A Study of Token Based Algorithms for Distributed Mutual Exclusion

Abhishek Swaroop*  
Department of Computer Science & Engg  
G. P. M. College of Engineering  
New Delhi 110036 India  
abhi_pu1@yahoo.co.in  

Awadhesh Kumar Singh  
Department of Computer Engineering  
National Institute of Technology  
Kurukshetra 136119 India  
aksinreck@rediffmail.com

Abstract

The selection of a ‘good’ mutual exclusion algorithm, for the design of distributed systems, is of great importance. A number of mutual exclusion algorithms, with different techniques and varying performance characteristics, are available in the literature. These algorithms can be broadly classified into token based algorithms and non-token based algorithms. A number of survey papers for non-token based mutual exclusion algorithms exist. Although, some of them include discussion on token based mutual exclusion algorithms too, however, none of them include any discussion on the newer variants of classic mutual exclusion, like $k$-mutual exclusion and group mutual exclusion. The paper presents an exhaustive survey of the token based mutual exclusion algorithms. The variants of mutual exclusion problem, namely $k$-mutual exclusion and group mutual exclusion, have also been covered.

Keywords: Distributed mutual exclusion; Token; Algorithm; $k$-mutual exclusion; Group mutual exclusion

1. Introduction

A distributed system consists of geographically dispersed independent computers, connected via a message passing network, collaborating on an application. Due to lack of common memory and global clock, handling such system is harder than any other system. The major advantage of distributed systems is resource sharing. However, special care has to be taken, while allocating shared resources to processes, in order to ensure the correct functioning of the system. The history of resource allocation problem can be traced back to 1965 when Dijkstra [11] introduced the mutual exclusion problem. Since then a number of variants of the resource allocation problem have also emerged, like drinking philosopher [22], $k$-exclusion [26], group mutual exclusion [56], and group $k$-mutual exclusion [27], to name a few.

* Corresponding author, Mailing address 360/7A, Gali No. 6 Bhola Nath Nagar, Shahadra, Delhi – 110032, India  
Tel: -91-11-22300003  
Email: - abhi_pu1@yahoo.co.in

Usually, a shared resource is required to be accessed by only one process at a time. Such resource must be accessed inside a critical section (CS) in a way that at any particular time only one of the contending processes is allowed to be inside of its critical section. This is called mutual exclusion. The mutual exclusion is very crucial for the design of distributed systems. For example, in case of a replicated database maintaining consistency of the data is a major concern. Thus, if a node wants to perform updates, it must ensure that no other node is involved in updating the database. Any mutual exclusion algorithm should satisfy following requirements.

(A) Mutual exclusion – at any point of time no two conflicting processes must be in their critical section simultaneously.

(B) Freedom from Starvation – a process should not be forced to wait indefinitely to execute CS while other processes are repeatedly executing CS.
(C) Deadlock Freedom – two or more processes should not endlessly wait for messages that will never arrive.

(D) Fairness – enforces that the requests must be executed in the order these are made.

The mutual exclusion algorithms can be broadly classified into two categories.

(A) Shared memory based algorithms

(B) Message passing based algorithms

The distributed shared memory is an abstraction used for sharing data between computers that do not share physical memory. The mutual exclusion algorithms, which use the shared memory approach, are namely, Anderson-Kim [17], Attiya-Bortnikov [14], Lamport [29], Burns [18], Lynch-Shavit [40], Styer [10], Lycklama-Hadzicoloș [9]. These algorithms access and update some common variables for solving mutual exclusion problem.

In message passing system, processes can communicate with each other only by exchanging messages over a communication network. The message passing algorithms can be further classified into two categories.

(A) Non-Token Based Algorithms

(B) Token Based Algorithms

In non-token based algorithms, a process wishing to enter CS sends its request to some or all other processes in the system and waits for their permission. The concept of logical clocks [28] is used for prioritizing requests. The non-token based algorithms can be put in to two categories voting based algorithms and coterie based algorithms. In voting based algorithms, a node can enter its CS only if it has obtained permission from a number of nodes whose total number of votes is at least majority of the total votes. Some examples of voting based algorithms are Gifford [7], Kumar [1], Thomas [53], Barbara et al. [6], Jajodia-Multcher [49]. A coterie [15] C , on a non empty set of nodes U , is a collection of non empty subsets, called quorums, of U satisfying following conditions.

(i) Minimality condition: 
\[ S, T \in C \Rightarrow S \subseteq T, T \subseteq S \]

(ii) intersection property: 
\[ S \cap T \neq \Phi \]
\[ \forall S, T \in C \]

A node wishing to enter CS must obtain permission from all members of some quorum in the assigned coterie. Some examples of coterie based algorithms are Garcia-Molina [15], Maekawa [31], Singhal [37], Agrawal-Abbadi [3].

In token based algorithms, a privilege message called token is shared among various processes sharing a resource. Only the process possessing the token can enter CS. A process wishing to enter CS and not possessing token sends its request to a set of processes and waits for the token. The focus of present paper is the token based algorithms for distributed mutual exclusion. Algorithms for k-mutual exclusion and group mutual exclusion have also been discussed. The rest of the paper is organized as follows. In section 2 we discuss performance metrics relevant for mutual exclusion algorithms in message passing systems. Section 3 contains discussion of token based algorithms for mutual exclusion. Section 4 and 5 cover k-mutual exclusion and group mutual exclusion respectively. Section 6 presents concluding remarks.

2. Performance metrics
The following performance metrics are generally used for measuring performance of message passing based mutual exclusion algorithms.
(i) Response time - is the time interval a request waits for its CS execution to be over after its request messages have been sent out.

(ii) Synchronization delay - is the time required after a site leaves CS and before the next site enters the CS

(iii) System throughput - is the rate at which the system executes requests for the CS. If synchronization delay is $D$ and average CS execution time is $E$, then the system throughput will be

$$\frac{1}{D+E}$$

(iv) Message complexity - is the average number of messages exchanged by a process per CS entry.

(v) Message size - is the average number of bits required to encode a message required by the algorithm.

3. Token based Algorithms for Mutual Exclusion

In token based algorithms, a privilege message (token) is shared among all the processes in the system. A process can enter CS, only if it is in possession of the privilege message. A process wishing to enter CS sends its request to a set of processes called its request set [55] and waits for the token. The token based algorithms generally use sequence numbers to distinguish between old requests and current request made by the same process. Token based algorithms produce less message traffic in comparison to non-token based algorithms. Because of the existence of a unique token in the system, token based algorithms are deadlock free. But their resiliency to failure is poor, because if the process having token fails or token is lost in transit, a complex process of token regeneration and recovery has to be started. Token based algorithms for mutual exclusion can be classified in to following categories.

(i) Broadcast based algorithms

(ii) Logical structure based algorithms

(iii) Centralized token based algorithms

3.1 Broadcast Based Algorithms

In broadcast based algorithms, a site requesting to enter CS sends a request message to the sites in its request set in parallel and waits for the privilege message. Upon receiving the privilege message, the site can enter CS. In the following subsections, we will discuss some broadcast based algorithms for distributed mutual exclusion.

3.1.1 Suzuki-Kasmi algorithm

Suzuki-Kasmi [16] proposed a mutual exclusion algorithm in which a request message is sent to all other sites by the requesting site. The requesting site waits for the privilege message before entering CS. Each site $i$ maintains an array $RN$ of size $N$, which stores the latest sequence number of requests made by every site, known to the site. The privilege message maintains a queue for storing pending requests and an array $LN$ of size $N$, for storing sequence number of last request served for each site. When a site $i$ finishes its CS, all sites $j$, for which $RN_j = LN_j + 1$, are appended to the token queue. The site at the head of the queue is removed from the queue and the privilege message is sent to that site. If the token queue is empty the site $i$ holds the token, until the arrival of request messages.

The algorithm requires at most $N$ messages. The synchronization delay is one way trip communication time ($T$). This is an improvement over Ricart-Agarwala’s algorithm [13], which has message complexity of $2(N-1)$ and the
synchronization delay equal to two way trip communication time (2T). The draw back of the Suzuki-Kasmi’s algorithm is that in it sequence numbers are not bounded. To remove this drawback a modified algorithm was presented by Suzuki-Kasmi. However, in modified algorithm the number of messages increases to \(L \cdot N + (N - 1)\) for \(L\) mutual exclusion invocation by a single node or \(N + (N - 1) / L\) messages per invocation.

Nishio et al. [50] proposed a fault tolerant extension of Suzuki-Kasmi’s algorithm [16]. The algorithm differs from Suzuki-Kasmi’s algorithm in the way next site \(j\) is selected, when a site \(i\) exits from CS. The token is sent to a site \(j (j \neq i)\) such that the value of \((j - i) \mod N\) is minimum, among those sites having pending requests. Nishio et al.’s algorithm does not require a queue to store pending requests. The relationship between array of sequence numbers at site and array of sequence numbers at token is used to determine whether a site is having pending request or not.

The additional data structures maintained by the algorithm are site-age \([N]\) stored at each site and token-age stored in the token. When a site \(i\) suspects token loss, it executes the algorithm regenerate, which determines candidate value propose-age, for the token age and sends a token-missing message to all sites \(j(\neq i)\). Site \(j\) checks if \(\text{propose-age} > \text{site-age}[j]\), \(\text{site-age}[j]\) is updated and ACK message is sent to \(i\) otherwise, NACK message is sent to \(i\). The site proposing highest age regenerates the token, on receiving positive response from all other sites.

A time out mechanism is used to detect the communication link failures or site failures. A requesting site waits for token up to a specified time, after that failure recovery algorithm is initiated. A two phase recovery scheme has been suggested to recover from situations, in which the information stored at sites or token may be lost, due to some catastrophic failure. The algorithm is able in handling a variety of network failures and ensures mutual exclusion as long as there is one operational site in the network.

### 3.1.2 Heuristic approach

The major drawback of Suzuki-Kasmi’s algorithm [16] is that a site wishing to enter CS and not possessing token has to send the request message to all other sites in the system. In Suzuki-Kasmi’s algorithm request set of every site is static and its cardinality is always \(N - 1\).

M. Singhal [36] used heuristic approach in his algorithm and showed that the number of messages required per CS invocation can be reduced by dynamically changing the request set. The algorithm makes use of the state information of the sites in changing the request set. A site may be in any one of the following states.

- \(R\) – Requesting.
- \(N\) – Not Requesting
- \(E\) – Executing.
- \(H\) – Holding.

Sequence numbers are used to differentiate between current and old requests. Each site maintains state vector \((SV)\) and sequence number vector \((SN)\). The token maintains two vectors \(TSV\) and \(TSN\), which are used for storing state information of the sites and sequence number of the last served request for each site respectively. The token and the request messages are used to disseminate state information among the sites.
Initially site 1 holds the token and \( SV_1[1] = H \). Site \( S_i \) \( 1 \leq i \leq n \) thinks that sites \( S_n, S_{n-1}, \ldots S_{i+1} \) are not requesting the token and sites \( S_{i-1}, \ldots S_1 \) are requesting the token and one of these has the token. When a site wants to enter CS, it use some heuristic to guess a set of sites which are likely to have the token and sends its request message only to those sites. The heuristic used in the algorithm is “All the sites for which state vector entries are \( R \) are added in the request set”. When a site exits from CS, it updates its state vectors and token vectors. For this purpose the state vectors at site are compared with the token state vector to determine, which vector has more current information about the state of sites. The outdated information is replaced with more current information. An arbitration rule is used to determine, which of the many requesting sites should get the token next. The fairness of the algorithm depends upon the arbitration rule. Following two arbitration rules were suggested in the paper.

(i) Grant the token to a requesting site with lowest sequence no.
(ii) Grant the token to the requesting site nearest to the current site.

The simulation study shows that, under light load the algorithms performs better than Suzuki-Kasmi algorithm as far as number of messages per CS is concerned. However under heavy load the performance of both the algorithms is comparable. Under light load the mean delay in granting CS will be equal to one round trip time (\( 2T \)). Under heavy load after every \( T + E \) units of time a site will get the token and execute its CS.

Singhal’s algorithm uses less number of messages as compared to Suzuki-Kasmi algorithm under light load conditions, but the fairness of the algorithm depends upon the arbitration rule used in selecting the next site to which token has to be sent. The space requirement of singhal’s algorithm is also quiet large, because array containing state information has to be stored at each site. The token also contains an array to store information about the state of the sites.

Chang et al. [55] presented another token based algorithm, in which the request set of a site changes dynamically. The algorithm does not use state information, therefore, the arrays for storing state information are not required. In this algorithm token maintains a FIFO queue storing waiting requests. Each site maintains an array of sequence numbers and a request set \( R \). The requesting site sends request message to only those sites which are in its request set \( R \). Initially all other sites are added in request set of a site, except the site \( X \), which is randomly chosen to hold the token and for which request set is empty. When a site exits from CS, it checks if new requests have arrived and add these requests to the token queue. The site then removes the site at the head of the token queue and sends the token to that site. When a site gets a token, its request set is emptied and all sites, which are in token queue, are added to its request set. Simulation studies performed, shows that only \( 0.6N \) messages/CS are required under light load, however under heavy load algorithm shows same performance as Suzuki-Kasmi’s algorithm.

### 3.1.3 Other Algorithms

Mizuno et al. [32] used a data structure, called quorum agreements [5] similar to coterie, to present a variant of Suzuki – Kasmi’s algorithm[16]. Let \( QA = (Q, Q^{-1}) \) be the quorum agreement used by the
algorithm. Each node $i$ has a request set $R_i (R_i \in Q)$ and an acquired set $A_i (A_i \in Q^{-1})$ and an array of sequence numbers $RN_i$. The token carries an array of sequence numbers of the last served request of each site and a queue of pending requests. Initial request messages are issued by $i$ to the nodes in $R_i$ only. When a node $j$ receives privilege message it sends acquired message to all nodes in its acquired set $A_i$. The privilege message contains a copy of array $LN$ which contains the sequence numbers of last served requests of each node. When $k$ receives an acquired message from $j$ and $RN[K] > LN[K]$, it sends back a request message to node $j$. Otherwise node $k$ waits until it receives new requests. The number of messages used by node $i$ for one CS entry are between $(|R_i| - 1) + (|A_i| - 1) + 1)$ and $(|R_i| - 1) + 2* (|A_i| - 1) + 1)$. Hence the number of messages required, depends upon the quorum agreements. If good quorum agreements are chosen algorithm performs nicely. Heavy load synchronization delay of the algorithm is $T$, while light load synchronization delay is $3T$ due to extra level of indirection in finding the token. However the performance of the algorithm is highly dependent on the selection of the quorum agreements.

Yan et al. [59] emphasized the importance of network topology on the design and performance of mutual exclusion algorithms and suggested that a mutual exclusion algorithm should have a strong adaptability to different network topologies and when a message has passed by $N-1$ intermediate nodes, the message complexity should be considered $N$ instead of 1. Yan et al.’s [59] algorithm makes use of dynamic state information and network topology. A unique token is used to allow entry in CS. Each processor dynamically updates the latest known location of the token. A requesting process only sends out a request message to chase token along the latest known location of the token. A request message dynamically changes its path based on the local information of the intermediate nodes. The message complexity of the algorithm is $O(N)$.

Saxena et al. [42] proposed a token based algorithm for arbitrary network topologies using broadcast based approach. A site can be in any one of the following states $R$ – Requesting, $N$ – Not Requesting, $E$ – Executing, $H$ – Holding. In this algorithm the token serves the requests of the sites which fall on the route to its destination. Although the algorithm does not satisfy request in strictly FCFS order, however under heavy load conditions this enroot servicing reduces the message complexity as well as delay considerably. Another technique used in this algorithm is, using special messages know as $tlps$ (token location propagators). When a token leaves for a requesting site this information is spread in the network using $tlps$. Under light load this technique is quiet effective in reducing the message complexity.

However, it is very difficult to collect and maintain the information about the network topology of a distributed system. Now a days in distributed systems the addition and deletion of sites is quiet frequent. Besides that due to mobile users the topology of the system may also change frequently because of the change of location by a user. Therefore the applications of the algorithms, which use information about network topology, are quiet limited.

### 3.2 Logical Structure Based Algorithms
The logical structure based algorithms impose a logical structure (such as tree) on the sites of the system. The request of a site wishing to enter CS follows a path between requesting site and the site holding token. The token also follows the path from the site holding the token to the requesting site. The logical structure based algorithms have an average lower message cost as compared to broadcast based algorithms, presents better scalability, and uses simple data structures. However logical structure based algorithms are very sensitive to site failure, because these can not tolerate with a single failure in the token request path.

3.2.1 Raymond’s static Tree based Algorithm

Raymond’s algorithm[25] assumes that the sites in the system being arranged in an unrooted tree structure. In it all the messages are sent along the undirected edges of this tree. There is a privilege message and the site having privilege message is allowed to enter its CS. Each site maintains a variable holder, which indicates the location of the privilege message relative to the node.

When a non-privileged site wishes to enter CS it sends a request to its holder site. If the holder site is also unprivileged, it forwards request to its holder. Thus a series of request messages travels along the path between requesting site and the privileged site. When privilege message is no longer required by the site, it sends it to one of its neighboring sites, which has requested the privilege. The holder is modified accordingly and privilege message is forwarded along the path. When site wishing to enter CS receives privilege, it sets holder = self and enters CS. A request queue (FIFO) is maintained at each site, which holds the names of those neighbors that have sent a request message to the site, but have not received the privilege yet. When a site itself wishes to enter in CS, self is placed in the queue of that site.

The algorithm does not require sequence numbers. The upper bound for the number of messages /CS is $2D^2$, where $D$ is the diameter (longest path length) of the tree. The worst topology is a straight line with a diameter of $N-1$ and message complexity of $2(N-1)$. The best topology is a radiating star with message complexity of $O(kn)$, where $k$ is the valence of each non-leaf node. Two modifications, based on piggyback strategy and greedy strategy, were also suggested, which reduce the number of messages required.

3.2.2 Naimi et al.’s Dynamic tree based algorithm

The structure of logical tree used in Raymond’s algorithm was static and only the direction of edges could be changed. This makes the algorithm highly dependent on the topology of the tree. Naimi, Trehel and Arnold [35] proposed an algorithm based on a dynamic rooted tree. In this algorithm every process maintains two pointers father and next. The father indicates the process to which the request for CS access should be forwarded and the next indicates the node to which access permission should be forwarded. This algorithm uses two distributed data structures.

(i) Queue- The head of the queue possesses the token and tail of the queue is the last process that has requested CS. Each process knows the next process in the queue, if its next exists.

(ii) Logical rooted tree- which gives the path to go to the tail of the queue.

When a process wishes to enter its CS, it sends its request to its father. From father to father a request is transmitted to the root, which has father =
This root is also the tail of the queue. The requesting process transmits its request to father and regards itself as the new root. Therefore, there may be several rooted trees when messages are in transit. However, when all the messages are arrived these are grouped to form a single rooted tree. When the request of a process arrives at the tail and if the tail is waiting for the token, the requesting process is linked to the tail. If the tail has the token and is in its CS, the requesting process is linked to the tail. Otherwise the token is transferred to the requesting process. When a process leaves the CS, it gives the token to the next process in the queue. If no such next exists, the process will hold the token. The algorithm does not use logical clocks or sequence numbers to serialize the concurrent events, all the variables are bounded and only $O(\log n)$ messages/CS requests are used.

Julien Sopena et al. [21] proposed a fault tolerant extension for the Naimi-Trehel’s algorithm[35]. The algorithm tries to reconstruct the next queue by gathering intact portions of previous next queue, which existed just before the failure. To maintain information about predecessors, whenever $S_i$ updates its next variable, it sends a commit message to the requester $S_i$. The commit message contains following information.

(i) The $k$ Predecessors of $S_i$ (k can be configured).

(ii) $S_i$’s Position in the queue ($S_i$’s closest predecessor’s position + 1)

After receiving a commit message $S_i$ periodically checks the liveness of its closest predecessor. Once a failure is detected by $S_i$ and $S_i$ has received the commit message, it sends a message to its predecessors from closest to farthest. $S_i$ Stops when it receives an answer from one of its predecessor site.

That site considers $S_i$ as its new successor. If none of the predecessors answers, $S_i$ broadcasts a search-prev message to all sites. All sites having smaller positions in the queue answers and $S_i$ chooses its closest predecessor and connect itself to that site by sending a message. If $S_i$ does not receive any answer in a specified time it regenerate the token and initialize its position to zero in the queue. However, if $S_i$ has not received commit message, an election algorithm has to be executed and only the elected site is allowed to continue the recovery process.

Only one commit message /CS request is added in case of no failure, hence the message complexity remains $O(\log n)$. In case of failure the algorithm requires less number of messages and less time in comparison to Naimi-Trehel’s fault tolerant algorithm [33]. The algorithm can tolerate at most $N-1$ site failures. The Next queue is rebuilt from the previous queue; therefore original ordering of requests is preserved up to an extent.

Bertier et al. [30] proposed a hierarchal token based mutual exclusion algorithm based on Naimi-Trehal algorithm [35], which takes into account the latency gap between local and remote machines. To reduce the number of inter cluster messages following three techniques were proposed.

(i) Per Cluster Proxy- In each cluster, there exists a dedicated process called proxy, which stores the last request to remote clusters. When node $i$ requests for the token, which it assumes to belong to a remote cluster, sends its request to the proxy. If another node $j$, in the same cluster has recently requested for the token and proxy is aware of it, it redirects $i$’s request towards $j$, thus avoiding messages to remote clusters.
(ii) Aggregation – In it when a request has to be sent to the probable token holder, belonging to a remote cluster; the request is not sent but stored in a queue. It is stored in a queue in the last node which will enter the critical section with in the cluster, which is called “Local Root”.

(iii) Token Preemption -In token preemption a high priority is given to requests originating from the local cluster to exploit locality. A threshold is defined, in order to avoid starvation. Whenever the number of local requests is below this threshold; the request path is modified, in order to serve local requests first.

The variant of Naimi-Trehel’s algorithm [35] using above mentioned techniques were presented by Bertier et. al. and it was observed that the above mentioned techniques are quiet useful in reducing the inter cluster messages.

3.2.3 Other Algorithms

In Agrawal-Abbadi’s [4] algorithm each site maintains a pointer owner which points towards possible owner of the token. When a site \(S\) wants the token, it sends request to its owner, if owner has the token and not in CS it sends the token to \(S\) and if it does not have the token, it forwards the request to its owner. If the requesting site does not receive the token within a specified time, its request is aborted.

The fault tolerant version of the above algorithm uses the concept of logical time. Each site \(S\) maintains the following information. owner, - A pointer to site which is possible owner of token. ownertime, - is the logical time, when \(S\) last sent the token to owner, token state, - captures the state of shared resource associated with the token, when it was last owned by the sites \(S\). The algorithm does not require separate election or recovery protocol to recover lost token, instead token recovery is integrated in the protocol itself. The concept of logical time is used to detect and recover from token loss. The algorithm can handle message loss, site failures and network partitioning.

Self-Stabilization is the most fundamental concept of automatic recovery from transient faults in distributed systems. Jun Kiniwa [19] presented a self stabilizing token passing algorithm in which a token is passed via a dynamic BFS tree rooted at a requesting process. No queues are used in this algorithm and every variable is bounded. The stabilization time of the algorithm is \(1+5*D\) rounds, where \(D\) is the diameter of the network. \(k\) Covering time of the algorithm is \(k*n\) round, where \(n\) is the number of processors in the system.

3.3 Centralized Token Based Algorithms

Centralized algorithms are those, in which there is a central coordinator node. Every requesting node requires the permission of the central coordinator to execute its CS. Number of messages required in centralized algorithms is very low as compared to other techniques. However, in centralized algorithms central coordinator is generally overloaded and becomes the performance bottleneck. There is a single point of failure in these algorithms; if central coordinator fails, whole system will fail. Another disadvantage of central algorithms is that two sequential messages (release and privilege) are required to pass the lock from one process to another. The developers of centralized token based algorithms, tried to combine the advantages of centralized algorithms and token-based algorithms.

In Felten- Rabinovich’s centralized token based algorithm [12], a node is assigned as master
node, while other nodes are clients. The master node maintains an active token queue and a passive token queue. A client wishing to enter CS sends the request message to the master. The request is stored in the passive token queue. A client node can enter its CS upon receiving the token-grant message. The token-grant message also contains the active token queue. The node receiving token, removes its id from the head of the queue and rest of the queue is stored in a local queue after-me. On exiting from CS, if the node’s after-me queue is non empty, the node sends token-grant message, directly to the first node in the after-me queue.

If the length of active token queue is less than a predefined length (warning length), the master sends a forward-token message containing a list of nodes in the passive token queue, to the node at the tail of the active token queue. Then the master node moves nodes from its passive token queue into its active token queue. The master node labels a batch number on each forward-token and token-grant messages, so that previous requests may be distinguished from the current requests. The batch number is passed along with the token and each client remembers the highest batch number it has seen. The light load synchronization delay of the algorithm is $T^2$ and number of messages/CS under light load is 3. The heavy load synchronization delay of the algorithm is $T$. Wu-Shu argued that in centralized algorithms, only the coordinator node has to do some extra work, however in decentralized algorithms; every node is swamped with extra work and communication traffic.

4. $k$-Mutual Exclusion
Raymond [26] introduced $k$-mutual exclusion problem, as a variant of mutual exclusion problem, in which at most $k$ processes ($1 < k \leq n$) can enter critical section at a time. For example there may be $k$ copies of licenses of software, therefore it can be used by only $k$ users at a time. Raymond also presented a non-token based algorithm for $k$-mutual exclusion based on Ricart-Agarwala’s algorithm [13], which requires $2(n-1)$ messages per CS invocation.

Srimani-Reddy [44] presented an extension of Suzuki-kasmi’s algorithm for $k$-mutual exclusion problem. In this algorithm $k$ tokens are circulated. Because of existence of $k$ tokens in the system up to $k$ processes may be inside their critical section
simultaneously. If a node owns a token, it may enter critical section directly, otherwise it sends request message to all other $N-1$ nodes and waits for a token. The upper bound for the message complexity of this algorithm is $N + K - 1$ per CS entry.

M. Naimi [34] proposed a directed graph based algorithm for $k$-mutual exclusion problem. Initially a logical spanning tree is defined from an arbitrary network and $k$ tokens are given to the root. A waiting queue is maintained at each node. A node requesting CS sends a request to its predecessor and put itself in the waiting queue. The requesting node then becomes the root and waits for the token. On exiting from CS, the node sends the token to the first node in its waiting queue. On receiving request from its neighbor $Y$, if a free token is available at $X$, it is directly sent to the $Y$, otherwise the node $X$ puts, Node $Y$ in its waiting queue and transmits $Y$’s request to its predecessors except $Y$. The node $Y$ then becomes predecessor of node $X$ and the directed graph is transformed in to another directed graph. The number of message required is between 0 to $2*(n-1)$ messages per critical section entry.

K. Makki et al. [23] used a general semaphore and token queue with token in their algorithm. One site is chosen as ‘good site’. The good site places all token requests, which it receives in its local queue. When good site eventually receives the token, it executes its CS and then appends its local queue in to the token queue. A new good site is chosen and an update message about new good site selection is sent to all those sites, which are not in token queue now. A general semaphore, which is part of the token, indicates the number of critical sections that are available. Each site that receives the token checks the value of this semaphore, if it is non-zero, the value is decremented by 1 and token is passed to next site in the token queue. If value of the semaphore is zero then no change is made to the semaphore and the site receiving token holds the token. When a site $S_i$ exits from its critical section, it sends a release message to the site that was $K_{th}$ in the token queue when $S_i$ had removed itself from the queue. If the good site on the token queue is before the $K_{th}$ site, the release message is sent to the good site. The value of semaphore is incremented by 1, with every release message.

In this algorithm, a cycle is defined to begin, when the good site picks a new good site, sends out update messages. A cycle ends when a new good site finally receives the token and executes its own CS. Let $m$ be the number of token requests in the token queue at the beginning of the cycle then $n/m + 2$ messages per CS request are required. Under light load ($m$ is close to 1) message complexity is $O(n)$. Under heavy load ($m$ is close to $n$) the performance of the algorithm is good and only 3 messages are required in the extreme case, when all sites are requesting to enter CS.

Wang-Lang [52] presented a token-based algorithm for $k$-mutual exclusion problem based on Raymond’s tree based approach. The nodes are assumed to be arranged in a tree structure, whose shape remains static. However the direction of an edge can be changed and multiple edges with mixed directions may exist between nodes. Each node has a token direction bag ($tdb$) to store those neighbors, which are on the outgoing paths leading to the nodes, holding the tokens. A node $j$ can appear several times in a $tdb$, if there are several tokens reachable from $j$. Each node maintains a local variable $token$-count, which indicates the number of free tokens at
the node and a queue, which stores the requests from its neighbors. A root node is chosen randomly to hold K tokens. The directions of edges are selected in a way so that from each node there exists k paths leading to the root. When a node i wants to enter CS and does not hold any token, it sends a request message to each distinct node in tdb, and deletes one occurrence of each distinct node in tdb. If tdb is empty (i has already sent messages on behalf of its neighbors), in that case i simply waits. When node i receives token and is waiting to enter CS, it could send the token to a neighbor whose request is ahead in the queue. It can also use greedy strategy by putting its requests always at the front of the queue. If the queue is empty i retains token and increments token–count by 1. The performance of the algorithm is highly dependent on topology. The algorithm requires at most \(2KD\) messages for a node to enter CS, where \(D\) is the diameter of the tree.

Bulgannawar-Vaidya [47] used dynamic forest structure for each token to forwarded token requests. Each node maintains a pointer array with one entry for each token. These pointers define \(k\) forests corresponding to \(k\) tokens. Each node maintains a FIFO queue. Token also contains a queue containing identifiers of the nodes to which token must be forwarded in FIFO order. For performance comparison, simulation experiments were performed with 3 other algorithms, namely Raymond’s, Makki’s and Srimani’s algorithms. The simulation results shows that proposed algorithm achieves lower delay in entering as well as lower number of messages. However average message size is 1.5 to 2 times larger than other algorithms.

5. Group Mutual Exclusion

The Group Mutual Exclusion(GME) problem introduced by Joung [56], deals with two contradictory issues in distributed systems, namely mutual exclusion and concurrency. Joung modeled GME problem as “Congenial Talking Philosopher Problem” [57]. In this problem, there is a set of \(n\) philosophers. A philosopher spends his time, either in thinking alone or in talking in a group. There is only one meeting room, therefore, only one group can be held at a time. A philosopher interested in a group, can succeed to enter the meeting room, only if the meeting room is empty, or some philosopher interested in the same group is already in the meeting room. Joung gave the example of a CD juke box containing large data objects. When a process needs a data object, the data object is loaded to a cache buffer from the CD juke box. The cache buffer is large enough to store only one data object at a time. The Processes interested in currently loaded data object, are allowed to read concurrently, while a process requiring different data object has to wait, till the requested object is loaded to the cache buffer. The mutual exclusion and readers-writer problem are special cases of Group mutual Exclusion problem. For mutual exclusion, one forum can be allocated to each process, so that only one process can be in the critical section at a time. For readers-writers problem, a common read forum can be used by all processes, while a unique write forum can be assigned to each individual process.

The solutions to GME problem in shared memory model have been proposed by Hadziclos [54],Kean-Moir [45],P. Jayanti et al. [43], S.Petrovic [51]. In message passing system, non-token based algorithms for GME problem have been presented by Manabe et al. [58], M.Toyomura [38], Wu-Joung [24], Attrey-Mittal [46]. Till date very few token
based algorithms for group mutual exclusion problem exist. Some of these algorithms are discussed in the following subsections.

5.1 Cantarell et al.’s algorithm for unidirectional rings

Cantarell et al. [48] presented a token based algorithm for GME problem in unidirectional ring networks. The existence of two layers, application layer and GME layer, is assumed in the system. The application layer sends request-session message to request a session and a release-session message on completion of session. The GME layer grants the application layer to access session by sending a grant-session message to the application layer.

The algorithm uses a token to open and close the session only. The token contains the ids of current session and next session respectively. The first process entering a session \(X\) is called the leader of the current session and it advertises other processes that session \(X\) is open. The token may be in any one of the following states.

1. Closed with No leader (CNL) - Token \((\perp, \perp)\)
2. Open with no request (ONR) - Token \((X, \perp)\)
3. Open with request (OR) - Token \((X, Y) (X \neq \perp, Y \neq \perp \text{ and } X \neq Y)\)
4. Closed with request (CR) - Token \((\perp, Y) (Y \neq \perp)\)
5. Open with no Leader (ONL) - Token \((X, X) (X \neq \perp)\)

When a process \(p\) requesting to enter the critical section for session \(X\), receives the token from its predecessor, it becomes the leader and sends token \((X, \perp)\) to tell that session \(X\) is open (state= ONR). Each process requesting session \(X\) may enter the critical section concurrently. When a process \(q\), requests for a session \(Y (Y \neq X)\) the state of token changes from ONR to OR. When process \(P\), leader of session \(X\) receives the token in OR state, it closes the session and changes token state from OR to CR. When a process, say \(r\) receives the token in CR state, \(r\) can be in critical section. In that case, \(r\) holds the token and releases the token on exiting from the critical section. When process \(P\) (leader of session \(X\)) again receives the token in CR state, it closes session \(X\) and changes the state of the token from CR to ONL, meaning that session \(Y\) is now open. The new leader may not be the first process which changed the value of Token.N from \(\perp\) to \(Y\). The main advantage of the above mentioned algorithm is that it does not require process id and maintains no data structure for implementing any queue. The size of the message is bounded. The worst case message complexity is \(O(n^2)\) per resource request and zero messages in the best case. This algorithm is starvation free. However, in this algorithm the sessions are not opened in a FCFS manner. Because while session \(X\) is open two processes \(p\) and \(q\) requested for session \(Y\) and \(Z\) respectively. It is possible that although process \(p\) made the request before process \(q\), session \(Z\) can be initialized before session \(Y\).

5.2 Mittal-Mohan’s algorithm for non uniform group access

Mittal-Mohan [41] presented a token based algorithm for GME problem, for applications in which a small no of groups are requested more frequently than others. For example in a CD-juke box of 1000 CD’s, 5-10 CD’s may be in high demand (say >50% time), or in multiple readers single writer problem, read requests are more common than write requests. The algorithm is based on Suzuki-Kasmi’s [16] algorithm.
In this algorithm two type of tokens, primary token and secondary tokens are used. At a time there can be only one primary token, while there may be multiple secondary tokens. Each token is associated with a type or group, and can be used to enter request of that type only. A process-holding primary token is allowed to issue secondary tokens to others. The primary token stores the number of secondary tokens issued. Each process maintains an array \(request_i\). The \(request_{i,j}\) contains the sequence number of latest request of process \(P_j\), along with its type, which process \(P_i\) knows. A primary as well as secondary token contains a vector \(fulfilled\), \(fulfilled[j]\) stores the number of requests of \(P_j\), which have been fulfilled so far. If the process \(P_i\) is holding a token and \(request_{i,j} < token_i.fulfilled[j]\), \(P_j\) has pending requests.

If the process wishing to enter CS holds the token of the same type, it enters CS. Otherwise it sends request to all processes. A process holding a secondary token, when learns about conflicting pending requests, it sends release message to all other processes, after coming out of its CS. The process holding primary token, on receiving new request of the same type, issues a secondary token. However, if the request type conflicts with that of token, the request is stored in the token queue. The process holding the primary token after coming out of its CS, selects one type for which pending requests are there and passes primary token to one of the processes having pending request of that particular type. For selecting the type of request to be served next, two criteria’s are used, first the age of the request and second the number of requests of that particular type. The age of pending requests is increased with each new session opened. Group age is calculated by summing the ages of all requests in the group of that type. The group, having maximum group age is selected to be opened next. Age factor is used to remove starvation and summation of ages is done to ensure that groups having higher number of requests get priority. A process on receiving the primary token, can not use it immediately because some secondary tokens may be in use. Therefore, it waits until; it has received all the required release messages from the processes holding secondary tokens.

The message complexity of above algorithm is \((2n-1)\) messages per CS invocation. Synchronization delay of the algorithms is \(T\) and waiting time is \(2T\). The maximum concurrency of algorithm is \(n\), that means all processes can be in their CS simultaneously, if their request type is same. Further request and release messages have a constant size, while the size of token is \(O(n)\).

5.3 other algorithms
David Lin [8] announced a token based algorithm for group mutual exclusion, which provides an upper bound to the number of messages and to the message size. The algorithm is compared with Wu and Joung’s algorithm [24] and Cantarell et al.’s algorithm [48] on the basis of message complexity, message complexity in absence of requests, message size, forum switch complexity and time complexity.

Vidyasankar [27] proposed a natural generalization of Group mutual exclusion problem called group \(k\)-mutual exclusion allowing up to \(k\) forums to be held simultaneously. J.R.Jiang [20] proposed a token based algorithm for the group \(k\)-mutual exclusion algorithm problem in distributed
systems. The proposed algorithm does not use node identifiers for node identification. The delay of the algorithm is $O(n^2)$ and forum switch complexity is $O(n)$.

6. Conclusion
The distributed mutual exclusion has always been inviting research attention since its inception. The token based algorithms are quiet useful and important among the algorithms for solving distributed mutual exclusion problem. The algorithms discussed in the paper shows varying performance characteristics with respect to different performance parameters. The system designer, while choosing a particular algorithm, has to take care about the performance characteristics of the algorithm, requirement of the application and implementation aspect.

In order to limit length of the presentation, we could not include the complete list of token based mutual exclusion algorithms. However, we could accommodate all significant contributions to the said area. For the sake of brevity, we discussed the algorithm concepts rather than the detailed algorithms. The token based group mutual exclusion and $k$-mutual exclusion algorithms have also been covered, although, they are small in number.

References


