A new resource sharing protocol in the light of token-based strategy for distributed system

Ashish Singh Parihar*

Department of Computer Science and Engineering,
National Institute of Technology (NIT),
Arunachal Pradesh, India
and
Department of Computer Science,
KIET Group of Institutions,
Delhi-NCR, Ghaziabad, Uttar Pradesh, India
Email: ashish.phd20@nitap.ac.in
Email: ashish.parihar@kiet.edu
*Corresponding author

Swarnendu Kumar Chakraborty

Department of Computer Science and Engineering,
National Institute of Technology (NIT),
Arunachal Pradesh, India
Email: swarnendu@nitap.ac.in

Abstract: One of the highly researched areas in distributed system is mutual exclusion. To avoid any inconsistent state of system, more than one processes executing on different processors are not allowed to invoke their critical sections simultaneously for the purpose of resource sharing. As a solution to such resource allocation issues, token-based strategy for distributed mutual exclusion algorithms as a prime classification of solutions is one of the most popular and significant ways to handle mutual exclusion in this field. Through this research article, we propose a novel token-based distributed mutual exclusion algorithm. The proposed solution is scalable and has better results in terms of message complexity compared to existing solutions. In this proposed art of work, the numbers of messages exchange per critical section invocation are $3(\lceil \log N \rceil - 1)$, $3\lceil(\lceil \log (N+1) \rceil - 1)/2\rceil$ and $6[\lceil \log (N+1) \rceil + 2(2^{(-\lceil \log (N+1) \rceil)} - 1)]$ in case of light load, medium load and high load situations respectively.

Keywords: distributed system; mutual exclusion; critical section; token-based; resource allocation.

Reference to this paper should be made as follows: Parihar, A.S. and Chakraborty, S.K. (2023) 'A new resource sharing protocol in the light of token-based strategy for distributed system', *Int. J. Computational Science and Engineering*, Vol. 26, No. 1, pp.78–89.

Biographical notes: Ashish Singh Parihar received his Bachelor of Engineering in Computer Science from Oriental Institute of Science and Technology (Madhya Pradesh, India) and also completed his MTech in Computer Science and Engineering from the National Institute of Technology, Arunachal Pradesh, India. He served more than five years in the industry as a Senior Developer. Currently, he is an Assistant Professor in the KIET Group of Institutions, Delhi-NCR, Ghaziabad (UP), India and also pursuing his PhD under the supervision of Dr. Swarnendu Kumar Chakraborty. His research areas are distributed systems, machine learning, blockchain and big data.

Swarnendu Kumar Chakraborty is working as an Assistant Professor in the Department of Computer Science and Engineering at the National Institute of Technology Arunachal Pradesh, Govt. of India. He has more than ten years of teaching experience. His research areas are advanced error control, cryptography and Information Security. He is the author of 25 peer-reviewed publications.

1 Introduction

Distributed system (Tanenbaum and Steen, 2016) is a kind of architecture in which components are on different networked computers and communication between them can be done through message passing from one to another in order to achieve a common goal. Processes within the systems in such a network can only communicate by passing messages and message transmission delay cannot be ignored in such an environment. Accessing shared resources simultaneously by the different processes belonging to different systems is a common scenario in this distributed architecture. Process synchronisation is a kind of mechanism that ensures to use of shared resources effectively in case of a single processor system but such access in a distributed system needs extra attention such that no more than one processes executing on different processors are allowed to invoke their critical sections simultaneously. This is required to avoid such painful scenarios like a race condition, deadlock, starvation and inconsistency in data.

Critical section (CS) is a code segment that is available at the process by which the process can access the shared resources in order to complete its certain operation. For providing access to the shared resource, the process can only execute its CS after ensuring that none of the other processes are executing their CS parallelly. Implementing such a mutual exclusion (Kordestani et al., 2020) to invoke CS for shared resource access in distributed systems by a process, various distributed mutual exclusion (DME) algorithms have been presented and broadly classified into – token-based DME (Parihar and Chakraborty, 2021) and non-token-based DME (Saxena and Rai, 2003). The performance evaluation of these algorithms are measured in terms of the number of messages exchanged per CS invocation by any process, delay in synchronisation between two consecutive execution of CS invocation by the same process, response time and system throughput. Out of many distributed mutual exclusive algorithms available in existence, token-based algorithms are the popular one among all in terms of their significance. In token-based algorithms, a unique token travels in network architecture based on a specific approach (either through token-asking or perceptual movement of token) and the process is permissioned to invoke its corresponding CS once it possesses that token. DME plays a vital role in various real-time applications, such as aircraft operations, unmanned aerial vehicle control, handling of central repositories, live video streaming, etc. Integration of some latest trending technologies, such as machine learning (Gupta and Gupta, 2019; Dai and Wang, 2021; Liu et al., 2021; Daid et al., 2021), artificial intelligence (Soto-Morettini, 2017; Tan, 2020) and blockchain (Parihar et al., 2021) are also an exciting work to be imposed on DME solutions. Hence through this research article, we propose a novel token-based distributed mutual exclusion algorithm. The proposed solution is scalable and has better results in terms of message complexity compared to existing solutions. In this proposed art of work, the total numbers of messages exchanged per critical section invocation are $3(\lceil \log N \rceil - 1)$ in case of light load situation, $3\lceil \log (N + 1) \rceil - 1/2]$ in medium load situation and $6[\lceil \log (N + 1) \rceil + 2(2^{-\lceil \log(N+1) \rceil} - 1]$ in case of high load situation. Overall, the major contributions of this research article are as follows:

- proposing a novel token-based DME algorithm
- verification of presented solution through a mathematical model.
- message complexity analysis and its comparison with the existing model.

The remaining part of this research work is organised like: in Section 2, we present the background study. Section 3 provides the overall system model for this proposed algorithm. Section 4 defines the proposed algorithm for distributed mutual exclusion along with a scenario to explain the working of the algorithm and also, the number of message exchanges per CS invocation by a process has been discussed in this section. In Section 5, the distributed mutual algorithm's properties have been validated on our proposed solution and then we provide a comparative study among various distributed mutual algorithms with this proposed algorithm in Section 6. Lastly, we conclude our discussion with future scope in Section 7.

2 Literature survey

Dijkstra (1965) thoroughly studied the problem of mutual exclusion and associated solutions were provided consequently. Later in time, various popular algorithms had been proposed in order to handle mutual exclusion problems in distributed systems based on the logical clock concept by Lamport (1978), token-based (Parihar and Chakraborty, 2021) and non-token-based (Saxena and Rai, 2003). Apart from these solutions, many other algorithms had been proposed to avoid concurrent access of shared resource by the processes in a distributed network which are discussed in this section.

Nishio et al. (1990) proposed an algorithm to solve distributed mutual exclusion which is basically an extended version of Suzuki-Kasami (1985) in terms of handling the case where token lost in network during the transaction of messages. To implement this scenario, they classified the failure in terms of processor, communication controller and communication link and also introduced a time out associated with these. In case of failure, the algorithm identifies it and regenerates the token in the network.

Helary et al. (1994) provided their token-based solution based on a tree topology for taking care of mutual exclusion. In this, a logical rooted tree is proposed for the distributed architecture in which there are various nodes that exist (represents processes). Every node in the logical rooted tree has native variables that describe the behaviour and state of the node in the logical rooted tree. CS asking node transferred its request to the neighbour's parent and waited. Finally, the parent node takes responsibility and sends the token towards its corresponding successor.

Wu and Joung (2000) introduce the concept of group mutual exclusion (GME) in distributed architecture in an asynchronous manner. To handle GME, their research works assume the network to be implemented through ring topology. Their approach is based on a model named as congenial talking philosopher in which there are a number of philosophers in a thinking mode initially, then they wait for their turn and finally talk. For the same approach as above, they introduce a variable more likely similar to Lamport (1978) solution to maintain the sequence numbers. This sequence number is getting increases as per the increasing number of CS demanding processes. Once this sequence number is at maximum size, then all processes are allowed to enter in CS asynchronously.

Lodha and Kshemkalyani (2000) present their research work on the underlying network of Ricart-Agrawala (1981). Instead of accepting the CS demanding request asynchronously in the request queue, the proposed solution takes the requests based on the priorities which are defined at the time of demanding for CS by any process. CS requests are granted to processes by the solution based on their decreasing order of priorities.

Keane and Moir (2001) identified a few drawbacks in Wu and Joung (2000) and tried to overcome them by proposing a GME algorithm by spin locally through cache coherent and non-uniform memory access systems. In their solution, the transfer of control switches from session to session. It uses a few variables which are basically consisting of the number of processes in CS and the current session holder. This overcomes the drawback wherein the process continuously reads the other process's flag variable which reduces the performance of the system.

Cao and Singhal (2001) present a delay quorum-based mutual exclusion solution with an optimal delay where they consider the delay in synchronisation to be minimal and the message exchange complexity is in logarithmic form. In this, each site is associated with some quorum and enters into its CS after blocking all other sites by sending request messages in the same quorum set. The intersection property in between two quorums also ensures that no process will enter into their CS simultaneously.

Cantarell (2005) introduces a GME algorithm in the token ring architecture. It proposes several algorithms in that research work to obtain the same. According to their algorithms, they provided the solution for GME with the help of the bounded size of messages during communication. As per their one of the algorithms, they do not maintain any request queue instead of that they provided the dependency of processes space requirement on the number of actually shared resources as m that incorporates the message size is $2\lceil \log m + 1 \rceil$. This algorithm maintains a relationship between the number of processes and shared resources with each other.

Zheng et al. (2007) designed their research work for the ad hoc network (Liu et al., 2019; Mondal et al., 2021). Their proposed algorithm concept is based on token asking. In their model, there are N number of processes and during the message transmission, these nodes behave like a router to network to support a multi-hop path. Point to point protocol occurs between two successive nodes. Under the assumption of no shared clock and memory, with the defined diameter of the network, they calculate the message complexity.

Atreya et al. (2007) suggest a group mutual algorithm based on quorum. As per their algorithm, they allowed processes of a similar type to enter their CS simultaneously and provided a mutual exclusion in the case of different types of processes. In their approach, a set of processes demands for CS execution then they lock their respective quorums and the algorithm chooses leader among them. In order to allow a leader into its CS, all the other processes as followers left the quorum by communicating to the leader.

Czyzowicz et al. (2011) targeted mutual exclusion and consensus properties in the distributed environment. In their proposed architecture, processes can communicate with each other through multiple access channels and there is a chance of collision during this kind of communication. They showed that their system is feasible in any of the collision detection, global clock information consisting of a number of rounds by any process and number of processes are available in the system.

Wu et al. (2015) proposed an algorithm to implement the concept of mutual exclusion on handling the traffic control at their intersection points. As per their proposal, they introduce a conflict graph based on the traffic information and try to find the corresponding intersection points in that. They showed that the behaviour of this traffic handling is the same as the scenario in distributed mutual exclusion where no two vehicles are allowed to enter the intersection point of lanes.

Neamatollahi et al. (2017) presented their mutual exclusion solution with N number of processes arranged in a torus logical architecture with each row having $\sqrt{N}$ processes. Their solution is based on token perceptual movement in the network that took place through the columns of the underlying network topology. Any CS asking request generated by any node travelled through rows of the torus and once it met with a token, the token started to move towards the asking node in order to serve its CS invocation. The average message complexity of their algorithm lies in the range of $2\sqrt{N} + 1$ to $3\sqrt{N} + 1$.

3 System model

This current study proposes an algorithm to solve the mutual exclusion problem in a distributed architecture. In this solution, N processes have been considered on an almost complete tree logically and a token has been placed at the root node of this tree. Whenever any process demands for CS then the token travels towards that CS demanding process and when the token is at the process, then it is allowed to invoke its corresponding CS. In between, if any other process asks for its CS to execute then that process sends its request towards the root of the tree and enqueue the request in a queue available at the root node. Once the process which is in its CS releases the token, the token

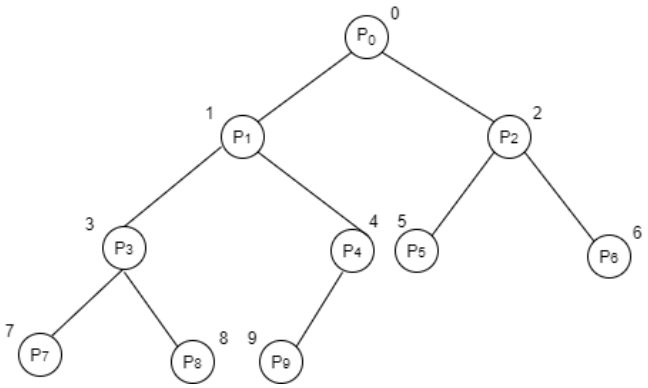
again travels towards root node and before serving to the next CS request for a process, a swapping occurs in the root node and another node in the tree based on the node's availability to avoid the centralisation in the system. Once swapping is done, again token moves towards the next CS demanding process in order to serve.

In order to define various data structures in the system, this algorithm uses two single-valued attributes for the token as the source index and destination index. Then a queue has been defined at the root node in order to keep track of CS demanding processes with a table where the exact path in between two nodes has been stored in the form of link list implementation.

4 Proposed algorithm

In our study, the proposed algorithm has N nodes that represent the processes. These processes are organised like P_0, P_1, P_2, …, P_{N-1}. Then it takes an array as *Processes_Array* of size N and starts inserting the processes into this array.

Figure 1 LACBT as *T* (N = 10)



Then our model creates a logical almost complete binary tree (LACBT) from *Processes_Array*. Corresponding indexes of various nodes in LACBT can be obtained from *Processes_Array* as per equations (1)–(3). Based on the above information gathered, corresponding LACBT as *T* (where *N* = 10) is shown in Figure 1.

$$Parent(i) = \lceil (i-1)/2 \rceil, \quad \text{if } i \neq 0 \tag{1}$$

$$Left(i) = (2i+1), \quad \text{if } (2i+1) < n \tag{2}$$

$$Right(i) = (2i+2), \quad \text{if } (2i+2) < n \tag{3}$$

After the creation of LACBT, the proposed solution finds and stores the exact path in between the root node index with others. Implementation steps are shown in Algorithm 1.

Based on Algorithm 1, a table is created as *M* shown in Table 1 in which the first column represents the key as (source node, destination node) and the second column is the exact path in between the source node and destination node.

Algorithm 1 Path finding

(Steps)	
1	**START**
2	int j, k // j = index of root node, i.e., k ∈ {0, 1, 2, …, n – 1
3	Linklist path
4	Table M[n + 1][2]
5	**FOR** j = 0, k = 0 to (n – 1)
6	Run DFS from j to k and push the traversed node's index into the stack
7	Backtracking occurs, POP the element from the stack and add to path
8	M[0][0]=(j,k) and M[0][1]=path
9	k=k+1
10	**END FOR**
11	**END**

Table 1 A path between root node to others

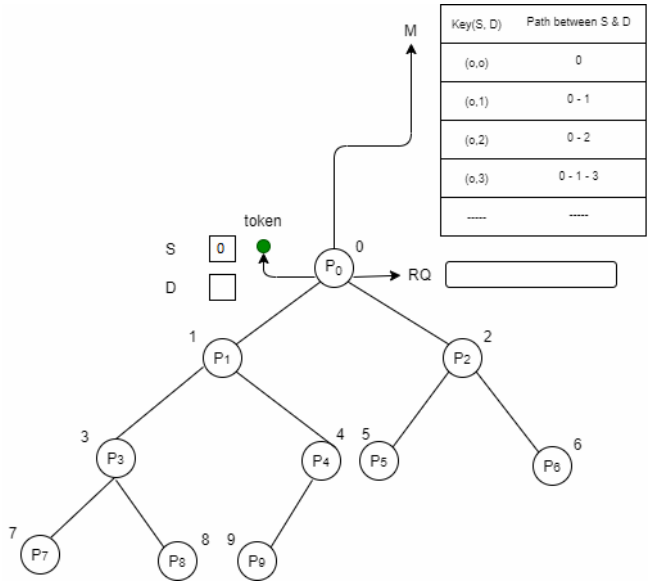
Key (j,k)	Path
(0, 0)	-
(0, 1)	0 → 1
(0, 2)	0 → 2
(0, 3)	0 → 1 → 3
…	.....
(0, n – 1)	0 → 2 → 6 → 14 … → n – 1

Table *M* is placed at the root node and along with that, a queue is created as *request_queue* (*RQ*) at the root node which is initially empty. Then the token is generated at the root node consisting of two attributes, *source_index* represents as *S* and *destination_index* as *D*, respectively. Token *S* is initialised with 0, i.e., root node index. This is explained in Algorithm 2 and the overall visualisation of LACBT is like Figure 2.

Algorithm 2 Data initialisation

(Steps)	
1	**START**
2	**FOR** i = 0 to size(Processes_Array) – 1
3	boolean req_flg = FALSE
4	boolean exec_flag = FALSE
5	**END FOR**
6	int x = 0, y = size(Processes_Array) – 1
7	Place M to root node of T
8	Create queue request_queue as RQ
9	Move RQ to root node of T
10	**Create_Token()**
11	int token.source_index, token.destination_index
12	token.S ← T's root index, i.e., 0
13	token.D ← NULL
14	**END Create_Token()**
15	**END**

Figure 2 Visualisation of LACBT (see online version for colours)



Preliminary steps for this proposed work have been done at this point. Now we proceed towards the CS request and release an implementation of this model done by a process in the distributed system. Algorithm 3 explains this implementation in detail.

Algorithm 3 Proposed algorithm

(Steps)

1 **START**
2 **CS_Request(Pi)** // i = index of corresponding process in T not the process sequence no.
3 Pi.req_flag = TRUE
4 Pi.exec_flag = FALSE
5 int enqueue_element ← i
6 int temp ← i
7 **WHILE** (temp != 0)
8 temp ← Parent (temp)
9 **END WHILE**
10 Process at temp.RQ.enqueue (enqueue_element)
11 **IF** (RQ != EMPTY && token is at root)
12 token.D ← dequeue from RQ
13 **END IF**
14 **IF** (token is at root && token.S != NULL && token.D != NULL)
15 token searches for key (S,D) column in M and access the exact path corresponding to key (S, D) to reach the destination index
16 **IF** (current process index == token's destination index)
17 Pi .exec_flag= TRUE
18 Pi enters into CS
19 **END IF**
20 **END IF**
21 **END CS_Request(Pi)**
22 **CS_Release(Pi)** // i= index of corresponding process in T
23 int temp ← token.D
24 **WHILE** (temp != 0)
25 temp ← Parent (temp)
26 MOVE token to tempth node in T
27 **END WHILE**
28 token.D ← NULL
29 SWAPPING_MODEL(x, y) //The below condition is added due to the next CS requesting process Pm
30 Pi.req_flag= FALSE
31 Pi.exec_flag= FALSE
32 **END CS_Release(Pi)**
33 **SWAPPING_MODEL(x, y)** // for definition of x and y, refer Algorithm 2
34 Pi = Processes_Array[x]
35 Pj = Processes_Array[y]
36 **IF** (!Pi.req_flag)
37 int k
38 static int flag = 0
39 **IF** (y == 0)
40 y = size(Processes_Array)-1
41 **END IF**
42 **WHILE** (y != 0)
43 **IF** (!Pj.req_flag)
44 Copy Processes_Array[0].M into Pj
45 Copy Processes_Array[0].RQ into Pj
46 SWAP Pi and Pj
47 MOVE Pk.M & Pk.RQ to Pj
48 **END IF**
49 **ELSE**
50 y= y-1
51 Pj = Processes_Array[y]
52 continue
53 **END ELSE**
54 **END WHILE**
55 **END IF**
56 y = y – 1
57 **END SWAPPING_MODEL(x, y)**
58 **END**

Case 1: process P_i request for the CS

1 P_i sends its CS request towards the root node of LACBT in the form of its node's index.

2 When P_i request is at the root node, RQ available at root is updated with that request.

3 Token fetch element from the RQ and update its *destination_index* attribute if the token is at root and RQ is not empty otherwise remains ideal at the root.

4 If both attributes of the token are non-empty, then the token starts moving towards CS requesting process index using its own attributes as *source_index* and *destination_index* with the help of table M shown in

Table 2 in which path between source and destination index is stored (refer Algorithm 3, steps 2–21).

Case 2: processes P_m and P_n also request for the CS in between

1 In such a scenario, case 1's points 1 and 2 are repeated for P_m and P_n.

2 Afterward, case 1's point 3 becomes FALSE because the token is not at the root and so for point 4.

Case 3: processes P_i executed CS and exit

1 The token starts moving towards the root node and once the token is at root, it removes its *destination_index* attribute and makes it blank by updating *NULL* in it.

2 Again, on reaching the root node with respect to token, case 1's point 3 becomes TRUE and so as case 1's point 4 (refer Algorithm 3, steps 22–32).

Till this, CS request and release scenario have been explained through this proposal. The root node process (here P_0) has key responsibilities towards this research work. We really don't want our system to be dependent on a single node or our system to behave like a central coordinator-based system. In order to handle this, our algorithm introduces a swapping node protocol model (SNPM) in T where after each time token reaches at root node after CS release by any process, Swapping is done between the root node and the last index node of LACBT. Next time swapping is done in between root node and (last index – 1) node and so on. By applying this SNPM, our system keeps rotating the load and responsibilities within the entire distributed system (refer Algorithm 3, steps 33–57).

There are below two situations that are handled by this swapping mode:

1 If the root node is waiting for CS to execute, then this SNPM did not perform anything (simply skip).

2 If the swapping node other than the root is waiting for CS to execute, then skip that node and continue with the next node by decreasing its index by one. This scenario repeats until SNPM does not find any suitable node in *T* where the process is not involved in any of CS operations.

4.1 Case study

Additionally, in order to explain the proposed research work, a scenario is explained in detail with the assistance of Figures 4–18. There are *N* processes in the system that are arranged in LACBT and shown in Figure 1 (here, $N = 10$). As a part of the preliminary step, token (*source_index* with value 0, i.e., root node index and *destination_index* with *NULL*), table *M* and request queue RQ have been created at root note as per Algorithm 1 and Algorithm 2 shown in Figure 3. Till this point system is ready for handling distributed mutual exclusion.

Figure 4 Initial setup and token asking state (see online version for colours)

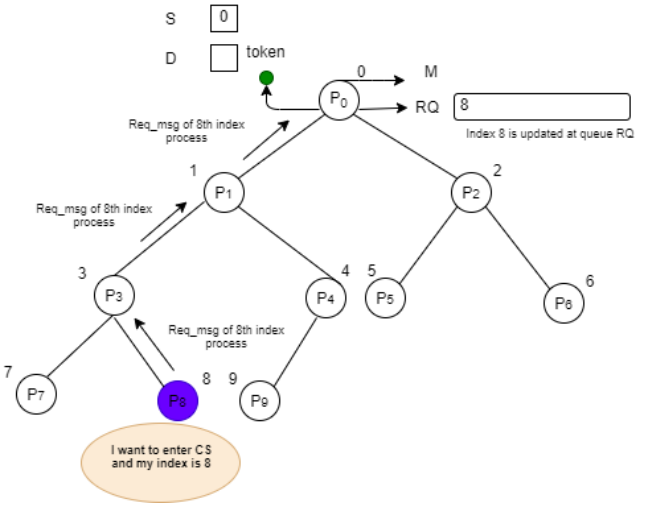


Figure 5 Token travel state (see online version for colours)

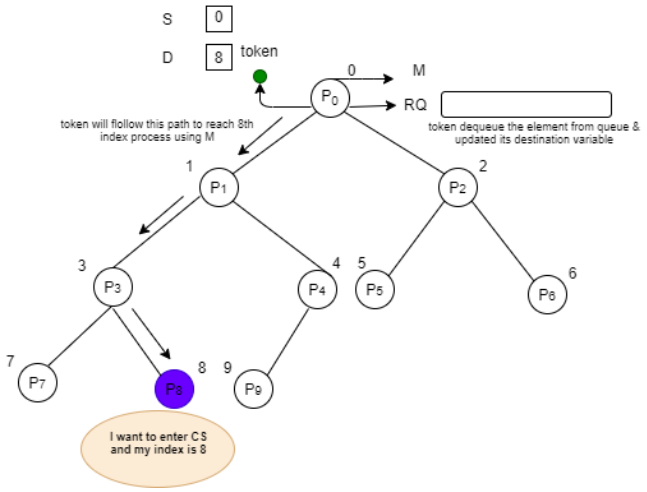


Figure 6 Token at destination state (a) (see online version for colours)

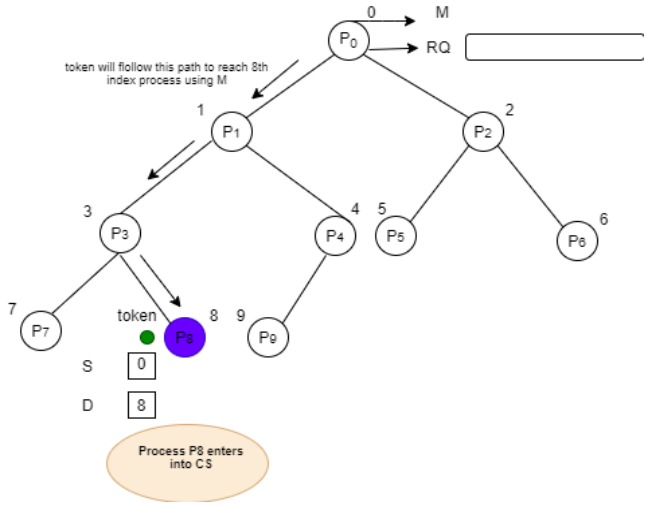


Figure 7 More token requests state (see online version for colours)

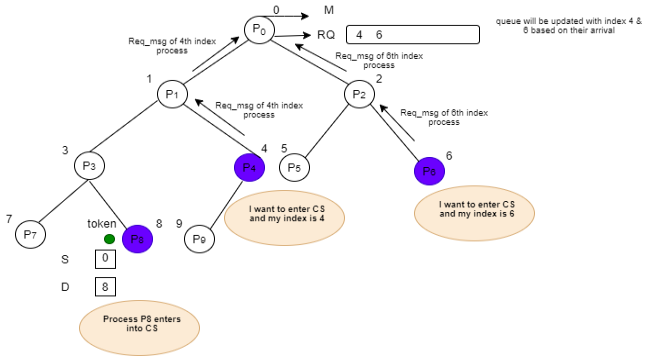


Figure 8 Token release state (a) (see online version for colours)

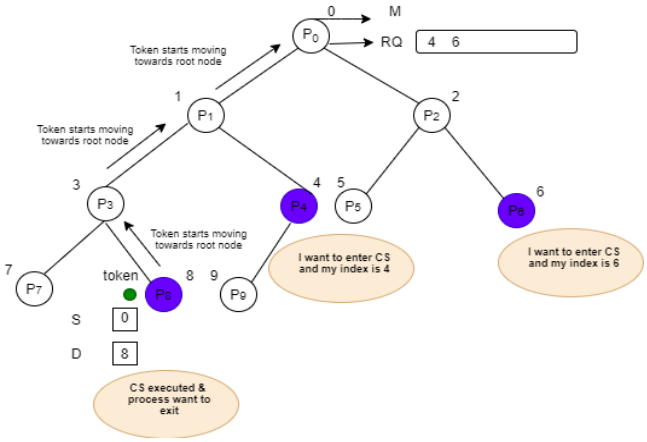


Figure 9 Token at root again after serving the request state (see online version for colours)

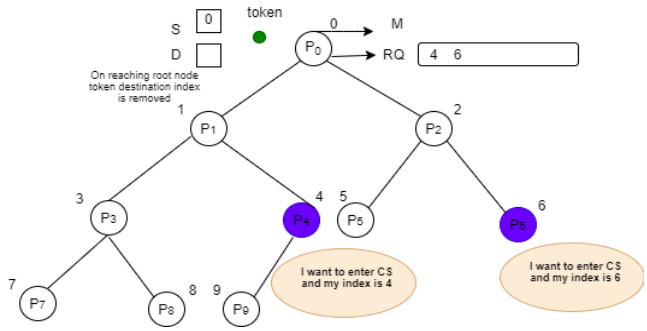


Figure 10 Request queue transfer state (see online version for colours)

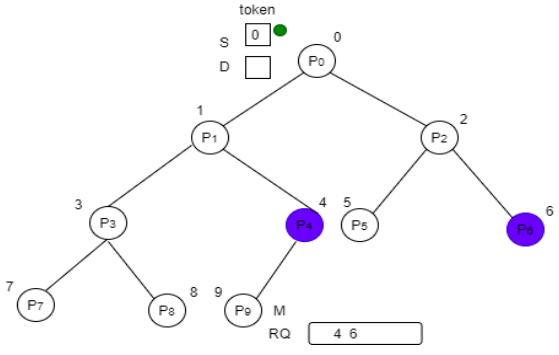


Figure 11 Swapping model state (a) (see online version for colours)

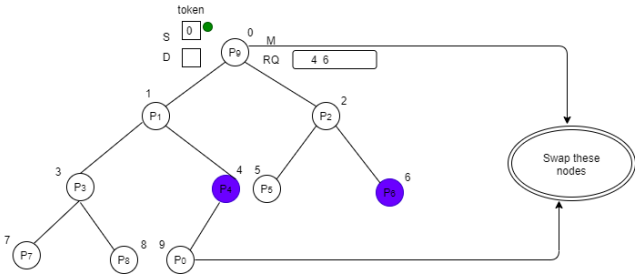
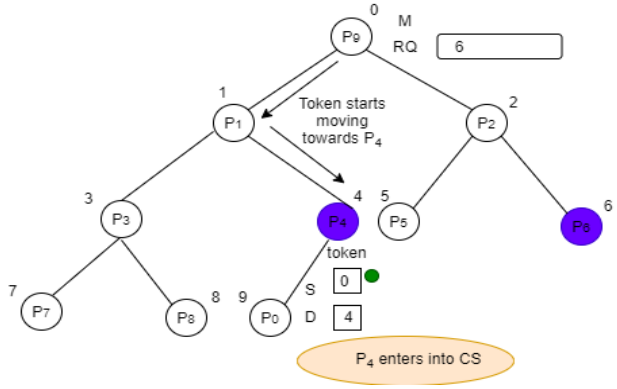


Figure 12 Token travel to next CS asking process state (a) (see online version for colours)



Process P_8 requests for CS to then invokes its request travels towards the root in the form of its respective node index and the request is updated in *RQ* shown in Figure 4. Now as per the proposed algorithm, when the token at the root and *RQ* is not empty, the token gets the element from *RQ* and updates its destination index. Once both the attribute of the token is filled then with the help of *M*, token starts following the path (0-1-3-8) corresponding to the key (0, 8) as shown in Figure 5. After getting a token, P_8 enters into its CS has shown in Figure 6. By the time, P_4 and P_6 also show their interest in executing their CS then again they send their request in the form of their node index towards root node and *RQ* is updated with their requests shown in Figure 7. In the time being P_8 is done with its CS invocation and sends the token towards the root node as shown in Figure 8 and on reaching the root node, token removes its *destination_index* shown in Figure 9. Before serving the next CS request, i.e., P_4's and P_6's request, the algorithm invoke SNPM in which two nodes are swapped with each other in which the first node is root node and the second node is the last indexed node of *T* (if last indexed node is in waiting state then next backward node is chosen, same continues till any suitable process finds for swapping) but before that algorithm move table *M* and *RQ* towards the second node chosen by SNPM. This swapping is logically done in the process array from which *T* is constructed. P_0 and P_9 is then swapped as shown in Figures 10 and 11. After the swapping of nodes, the token again starts moving towards next CS requesting process, i.e., P_4. After serving to P4, again swapping occurs and then the next CS requesting process is served, i.e., P_6 which is shown in Figures 12 to 18 and so on.

Figure 13 Token release state (b) (see online version for colours)

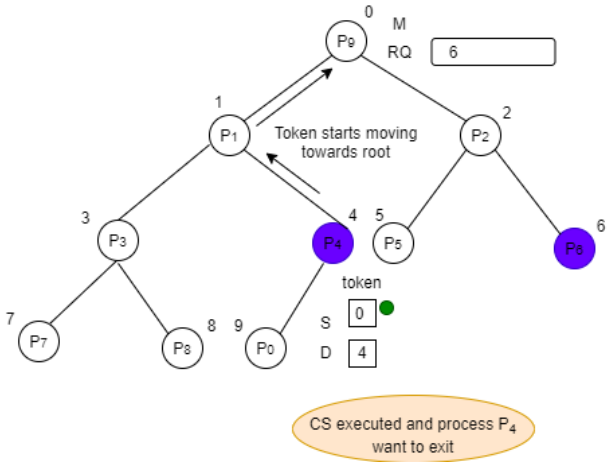


Figure 14 Swapping model state (b) (see online version for colours)

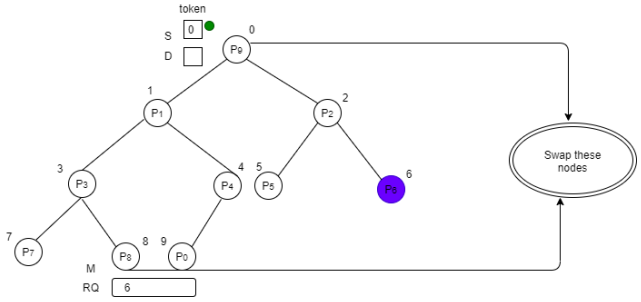


Figure 15 Token travel to next CS asking process state (b) (see online version for colours)

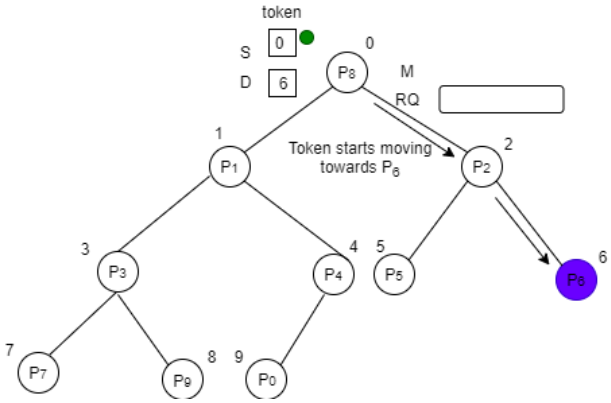


Figure 16 Token at destination state (b) (see online version for colours)

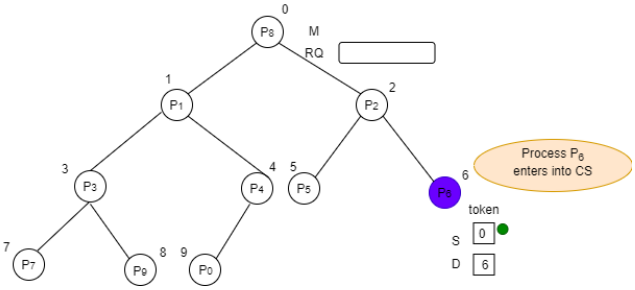


Figure 17 Token release state (c) (see online version for colours)

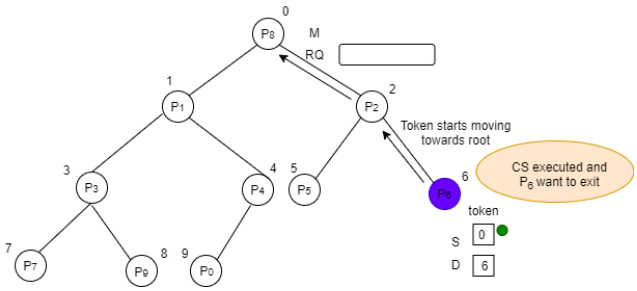
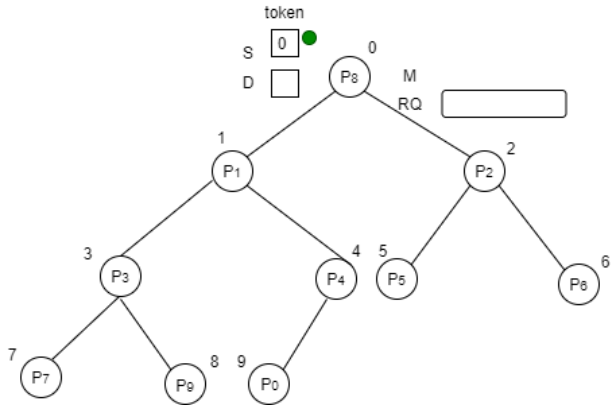


Figure 18 Idle state (see online version for colours)



4.2 Messages exchanged

4.2.1 Light load situation

The present work assumes that only a single process attempts to invoke its CS at a time for the consideration of a light load situation. As a worst-case scenario, the leaf node process asks for its CS to invoke transfer the request to the parent and further till it reaches to root node. ($\lceil \log N \rceil - 1$) messages (N = nodes in the system) are required for this. Then token starts moving towards the CS demanding process and again ($\lceil \log N \rceil - 1$) messages required for this. Once the token-holder node releases its CS, the token again starts approaching towards root node then ($\lceil \log N \rceil - 1$) message exchanges are required. In this complete scenario of light load, a total of $3(\lceil \log N \rceil - 1)$, $N \leq 3$ message exchanges are required by a process to access the shared resource in the system.

4.2.2 Average load situation

The current study assumes that ($\lceil \log N \rceil + 1$) processes attempt to invoke their CS simultaneously. In such a case, the total number of messages exchange are $3(\lceil(\lceil \log(N + 1)\rceil(\lceil \log(N + 1)\rceil - 1))/2\rceil)$ and the average number of exchanges in terms of messages per CS invocation by a process is $3(\lceil(\lceil \log(N + 1)\rceil - 1))/2\rceil)$.

During these transactions in the system, number of total messages exchange can be seen in equation (4). The series in equation (4) can be comprised in summation form like equation (5) which is further be evaluated and equivalent to $(3\lceil(\lceil \log(N + 1)\rceil(\lceil \log(N + 1)\rceil - 1))/2\rceil)$. The average number

of messages exchanged by a process to invoke CS are $(3\lceil(\lceil\log(N+1)\rceil(\lceil\log(N+1)\rceil-1))/2\rceil)/\lceil\log(N+1)\rceil = 3\lceil(\lceil\log(N+1)\rceil-1)/2$.

4.2.3 High load situation

During high load situations where all the processes in the logical system attempt to invoke their CS and under this consideration that logical architecture is a fully complete tree, the total number of messages exchange are $3\lceil(N+1)\rceil$ $3\lceil(N+1)\rceil$ $[\lceil\log(N+1)\rceil + 2(2^{-\lceil\log(N+1)\rceil}-1]$ and an average number of messages per CS invocation by any process is $6[\lceil\log(N+1)\rceil + 2(2^{-\lceil\log(N+1)\rceil}-1]$. This can be obtained from equation (7) in which the total number of messages are shown and in equation (4) where an average number of messages exchange are discussed.

4.3 Limitation to the proposed algorithm

Let's discuss the basic limitations of our proposed algorithm in this section. Nowadays due to various technological enhancements, wireless networks are the fastest-growing architectures. Communication and collaboration among the nodes in such a network are complex tasks as the node movements are highly dynamic. We have targeted our solution firstly on static network and requires essential modifications as a part of future work for its imposement on dynamic architectures like the wireless networks. We had also defined probability density function $f(p_2, p_3)$ on the given sample space $S(P) = \{P_1, P_1', P_2, P_2', P_3, P_3', \ldots, P_n, P_n'\}$ (discussed in Section 5.1) such that $f_{P_2|P_3} = p_3^{(p_2)}$ $f(p_2, p_3) \mid f_{P_3}(p_3)$, which fulfils a contradiction scenario with our assumption of more than one processes involvement in the CS.

5 Proof of DME properties

5.1 Safety

Theorem 5.1: The proposed algorithm achieves mutual exclusion.

Proof: Safety is must implemented property to ensure that no two processes in the system invoke their CS simultaneously in order to support mutual exclusion. Our algorithm achieves safety as an invocation of CS by any process depends on the token. The process can invoke CS of its own if it possesses the token. According to our solution, the token can only be granted for a single process at a time by fetching data from RQ and travels towards that process for allowing that into its corresponding CS. Also at any point of time P1 can only enter into CS, if it is *Pi.exec_flag* is true and no two processes at the same time can have their *exec_flag* true according to our proposal.

Proof: By contradiction to support such a statement, two or more processes must be in their CS. We consider our whole proposed DME as a sample space S(P) in which a set of events are available in the form of Pi(Process allowed to enter into CS) and P_i' (process does not allow to enter into CS).

Hence, $S(P) = \{P_1, P_1', P_2, P_2', P_3, P_3', \ldots, P_n, P_n'\}$.

Assuming P_2 and P_3 are in CS at the same time. Each and every event in equations (9) and (10) must be true to support our assumption that leads to equations (11), (12) and (13) respectively. Our initial assumption fails as P_2' and P_3' cannot be true at the same time otherwise, both P_2 and P_3 will not be allowed to enter into CS as per concluded from equation (13). Similarly, Considering that P_2 and P_3 have probability density function $f(p_2, p_3)$. Defining a such of this conditional probability density of P_2 such that $P_3 = p_3$ by $f_{P_2|P_3} = p_3^{(p_2)} = f(p_2, p_3) \mid f_{P_3}(p_3)$. This restricts $f(p_2, p_3)$ to the given value of p_3 with the assumption definition of $f_{P_3}(p_3) \neq 0$ and $f_{P_3}(p_3) = \int_{-\infty}^{\infty} f(p_2, p_3)dp_2 < \infty$.

Probability of even P_2 to occur given that P_3 has already occurred $P(P_2|P_3) = P(P_2P_3)/P(P_3)$. Given the law of condition on P_2 is $P_3 \in (p_3 - \varepsilon, p_3 + \varepsilon$ if $\varepsilon \to 0$. We concluded

$$F_{P_2|P_3} = p_3^{(k)} := \lim_{\varepsilon \to 0} P\{P_2 \leq k \mid P_3 \in (p_3 - \varepsilon, p_3 + \varepsilon)\}$$

on set $f_{P_2|P_{3=p_3^{(k)}}} = F'_{P_2|P_{3=p_3^{(k)}}}$ which is consistent with our contradiction concluding $P(P_3) = 0$ on an event P_2. Observing a contradiction exist that proves our assumption that more than one process can be allowed to enter into CS simultaneously is false.

5.2 Liveness

Theorem 6.2: The proposed algorithm achieves liveness.

Proof: Liveness ensures that neither deadlock nor starvation can occur in the system during the message transactions. To prove that our system is deadlock/starvation-free begins with this contradictory assumption that the system is suffered from these, hence token has information about more than one process at the same time which is not correct because in our algorithm, token consists of two single-valued attributes by which it decides the exact process to be allowed for CS invocation. Since this is a contradiction to our initial assumption, we can conclude that this system is deadlock-free. Also, our proposed system based on LACBT architecture ensures that none of the transactions creates a circular form during the communication.

5.3 Fairness

Theorem 6.3: The proposed algorithm is fair.

Proof: Fairness assured the system to be starvation free, i.e., processes should not wait for an indefinite time for their chance to invoke CS. The proposed algorithm is starvation-free as processes get a fair chance to execute their CS according to the request to be inserted in RQ as token always gets the information from the request queue and grant that process to invoke its CS whichever information is currently held by a token. This ensures that processes in the system are allowed to invoke their CS in the order they request for the same.

$$3\left\lceil \begin{array}{c} \left(\lceil \log(N+1)\rceil -1\right)+\left(\lceil \log(N+1)\rceil -2\right) \\ +\left(\lceil \log(N+1)\rceil -3\right)+\ldots+0 \end{array} \right\rceil \tag{4}$$

$$3\left\lceil \sum_{k=1}^{\lceil \log(N+1)\rceil} \left(\lceil \log(N+1)\rceil -k\right) \right\rceil \tag{5}$$

$$\begin{aligned} &3\lceil (N+1)\rceil \left\lfloor \frac{1}{2^1}\left(\lceil \log(N+1)\rceil -1\right)\right. \\ &\left.+\frac{1}{2^2}\left(\lceil \log(N+1)\rceil -2\right)+\frac{1}{2^3}\left(\lceil \log(N+1)\rceil -3\right)+\ldots+0\right\rfloor \end{aligned} \tag{6}$$

$$\begin{aligned} &\left(3\lceil (N+1)\rceil \sum_{k=1}^{\lceil \log(N+1)\rceil} \frac{1}{2^k}\left(\lceil \log(N+1)\rceil -k\right)\right)\Big/N \\ &=6\left\lfloor \lceil \log(N+1)\rceil +2\left(2^{-\lceil \log(N+1)\rceil}-1\right)\right\rfloor \end{aligned} \tag{7}$$

$$\mathrm{P}_2 \rightarrow \left(\mathrm{P}_1' \wedge \mathrm{P}_3' \wedge \mathrm{P}_4' \wedge \ldots \wedge \mathrm{P}_n'\right) \tag{8}$$

$$\mathrm{P}_3 \rightarrow \left(\mathrm{P}_1' \wedge \mathrm{P}_2' \wedge \mathrm{P}_4' \wedge \ldots \wedge \mathrm{P}_n'\right) \tag{9}$$

$$\left(\mathrm{P}_1' \wedge \mathrm{P}_3' \wedge \mathrm{P}_4' \wedge \ldots \wedge \mathrm{P}_n'\right) \wedge \left(\mathrm{P}_1' \wedge \mathrm{P}_2' \wedge \mathrm{P}_4' \wedge \ldots \wedge \mathrm{P}_n'\right) \Leftrightarrow T \tag{10}$$

$$\left(\mathrm{P}_1' \wedge \mathrm{P}_2' \wedge \mathrm{P}_3' \wedge \mathrm{P}_4' \wedge \ldots \wedge \mathrm{P}_n'\right) \Leftrightarrow T \tag{11}$$

$$\begin{aligned} &\forall \mathrm{P_i} \Rightarrow \mathrm{T}, \quad 1 \le \mathrm{i} \le \mathrm{n} \\ &\left(\mathrm{P}_1' = \mathrm{T},\ \mathrm{P}_2' = \mathrm{T},\ \mathrm{P}_3' = \mathrm{T},\ \mathrm{P}_4' = \mathrm{T}, \ldots, \mathrm{P}_\mathrm{n}' = \mathrm{T}\right) \end{aligned} \tag{12}$$

6 Comparative analysis

From the existing literature survey, we have performed a comparative analysis in between various existing DME algorithms with our proposed model in terms of the total number of processes available in the system and the average number of message exchanges during light load condition shown in Figure 19 and high load condition shown in Figure 20. We had modelled our solution through the given mathematical model towards the calculation of message complexity as per the discussion in Section 4.2. Our solution shows better results in high load situations in which continuous requests for CS invocation are made by the processes available in the system. Detailed comparison in terms of average synchronisation delay and message complexity can be seen in Table 2 for existing algorithms to handle mutual exclusion.

Figure 19 Message complexity in light load situation

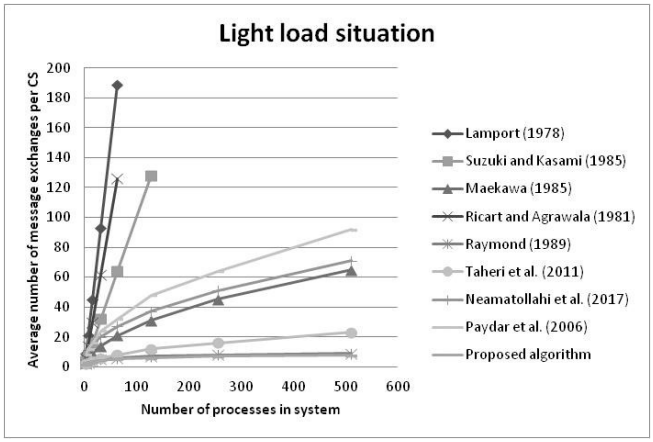
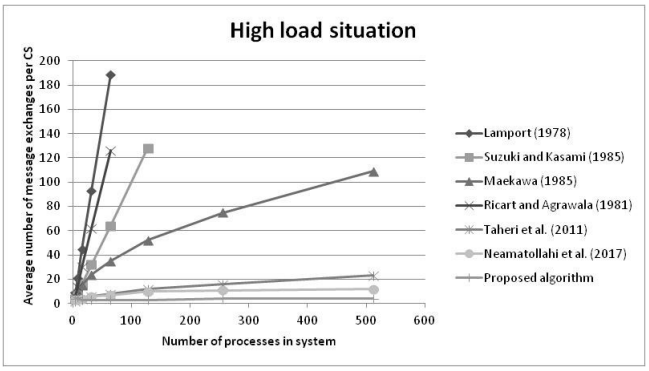


Figure 20 Message complexity in high load situation



Notation used in Table 2.

N	number of processes
Z	total number of groups
S	size of quorum
R	number of mutually exclusive resources
$Bc(Pm)$	ad hoc traversing cost from node Pm
$Hp(Pn, Pm)$	number of switch count between Pm and Pn
$\|Q\|$	quorum size
l	maximum number of processes from where CS request can be received by a node
m	maximum size of quorum
n	maximum size of request.

Table 2 Performance measure of various DME-based algorithms

Algorithm	*Approach/ structure*	*Average synchronisation delay*	*Message complexity*	
			Heavy load	*Light load*
Lamport (1978)	Non-token-based	$O(1)$	$3(N-1)$	$3(N-1)$
Ricart and Agrawala (1981)	Non-token-based	$O(1)$	$2(N-1)$	$2(N-1)$
Maekawa (1985)	Quorum-based	$O(1)$	$5(\sqrt{N}-1)$	$3(\sqrt{N}-1)$
Suzuki and Kasami (1985)	Token-based	$O(1)$	N	N
Raymond (1989)	Token-based	$O(N)$	4	$O(\log N)$
Nishio et al. (1990)	Token-based	$O(N)$	N	N
Helary et al. (1994)	Token-based	$O(\log N)$	$O(N)$	$O(N)$
Wu and Joung (2000)	Ring-based	-	$O(NZ)$	$O(NZ)$
Lodha and Kshemkalyani (2000)	Priority-based	-	$N-1$	$2(N-1)$
Keane and Moir (2001)	Cache coherent-based	-	$O(N)$	$O(N)$
Cao and Singhal (2001)	Quorum-based	$O(T)$	$O(K)$	$O(K)$
Cantarell (2005)	Token ring-based	-	$\log(\min(N, R))$	$\log(\min(N, R))$
Paydar et al. (2006)	Logical structure-based	-	-	$4\sqrt{N}$
Zheng et al. (2007)	Token-based	$2ST + E$	Bc(Pm) + Hp(Pn, Pm)	Bc(Pm) + Hp(Pn, Pm)
Atreya et al. (2007)	Quorum-based	-	-	$O(lmn)$
Kakugawa et al. (2008)	Non-token-based	4	$O(\|Q\|)$	$O(\|Q\|)$
Taheri et al. (2011)	Logical structure-based	$O(\sqrt{N})$	$\sqrt{N}$	$O(\sqrt{N})$
Neamatollahi et al. (2017)	Logical torus structure-based	$O(\sqrt{N})$	3	$2(\sqrt{N}+1)$ to $3(\sqrt{N}+1)$
Proposed algorithm	LACBT-based	$O(\log N)$	$6[\lceil\log(N+1)\rceil + 2(2^{-\lceil\log(N+1)\rceil} - 1)]$	$3(\lceil\log N\rceil - 1), N \leq 3$

7 Conclusions and future scope

The research work proposed through this paper is a novel token-based distributed mutual exclusion algorithm. The implementation of this given algorithm is based on logical almost complete tree topology. The average number of message exchanges per CS by a process through this algorithm is in logarithmic which is effective in contrast to other existed algorithms till date. The system is scalable and satisfies all necessary properties of distributed mutual algorithms. The movement of the token in this architecture maintains the sequence of CS demands by processes. For future work, the point of focus will be on the generality of loss in terms of nodes and tokens.

References

Atreya, R., Mittal, N. and Peri, S. (2007) 'A quorum-based group mutual exclusion algorithm for a distributed system with dynamic group set', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No. 10, pp.1345–1360, doi:10.1109/tpds.2007.1072.

Cantarell, S. (2005) 'Group mutual exclusion in token rings', *The Computer Journal*, Vol. 48, No. 2, pp.239–252, doi:10.1093/comjnl/bxh077.

Cao, G. and Singhal, M. (2001) 'A delay-optimal quorum-based mutual exclusion algorithm for distributed systems', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 12, pp.1256–1268, doi:10.1109/71.970560.

Czyzowicz, J., Gasieniec, L., Kowalski, D.R. and Pec, A. (2011) 'Consensus and mutual exclusion in a multiple access channel', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 7, pp.1092–1104. doi:10.1109/tpds.2010.162.

Dai, Y. and Wang, T. (2021) 'Prediction of customer engagement behaviour response to marketing posts based on machine learning', *Connection Science*, Vol. 33, No. 4, pp.891–910, doi: 10.1080/09540091.2021.1912710.

Daid, R., Kumar, Y., Hu, Y. and Chen, W. (2021) 'An effective scheduling in data centers for efficient CPU usage and service level agreement fulfilment using machine learning', *Connection Science*, Vol. 33, No. 4, pp.954–974, doi: 10.1080/09540091.2021.1926929.

Dijkstra, E.W. (1965) 'Solution of a problem in concurrent programming control', *Commun. ACM*, Vol. 8, No. 9, p.569, doi:10.1145/365559.365617.

Gupta, U. and Gupta, D. (2019) 'An improved regularization based Lagrangian asymmetric ν-twin support vector regression using pinball loss function', *Appl. Intell.*, Vol. 49, pp.3606–3627, doi: 10.1007/s10489-019-01465-w.

Helary, J-M., Mostefaoui, A. and Raynal, M. (1994) 'A general scheme for token and tree-based distributed mutual exclusion algorithms', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 11, pp.1185–1196, doi:10.1109/-71.329-670.

Kakugawa, H., Kamei, S. and Masuzawa, T. (2008) 'A token-based distributed group mutual exclusion algorithm with quorums', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 9, pp.1153–1166, doi:10.1109/tpds.2008.22.

Keane, P. and Moir, M. (2001) 'A simple local-spin group mutual exclusion algorithm', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 7, pp.673–685, doi:10.1109/71.940743.

Kordestani, J.K., Meybodi, M.R. and Rahmani, A.M. (2020) 'A note on the exclusion operator in multi-swarm PSO algorithms for dynamic environments', *Connection Science*, Vol. 32, No. 3, pp.239–263, doi: 10.1080/09540091. 2019.1700912.

Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system', *Commun. ACM*, Vol. 21, No. 7, pp.558–565.

Liu, W., Hu, E., Su, B. and Wang, J. (2021) 'Using machine learning techniques for DSP software performance prediction at source code level', *Connection Science*, Vol. 33, No. 1, pp.26–41, doi: 10.1080/09540091.2020.1762542.

Liu, Z., Japkowicz, N., Wang, R. and Tang, D. (2019) 'Adaptive learning on mobile network traffic data', *Connection Science*, Vol. 31, No. 2, pp.185–214, doi: 10.1080/09540091. 2018.1512557.

Lodha, S. and Kshemkalyani, A. (2000) 'A fair distributed mutual exclusion algorithm', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 6, pp.537–549, doi:10.1109/71.862205.

Maekawa, M. (1985) 'A √N algorithm for mutual exclusion in decentralized systems', *ACM Trans. Comput. Syst.*, May, Vol. 3, No. 2, pp.145–159, doi: 10.1145/214438.214445.

Mondal, P.M., Sanchez, L.P.A., Benedetto, E., Shen, Y. and Guo, M. (2021) 'A dynamic network traffic classifier using supervised ML for a Docker-based SDN network', *Connection Science*, Vol. 33, No. 3, pp.693–718, DOI: 10.1080/09540091.2020.1870437.

Neamatollahi, P., Sedaghat, Y. and Naghibzadeh, M. (2017) 'A simple token-based algorithm for the mutual exclusion problem in distributed systems', *J. Supercomput.*, Vol. 73, pp.3861–3878, doi:10.1007/s11227-017-1985-y.

Nishio, S., Li, K.F. and Manning, E.G. (1990) 'A resilient mutual exclusion algorithm for computer networks', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp.344–355, doi:10.1109/71.80161.

Parihar, A.S. and Chakraborty, S.K. (2021) Token-based approach in distributed mutual exclusion algorithms: a review and direction to future research', *J. Supercomput.*, doi:10.1007/s11227-021-03802-8.

Parihar, A.S., Prasad, D., Gautam, A.S. and Chakraborty, S.K. (2021) 'Proposed end-to-end automated e-voting through blockchain technology to increase voter's turnout', in Prateek, M., Singh, T.P., Choudhury, T., Pandey, H.M. and Nhu, N.G. (eds.): *Proceedings of International Conference on Machine Intelligence and Data Science Applications. Algorithms for Intelligent Systems*, Springer, Singapore, doi: 10.1007/978-981-33-4087-9_5.

Paydar, S, Naghibzadeh, M. and Yavari, A. (2006) 'A hybrid distributed mutual exclusion algorithm', *International Conference on Emerging Technologies 2006, ICET'06*. IEEE, pp.263–270.

Raymond, K. (1989) 'A tree-based algorithm for distributed mutual exclusion', *ACM Transactions on Computer Systems*, Vol. 7, No. 1, pp.61–77, doi:10.1145/58564.59295.

Ricart, G. and Agrawala, A.K. (1981) 'An optimal algorithm for mutual exclusion in computer networks', *Communications of the ACM*, Vol. 24, No. 1, pp.9–17, doi:10.1145/ 358527.358537.

Saxena, P.C. and Rai, J. (2003) 'A survey of permission-based distributed mutual exclusion algorithms', *Computer Standards & Interfaces*, Vol. 25, No. 2, pp.159–181, doi:10.1016/S0920-5489(02)00105-8.

Soto-Morettini, D. (2017) 'Reverse engineering the human: artificial intelligence and acting theory', *Connection Science*, Vol. 29, No. 1, pp.64–76, doi: 10.1080/09540091.2016. 1271398.

Suzuki, I. and Kasami, T. (1985) 'A distributed mutual exclusion algorithm', *ACM Transactions on Computer Systems*, Vol. 3, No. 4, pp.344–349, doi:10.1145/6110.214406.

Taheri, H., Neamatollahi, P. and Naghibzadeh, M. (2011) 'A hybrid token-based distributed mutual exclusion algorithm using wraparound two-dimensional array logical topology', *Inf. Process Lett.*, Vol. 111, pp.841–847.

Tan, C. (2020) 'Digital Confucius? Exploring the implications of artificial intelligence in spiritual education', *Connection Science*, Vol. 32, No. 3, pp.280–291, doi: 10.1080/ 09540091.2019.1709045.

Tanenbaum, A.S. and Steen, M.V. (2016) *Distributed Systems: Principles and Paradigms*, 2nd ed., CreateSpace Independent Publishing Platform, ASIN 153028175X.

Wu, K.P. and Joung, Y.J. (2000) 'Asynchronous group mutual exclusion in ring networks', *IEEE Proceedings – Computers and Digital Techniques*, Vol. 147, No. 1, p.1, doi:10.1049/ip-cdt:20000162.

Wu, W., Zhang, J., Luo, A. and Cao, J. (2015) 'Distributed mutual exclusion algorithms for intersection traffic control', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 1, pp.65–74, doi:10.1109/tpds.2013.2297097.

Zheng, W., Song, L.X. and Mei'an, L. (2007) 'Ad hoc distributed mutual exclusion algorithm based on token-asking', *Journal of Systems Engineering and Electronics*, Vol. 18, No. 2, pp.398–406, doi:10.1016/s1004-4132(07)60104-2.