An Algorithm Proposed for Distributed K-Mutual Exclusion

[1]Ashish Chauhan, [2]Dr. Syed Mohammad Saqib
[1]chauhan.mrt@gmail.com, [2]saqibmca@gmail.com
[1]Ph D Scholar, [2]Assistant Professor, Himalayan Garhwal University,
Pokhra, District Pauri Garhwal, Uttarakhand

Abstract:-In this research an algorithm is used to accomplish K-mutual exclusion uses K tokens. The N nodes are arranged in a logical forest, with the token-holding node serving as the tree's root. Such forests exist for systems with K tokens. The token is transferred between the nodes together with its token-queue and tag. Only nodes in possession of a token are permitted access to the Critical Section. Nodes that are missing a token transmit a request for it along the forest's edges. They enter the vital part after receiving the token. The suggested algorithm achieves K-mutual exclusion, as we exhibit.

1. INTRODUCTION

The distribution of computing among numerous processors has become common in computer systems. Two plans exist for developing such systems. The processors in a closely connected system share memory. Communication in these multiprocessor systems often happens through a shared memory. A clock or memory are not shared by the processors in a loosely connected system. Each processor has a separate local memory. Through various communication networks, including ethernet and FDDI, the processors exchange messages. Distributed systems are the name given to these systems. A distributed system's processors may range in size and purpose. These processors go by a variety of names, including sites, nodes, etc. In a concurrent (or parallel) programme known as a distributed programme, processes communicate with one another by way of message passing. There are two approaches are sued for message passing these are peer to peer and client server approach In client server approach the architecture request the message from clients cause server processes which are always awaiting client request respond. The client and server are effectively in a master-slave relationship as a result, and once allocated, these roles are fixed. In a peer-to-peer model, processes work together to accomplish a calculation or to deliver a service while being on an equal footing (i.e., there is no master-slave connection between them). There are multiple modes i.e. coupling between the networks through which the communication network becomes efficient and convenient by distributed systems. This makes it feasible to share files at distant locations, publish files at distant locations, process data in a distributed database, and use distant hardware (such array processors). Another option is load sharing, which involves partitioning a computation and running the separate computations simultaneously on other sites.

1.1 Distributed system's topology:

There are various ways to physically connect the system's sites. Each site in a network that is completely connected has a direct connection to every other site. When a system has been divided into two (or more) of these independent subsystems, it is said to be partitioned. A direct link only exists between some site pairs in a partially linked network. However, any pair of sites can construct a logical channel.

1.2 Mutual Exclusion:

As distributed systems gain popularity, various algorithms are being proposed to solve issues including synchronization distributed concurrency, allocating work among processors, and preserving the reliability of a distributed, dynamic computing system. Mutual exclusion is a technique that protects a shared resource's integrity by handling concurrent requests from numerous uncoordinated users. The location that has gained access to a resource and is carrying out

the necessary tasks is referred to as being in the critical section (CS). Consequently, the site (node) runs in the critical portion whenever a user accesses a shared resource. In a distributed system, mutual exclusion approaches rely on message forwarding between processes. In order to exchange information, the processes freely convey it via messages. In the critical zone of the majority of the mutual exclusion algorithms shown, only one node may operate. The 1-The mutual exclusion is the issue of which can permit one node in the crucial region. In some circumstances, it might be possible to permit a bounded number of nodes (K>=1) to execute concurrently in the crucial region in order to make better use of the resources. The K-mutual exclusion problem is the challenge of allowing at most 'K' nodes to run simultaneously in the important region for a fixed K (K >= 1).

1.3 Relations in Distributed Systems:

In order for arbitrary or connected processes to share data or synchronise execution, communication is necessary. UNIX has a number of capabilities that enable processes to communicate with one another. The fundamental drawback of pipelines is that they prevent processes from interacting without shared predecessors. Despite the fact that they typically cannot be utilised for cross-network communication, named pipes do allow for this facility. Sockets and streams 4 offer effective IPC techniques for communication across networks. By issuing the kill signal, arbitrary processes can also communicate with one another. A well-liked transport-level interface is the 4.2BSD sockets interface. Strong network applications may be created quickly and easily by UNIX programmers thanks to the socket interface. On the majority of networked UNIX systems, and TCP/IP are used to implement OSI layers 1 through 4. It is preferable for UNIX networking applications to use either of the sockets or streams provided by this library of functions to create the transport layer in various networked contexts to establish interconnectivity. Numerous distributed systems contexts benefit from the client-server design.

2. A REVIEW OF DISTRIBUTED 1-MUTUAL EXCLUSION

One-mutual exclusion algorithms is an algorithm that permit only single node to enter in the crucial section. The two types of mutual exclusion algorithms used in distributed systems are token-based and permission-based algorithms. In the permission-based methods, nodes can only access the crucial section (C S) with the consent of a subset of other nodes. The nodes enter the vital phase of the token-based algorithms after receiving a token (a unique privilege).

2.1 Algorithms with Permissions:

In 1978, Lamport put forth one of the earliest distributed mutual exclusion schemes [1]. No a priori order is imposed by the algorithm on how the privilege is transferred from one process to another. Using a message time stamping procedure, the system events are arranged chronologically. Every message's transmitter assigns a number to it that is known as the timestamp or logical date. All of the system's nodes (sites) share a queue that stores requests for crucial sections and release messages. As a result, any request for a release from or addition to the crucial sector is broadcast to every node in the system. Before choosing to enter the vital area based solely on its own queue, a node must first obtain a reply message from each of the other nodes. This is made possible by the acknowledgment-type messages that every node sends in response to a request message. The procedure is permission-based and needs 3 (N - 1) messages for each entry in a crucial portion because there are N nodes in the system. In order to request entry into the important part, a node sends (N - 1) request messages. In response to the request, (N - 6) nodes send acknowledgement messages. Another algorithm i.e. permission-based approach by Ricart and Agrawala from 1981 lowered that the message contained number is needed for each crucial area entrance to 2 (N-1) [2]. By getting rid of the acknowledgment messages utilised by Lamport's algorithm, the improvement is made. There are only (N-1) request the messages and (N-1) contained the affirmative responses. This technique makes the assumption that there is

an error-free transport network, where messages may be transmitted at different times and may even pass each other. When a node I want to enter the critical region, it generates a timestamp and broadcasts the request message to all other nodes. If a request is in the critical category, a node j either accepts it or defers a response. After leaving the important zone, node j transmits any deferred responses to the inquiring nodes. Once every other node in the system has responded, a node reaches the critical area. A node determines if it wishes to enter the critical region by comparing the timestamp of its request with the request it has already received. If its value is less than the one of the received request, it delays sending the reply message. If not, a response is delivered right away. The number of messages was decreased to be between 0 and 2 by Carvalho and Roucairol's proposed upgrade to Ricart and Agrawala's algorithm [1] in the same year (N - 1). The key tenet upon which the enhancement is built is that a node may enter the crucial area as many times as necessary after receiving permission from (N- 1) other nodes, provided that no other node chooses to do so. As a result, once a node I obtains a favourable response from a node j, the node j's implicit permission is valid until the node j makes a request to the node i. As a result, node I has unlimited access to the vital region between the moment node j grants it authorization and the time it sends a request message to node i. Consequently, there may be 0 to 2 messages (N- 1). A permission-based approach with an O(sqrt N) message requirement was proposed by Maekawa in 1985 [3]. In this case, coteries are built using finite projective planes. Each node must request approval from the others. By requiring that each node retain an information structure, Sanders expanded the scope of permission-based algorithms [4]. Which nodes keep track of the states of the other nodes and from which nodes information must be gathered prior to entering the critical section are specified by the information structure. An inform set and a request set make up the information structure. These sets are maintained by each node. A node sends a message to every other node in its inform set each time it changes its state. The states of a node include in-cs, not-in-cs, and waiting. Before proceeding to the crucial step, each node must send the request message to every other node in its request set and get their approval. If the information structure is static, all nodes may be aware of the entire structure. Because dynamic information structures evolve while the algorithm is developed, it's possible that not all nodes are aware of the global information structure.According to a dynamic information structure proposed by Singhal [5], the group of nodes from whom a specific node must request permission to access the critical section varies as the nodes learn more about the state of the system. The information set is composed of the request set and the inform set as proposed by Sanders. In the worst case, this method calls for 2 (N- 1) messages per crucial section entry.

2.2 Token Based Algorithm :

In this algorithm, token is passed and generated for accessing the critical area between the nodes. The first token-based method was proposed by Suzuki and Kasami in 1982 [6]. As an upgrade to their permission-based algorithm described in 1981 [2], In 1983 [7], Ricart and Agrawala presented a token-based strategy. This approach was nearly identical to the token-based technique developed by Suzuki and Kasami [6]. Every other node receives a request for the token along with a timestamp whenever a node I needs to enter the crucial portion of either method. The timestamps of each node's most recent visits are recorded by the token. The token is therefore transferred to the first process whose most recent request has a timestamp higher than the timestamp recorded on the token during the requesting node's most recent visit if node j has it when it leaves the critical region. N messages are required by this procedure, where N - 1 messages are required to broadcast the request and 1 message to obtain the token. The placement of the token can be predicted more accurately by leveraging state information, according to Singhal's proposal [8]. In this method, each site keeps two vectors: a state vector that stores the most recent state of the site, which could either be requesting or not, and a sequence number vector that stores the highest known sequence number for each site. Similar arrays are attached to

the token. All of the sites with a state of requesting in their state vector are included in the list of sites to which a node is most likely to send its request messages. The status of the system is determined by the messages that each node receives and the data that is acquired using a token.

Per crucial section entry, this method calls for a maximum of N messages, where N is the total number of system nodes. 9 Mizuno, Neilsen, and Rao created a token-based technique using quorum agreements [9]. Quorum data structures are comparable to Maekawa's p N method's coteries [3]. The quorums are made up of the request set and acquired set. A node delivers an acquired message to each node in its acquired set aside from itself after getting the token by sending requests to all the nodes in its acquired set away from itself.The quorum type determines the number of messages.

Makki, Pissinou, and Yesha proposed a token-based algorithm using timestamps, the theory of projective planes, and a token queue in 1993 [10]. Here, projective planes theory is used to generate sets of sites. In this method, if a site A wishes to enter the crucial region, it sends request messages to all other sites in the set S. If the token is present at site B in another set T, one of the sites in set S that is also in set T will send an inform message to site B. Following that, Site B broadcasts a message to every site in Set T, notifying Site B by sending a message there. After waiting for their responses, Site B notifies each of the sites in its set T that the token has been sent to another site. The second After gathering the responses from the other sites in its set T, site B sends the token to the inquiring site A. Site A sends a message to all of the other sites in its set informing them that it has the token. The algorithm needs 0 messages in the best case and 4 p (N-2) messages in the worst case for each crucial section entry. 1. A tree-based logical structure Nodes that have a logical tree structure are used by some token-based algorithms. The only node not on this path is the root node, a sink node. The logical structure determines the path a request message takes.Each node maintains a pointer that bars access to the structure. Dynamic and static logical structures are two distinct categories. A node's neighbour is a collection of nodes j that permits the existence of an edge from node I to node j or vice versa. In the dynamic structure, a node's surrounding nodes can occasionally alter. In contrast, the neighbours of a node are constant in a static structure. Trehel and Naimi [11] suggested a method based on a dynamic logical tree structure.

Later, Bernabeu-Auban and Ahamad proposed a similar algorithm [12]. The height of the tree structure is decreased by these techniques via path reversal. In a route reversal, When a request from node I travels along the path to the root node, all nodes in the path except for node I make node I their new parent. The top bound on the amortized cost of path reversal is covered in [13]. The messages are often needed for each crucial section entry (log N). The number of messages depends on the height of the rooted tree. The tree is displayed in the following picture as a solid line, while the implicit queue structure of the algorithm is shown as dotted lines. Raymond [14] suggested a method based on a static logical structure. Unrooted trees are used in the algorithm. When a request moves in the direction of the root node, edge reversals are carried out. As the request from node x travels to the root node, if node I along the journey transfers the request to node j before passing it to the next node along the path, then node j points to node i. The usual number of messages required is O. (log N).In their Dag-Based approach, Neilsen and Mizuno suggested a modification to this algorithm [15]. The approach employs a static, directed acyclic graph (dag) structure. The logical topology of the nodes determines the number of messages required for mutual exclusion. The amount of messages needed is cut in half compared to Raymond's algorithm. Three messages are all that are needed when the best topology, the star topology, is used.

3 REVIEW BASED ON DISTRIBUTED K-MUTUAL EXCLUSION

The K-mutual exclusion problem arises when it is required to restrict the usage of shared resources to a single user at a time in order to protect their integrity. Either token-based algorithms or permission-based algorithms could be used to

implement distributed K-mutual exclusion. A single token or K tokens may be employed in token-based algorithms to accomplish K-mutual exclusion. There have also been proposed fault tolerant K-mutual exclusion algorithms, such as [18].

3.1 Permission Based Algorithms

With these procedures, the node that needs to enter the critical section (CS) first sends request messages to a particular subset of system nodes, then waits for their approval.

3.1.1 Raymond's algorithm

As an extension of Ricart and Agrawala's algorithm [2], Raymond introduced a permission-based technique in 1989 to permit K nodes to be in the critical region simultaneously [19]. The request message and the reply message are the two different sorts of messages used by this algorithm. A node sends request messages to (N- 1) nodes when it wants to access the crucial region and waits for (N -K) reply messages. As a result, the algorithm's lower bound is (2N- K- 1) messages per critical section entry, where N is the number of system nodes and K is the maximum number of simultaneous critical section entries permitted. Each request message, however, might eventually produce a reply message. Consequently, this algorithm's upper bound is 2 (N- 1). Once the node receives the (N-K) reply message, it could enter the critical portion. When the node is in the critical part or when it has left the critical section, the remaining (K -1) reply messages arrive. A reply count array is therefore kept at each node for each node in the system to avoid confusing the reply messages of one attempt with those of another. A node is said to have transmitted all of its reply messages to the asking node if its reply count is zero.If a node is in the critical section or has requested admission into the critical section itself and its sequence number is less than the sequence number of the request received from another node, it defers the reply to the requesting node when it receives a request from another node. If both sequence numbers are same, the request is delayed if the receiving node's identification is smaller. If not, the node will reply to the node making the request. If a node has been in the critical section for a longer period of time than other asking nodes, causing the reply messages to be delayed, the whole number of postponed reply messages is transmitted in a single reply message. The deferred replies for each node are therefore transmitted together as one physical message when a node leaves the crucial section. Sometimes fewer messages might result from this.

3.2 Token Based Algorithms

When a node in a token-based algorithm needs to access the vital region, it sends a request to the other nodes asking for a token (also called a privilege message). A token is required to access the essential area.

3.2.1 Srimani and Reddy's algorithm

Srimani and Reddy [20], who presented a method for repeated entries to a crucial section that reduced the quantity of messages needed for each entry to a critical region in half, improved the technique provided by Raymond [19]. Their algorithm is based on the token-based strategy proposed by Suzuki and Kasami [1]. The technique allows K simultaneous access into the critical sector using K tokens (privilege messages). Any node that is in the vital part must have a token in order to be there. As a result, the messages' lower bound is zero. In the worst case, all K tokens are broadcast to the asking node when there are no other nodes making requests and each token is located at a different node, resulting in a total of (N + K - 1) messages. The requesting node sends (N 1) request messages if it does not have the token (the upper bound). The algorithm employs two different sorts of messages: request messages and privilege messages. Each of the K privilege messages contains an array LN[j] of size N (where N is the number of nodes in the system) that stores the most recent request number for node j. (tokens). A privilege count, an array P LN of size N that compiles the most recent data on the sequence numbers of the requests already handled, an array RN[j] of size N that records the arrival of a request from node j, and an array RN[j] of size N are all kept track of by each node. The P LN array is altered

using the LN arrays of the privilege messages the node has received. The P LN array is used to update the Q in any privilege message, preventing the sending of redundant privilege messages to a requesting node after it has already received the privilege.

3.2.2 Makki et al.'s algorithm

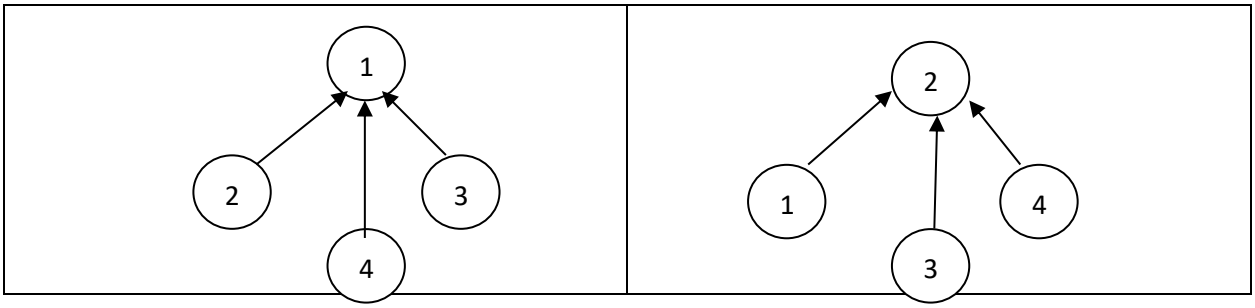
Another token-based K-mutual exclusion strategy was put forth by Makki et al. [21]. This algorithm only makes use of one token. The single token is linked to a queue that stores requests from nodes and a token semaphore that has the total number of critical sites and the maximum number permitted. When a node receives the token and either a non-zero token semaphore or a zero semaphore (signifying that K other nodes are in critical section entries) and then receives a release message from another node, it enters the critical section as the token is passed among the requesting nodes. The "great site" is the last node on the token queue, according to the token queue. This node receives token requests from other nodes not in the token queue, and it keeps them in its local queue. The four different sorts of messages used in this algorithm are the token message, which includes the token queue and the token semaphore, the request message, the good site update message, and the release message. After removing itself, each node that leaves the critical section sends a release message to the kth node on the token queue. This is due to the fact that if only K nodes are allowed in the critical region at time, the kth node from the current node will get the token with a zero semaphore and will therefore need the release message to enter. However, if the good site is listed before the kth node in the token queue, release alerts are sent to it. This results in the good site receiving a release message from the K preceding sites in order to increase the semaphore value back to its maximum of K.

4. ALGORITHM FOR PROPOSED K-MUTUAL EXCLUSION

We offer a new distributed K-mutual exclusion algorithm based on tokens, which makes use of K tokens and allows K concurrent entries in the crucial zone (CS). This method significantly outperforms the K-mutual exclusion techniques created by Makki et al. [21], Srimani and Reddy [20], and Raymond [19] in terms of the number of messages and time to reach the critical zone.

4.1 Distributed K-Mutual Exclusion Proposed Method

The system is thought to have N nodes, numbered 1 to N. Each node is only allowed to have one open request for entry into the crucial zone at any given moment. It is believed that each node is fully linked. It is expected that the network and nodes are trustworthy. Additionally, it is anticipated that the network will transmit messages on each channel in FIFO1 order. In our algorithm, node t holds a token t, where 1 = t = K. At first, a predetermined number of K nodes hold the K tokens. Each node maintains a pointer array, where pointer[i] starts out with the value I because node I is the first node to contain the token i. The illustration below shows this. A node storing the token serves as the root of a tree in the forest, which is defined by the pointer array for token i. As a result, there are K tree topologies in a system with K tokens. The method for K-mutual exclusion is derived by enhancing Trehel and Naimi's 1-mutual exclusion algorithm [11]. For K = 1, our strategy is anticipated.



Token-1	Token-2
Pointer[1]=1 for Node 1	**Pointer[2]=2 for Node 1**
Pointer[1]=1 for Node 2	**Pointer[2]=2 for Node 2**
Pointer[1]=1 for Node 3	**Pointer[2]=2 for Node 3**
Pointer[1]=1 for Node 4	**Pointer[2]=2 for Node 4**

Fig: Initialization with a Token Tree

The Data structure associated with these token and each token has a corresponding data structure, which is transferred from one requesting node to the next along with the next token. The data structure will be as:

The token-queue, a FIFO queue, contains the names of the nodes to whom the token must be transmitted in FIFO order.

TOKEN QUEUE

For Token 2

No. of identifiers	3	5	7	6	8	9	11	14
Tags	2	2	---	4	4	---	1	1

Head of Queue **Tail of Queue**

Fig: Token Queue with associated Tag

Token changed pointer: Maintain a "tag" in the token-queue for each node identification. There may be some missing tags (—). The image displays a common token queue and its associated tag below.

The algorithm employs one of three message types:

- REQUEST message
- TOKEN message
- INFORM message

Proposed algorithm

Below is a presentation of the algorithm. The algorithm employs five steps to keep K-mutual exclusion in place. The ENTRY CS method is launched whenever a node wants to enter the critical section. The method EXIT CS is used when a node exits the crucial section. For the three different types of messages stated above, the other three procedures act as message handlers.

The algorithm's pseudo-code is shown first, then the processes are explained. Remember that "I" in these processes refers to the node doing the procedures.

```
Procedure Entry CS:
 {
 if (token = FALSE) then
{
 Choose token t using some heuristics;
send REQUEST(I,I,t) to pointer[t];
 waiting_for_token := t;
 wait until HOLDING becomes TRUE;
}
Enter Critical Section
}
 Procedure Exit_CS:
{
 dest := First node on the token-queue;
if (dest 6= NULL) then
{
 Token_id := NULL;
token := FALSE;
 pointer[t] := last nulltag(t);
 /* pointer[t] is set equal to the last node whose tag is null If no such node exists then pointer[t] is set equal to the first node in the token-queue */
send TOKEN(I,t) to dest;
}
 else /* dest = NULL */
```

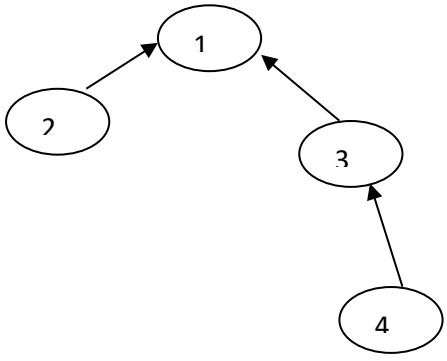
```
send INFORM(I,t) message to any v nodes;
}
 Procedure Handle_REQUEST(X,Y,t):
/* X denotes the adjacent node which sent the REQUEST message and Y denotes the node where the request originated and t denotes the token being requested */
{
 if (token = TRUE) and (holding = TRUE) then
{
 enqueue Y into the token-queue;
if (token_id != t) then
set tag of Y equal to I;
 else
set tag of Y equal to NULL;
}
 else if (token = TRUE) and (holding = FALSE) then
{
 if (token_id != t) then
 set tag of Y equal to I;
 else
set tag of Y equal to NULL;
send TOKEN(I, token_id) to Y;
pointer[token id] := Y;
token := FALSE;
token_id := NULL;
}
 else if (waiting for token = t) then
enqueue Y into the node-queue;
else
 send REQUEST(I,Y,t) to pointer[t];
}
 Procedure Handle_TOKEN(t):
{
 if (waiting for token != t) then
{
pointer[waiting for token] := tag for node I on token-queue;
/* To maintain proper pointer for the token that was originally requested */
Append the node-queue to the token-queue;
For all the nodes that were in the node-queue, set their tags
(in token changed pointer) equal to pointer[waiting for token];
}
 else
{
 Append the node-queue to the token-queue;
For all the nodes that were in the node-queue, set their tags equal to NULL;
}
 dequeue node I and its tag from the token-queue;
waiting_for_token := NULL;
 token := TRUE;
 holding := TRUE;
 token_id := t;
 pointer[t] := I;
}
 Procedure Handle INFORM(Y,t):
 /* Y is the node that has token t */
{
if (waiting for token != t)
 pointer[t] := Y;
}
```

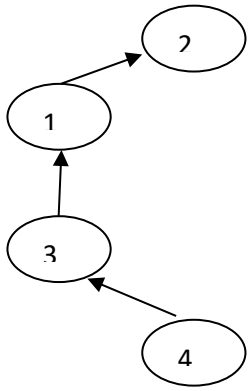
The procedures are explained below:

Entry_CS: When node I wish to enter the critical region, this procedure is called. Node I immediately move onto the critical part if it holds a token. If not, it selects a token using a heuristic where I = t = K and sends a request for that token to the node pointed to by pointer[t]. In Figure, node 4 sends a REQUEST message to node 3 if it wants to request token 1.

Exit_CS: This procedure is carried out after a node leaves the CS. Node I should own the token t. If the token queue is not empty when node I finishes the critical area, it transmits the token t to the node x at the front of the queue (of token t). The last token-queue node whose tag is retained null is what is set as the pointer[t] of node I. Pointer[t] of node I is set to node x, which is at the front of the token queue, if none of the tag held are null. Node I transmit token 2 to node 3 and sets pointer [2] equal to 9, the final node on the token-queue whose tag is held null, if node I holds token 2 with the token-queue given in Figure token tree setup. Node I would set pointer [2] to 3, which is the identity of the node at the top of the token-queue, if none of the tag held were null. The current node keeps the token if there are no requests on the token queue and notifies other nodes selected at random through INFORM messages that it has token t. This is done

in order to decrease the typical distance between a node and a token, which can lower the typical amount of messages needed for each CS entry. The quantity is an algorithmic input parameter. The algorithm's performance can be changed by adjusting v.





TOKEN-1	TOKEN-2
node 1: Pointer[1] = 1	**node 1: Pointer[2] = 2**
node 2: Pointer[1] = 1	**node 2: Pointer[2] = 2**
node 3: Pointer[1] = 1	**node 3: Pointer[2] = 1**
node 4: Pointer[1] = 3	**node 4: Pointer[2] = 3**

Fig : A typical token Tree

Handle REQUEST(X; Y ; t):
This function is called whenever a node receives a token request. Y is the node seeking the token t, and X is the node that sent the request to node I. The response node I gives to the request depends on its current state.

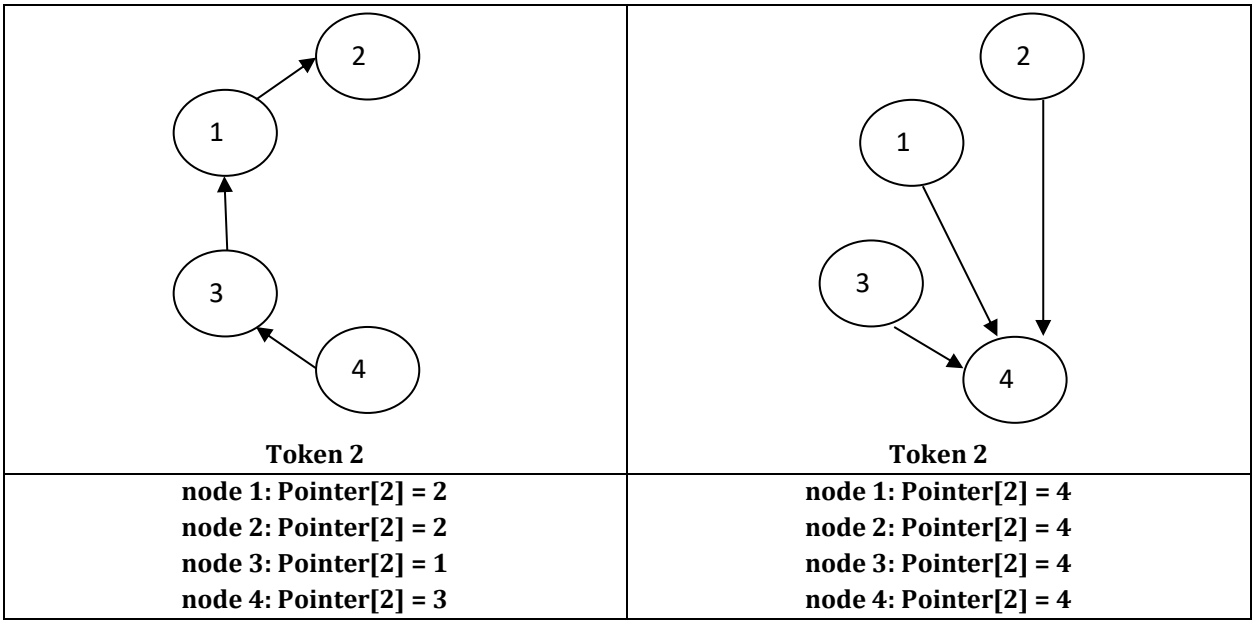
Case 1: node I does not possess a token and is waiting to enter the CS. The action taken by node I depend on the token requested by node I.

- **case a:** If node I is waiting for token t then, Y is stored in the node-queue.
- **case b:** If node I is waiting for token p (p 6= t) then, REQUEST(I ; Y ; t) is sent to pointer[t] and pointer[t] set to Y .

Case 2: node I does not possess a token and is not waiting to enter the CS. The request is forwarded to pointer[t] and pointer[t] is set equal to Y. Referring to Figure below, if node 3 receives a request from node 4 for token 2, node 3 forwards the request to node 1 (as pointer[2] = 1) and changes pointer[2] to 4.

Case 3: node I possess a token and is not in the CS. Action taken by node I depends on the token present at node I.

- **Case a:** If node I possesses token t then, it sends the token to node Y , with Y inserted into the token-queue and sets pointer[t] to Y . The tag for node Y is set to null. An example is illustrated in Figure 6(a) where node 3 possesses token 2 and receives a request for token 2 from node 4.
- **Case b:** If node I possesses token p (p 6= t) then, it sends token p to node Y with Y inserted into the token-queue and sets pointer[t] to Y . The tag (in token changed pointer) for node Y is set to I . An example is illustrated in Figure 6(b) where node 3 possesses token 2 and receives a request for token 1 from node 4.



Token 2	**Token 2**
node 1: Pointer[2] = 2 **node 2: Pointer[2] = 2** **node 3: Pointer[2] = 1** **node 4: Pointer[2] = 3**	**node 1: Pointer[2] = 4** **node 2: Pointer[2] = 4** **node 3: Pointer[2] = 4** **node 4: Pointer[2] = 4**

Fig. Forest structure for token 2 before and after a request from node 4

Case 4: node I possesses a token and is in the CS. Action taken by node I depends on the token present at node I.

- **Case a:** If node I possesses token t then, it adds the request of node Y to the token-queue of token t. The tag in token changed pointer for node Y is set to null.
- **Case b:** If node I possesses token p (p 6= t) then,it adds the request of node Y to the token-queue of token p. The tag in token changed pointer for node Y is set to I .

The four cases above cause the \forest" structure of the token to change dynamically.

Handle TOKEN (t):

This routine is executed when a node receives a token. Before entering the CS. node I checks if token t is the same as the token it requested. The action taken by the node depends on the token received.

Case 1: If node I had requested token p (p 6= t)then, pointer[p] is set to the tag (in token changed pointer) of node I at the head of the token-queue. Node I sets pointer[t] equal to I and appends the requests in the node-queue to the token-queue. The tags of the nodes were on the node-queue, are set equal to pointer[p] of node I. Referring to Figure Token-queue with its associated tag, if node 3 receives the token 2, but had requested token 1, node 3 sets pointer[1] to 2, which is in the tag of node 3.

Case 2: If node I had requested token t then, no special action is taken by node I . In this case, the tag of I in the token-queue of token t will be null. Node I sets pointer[t] equal to I and appends the requests in the node-queue to the token-queue. The tags of the nodes, that were on the node-queue, are set to null. In both cases, node I dequeues its identifier and tag from the head of token-queue.

Handle INFORM(Y; t):

When the node receives an INFORM message from node Y, this function is carried out. When this message is received, pointer[t] is set to Y. As the most recent information regarding the location of token t is made available via INFORM messages, this aids in reducing the distance between a node and a token.

5. Result and conclusion

For the above proposed algorithm further I will gives the proof of the proposed algorithm and give the deadlock free with the starvation free. Also give s the simulation result for the proposed algorithm

to boost the speed of a distributed system where sites only communicate by sending messages over a network and do not share a common memory. This might enhance the algorithm's performance under light loads. A performance test of our approach on parallel message passing systems like nCube and Maspar would be interesting as well.

References

[1] M. Raynal, Algorithms for Mutual Exclusion. Cambridge, MA: MIT Press, 1st ed., 1986.

[2] G. Ricart and A. K. Agrawala, “An optimal algorithm for mutual exclusion in computer networks," Comm. ACM, vol. 24, no. 1, pp. 9-17, January 1981.

[3] M. Maekawa, “A p N algorithm for mutual exclusion in decentralised systems," ACM Trans. Comp. Syst., vol. 3, no. 2, pp. 145-159, May 1985.

[4] B. A. Sanders, “The information structure of distributed mutual exclusion algorithms," ACM Trans. Comp. Syst., vol. 5, no. 3, pp. 284-299, August 1987.

[5] M. Singhal, “A dynamic information-structure mutual exclusion algorithm for distributed systems," in International Conf. Distributed Computing Systems, Newport Beach, CA, pp. 70-78, June 1989.

[6] I. Suzuki and T. Kasami, “A distributed mutual exclusion algorithm," ACM Trans. Comp. Syst., vol. 3, no. 4, pp. 344{349, November 1985.

[7] G. Ricart and A. K. Agrawala, “Author's response to `On mutual exclusion in computer networks' by Carvalho and Roucairol," Comm. ACM, vol. 26, no. 2, pp. 147-148, 1983.

[8] M. Singhal, “A heuristically-aided algorithm for mutual exclusion in distributed systems," IEEE Trans. Computers, vol. 38, no. 5, pp. 651-662, May 1989.

[9] M. Mizuno, M. L. Neilsen, and R. Rao, “A token based distributed mutual exclusion algorithm based on quorum agreements," in International Conf. Distributed Computing Systems, Arlington, TX, pp. 361-368, 1991. 73

[10] K. Makki, N. Pissinou, and Y. Yesha, “A new token based distributed mutual exclusion algorithm," in International Conf. Distributed Computing Systems, Pittsburgh, Pa, pp. 164-169, 1993.

[11] M. Trehel and M. Naimi, “A distributed algorithm for mutual exclusion based on data structures and fault tolerance," in 6th Annual International Phoenix Conference on Computers and Communications, Scottdale, AZ, pp. 35-39, 1987.

[12] J. M. Bernabeu-Auban and M. Ahamad, “Applying path compression techniques to obtain an effecient distributed mutual exclusion algorithm," in Lecture Notes in Computer Science, vol. 392, pp. 33-44, 1989.

[13] D. Ginat, D. D. Sleator, and R. E. Tarjan, “A tight amortized bound for path reversal," Information Processing Letters, vol. 31, pp. 3-5, April 1989.

[14] K. Raymond, “A tree-based algorithm for distributed mutual exclusion," ACM Trans. Comp. Syst., vol. 7, no. 1, pp. 61-77, February 1989.

[15] M. L. Neilsen and M. Mizuno, “A dag-based algorithm for distributed mutual exclusion," in International Conf. Distributed Computing Systems, Arlington, TX, pp. 354-360, 1991.

[16] T. Woo and R. Newman-Wolfe, \Hu man trees as a basis for a dynamic mutual exclusion algorithm for distributed systems," in International Conf. Distributed Computing Systems, Yokohama, Japan, pp. 126{133, June 1992.

[17] H. Koch, “An eient replication protocol exploiting logical tree structure," in Digest of papers: The 23rd Int. Symp. Fault-Tolerant Comp., Toulouse, France, pp. 382-391, June 1993.

[18] S. Huang, J. Jiang, and Y. Kuo, “k-coteries for fault-tolerant k entries to a critical section," in International Conf. Distributed Computing Systems, Pittsburgh, Pa, pp. 74-81, 1993.

[19] K. Raymond, “A distributed algorithm for multiple entries to a critical section," Information Processing Letters, vol. 30, no. 4, pp. 189-193, February 1989.

[20] P. K. Srimani and R. L. Reddy, “Another distributed algorithm for multiple entries to a critical section," Information Processing Letters, vol. 41, no. 1, pp. 51- 57, January 1992.

[21] K. Makki, P. Banta, K. Been, N. Pissinou, and E. Park, “A token based distributed k mutual exclusion algorithm," in IEEE Proceedings of the Symposium on Parallel and Distributed Processing, Arlington, TX, pp. 408-411, December 1992.