting: A general method for implementing synchronisation in distributed systems. In: *Proceedings 10th International Conference on Distributed Computer Systems*, pp. 362–369 (1990)
77. Barbara, D., Garcia-Molina, H., Spauster, A.: Increasing availability under mutual exclusion constraints with dynamic vote reassignment. *ACM Trans. Comput. Syst.* **7**(4), 394–428 (1989)

78. Rabinowich, M., Lazowska, E.D.: Improving fault-tolerance and supporting partial writes in structured coterie protocols for replicated objects. In: *Proceedings ACM SIGMOD*, pp. 226–235 (1992)
79. Adam, N.R.: New dynamic voting algorithm for distributed database systems. *IEEE Trans. Knowl. Data Eng. Arch.* **6**(3), 470–478 (1994)
80. Xu, L., Bruck, J.: Deterministic voting in distributed systems using error-correcting codes. *IEEE Trans. Parallel Distrib. Syst.* **9**(8), 813–824 (1998)
81. Hardekopf, B., Kwiat, K., Upadhyaya, S.: Secure and fault-tolerant voting in distributed systems. In: *IEEE Aerospace Conference* (2001)
82. Paris, J.F., Long, D.D.E.: Efficient dynamic voting algorithms. In: *Proceedings of the Fourth International Conference on Data Engineering*, pp. 268–275 (1998)
83. Paris, J.F., Long, D.D.E.: A realistic evaluation of optimistic dynamic voting. In: *Proceeding of Seventh Symposium on Reliable Distributed Systems*, pp. 129–137 (1988)
84. Jajodia, S., Mutchler, D.: Dynamic voting algorithms for maintaining the consistency of a replicated data. *ACM Trans. Database Syst.* **15**(2), 230–280 (1990)
85. Agrawal, D., El Abbadi, A.: An efficient and fault tolerant solution for distributed mutual exclusion. *ACM Trans. Comput. Syst.* **9**(1), 1–20 (1991)
86. Zarafshan, F., Karimi, A., Al-Haddad, S.A.R., Saripan, M.I., Subramaniam, S.: A preliminary study on ancestral voting algorithm for availability improvement of mutual exclusion in partitioned distributed systems. In: *Proceedings of International Conference on Computers and Computing (ICCC'11)*, pp. 61–69 (2011)
87. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. *J. Assoc. Comput. Mach.* **32**(4), 841–860 (1985)
88. Tong, Z., Kain, R.Y.: Vote assignments in weighted voting mechanisms. In: *Seventh Symposium on Reliable Distributed Systems*, pp. 138–143 (1988)

Concurrency Control in Distributed System Using Mutual
Exclusion

Kanrar, S.; Chaki, N.; Chattopadhyay, S.

2018, X, 95 p. 43 illus., Hardcover

ISBN: 978-981-10-5558-4