A Review of various Mutual Exclusion Algorithms in Distributed Environment

Article *in* International Journal of Computer Applications · November 2015

DOI: 10.5120/ijca2015907089

3 authors:

Nisha Yadav
The IIS University
1 PUBLICATION **3** CITATIONS
SEE PROFILE

Sudha Yadav
Amity University
3 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Sonam Mandiratta
Amity University
1 PUBLICATION **3** CITATIONS
SEE PROFILE

International Journal of Computer Applications (0975 – 8887)
Volume 129 – No.14, November2015

A Review of various Mutual Exclusion Algorithms in Distributed Environment

Nisha Yadav
Department of computer science Amity University, Haryana

Sudha Yadav
Department of computer science Amity University, Haryana

Sonam Mandiratta
Department of computer science Amity University, Haryana

ABSTRACT

In computer science, mutual exclusion (MUTEX) refers to a way of making sure that if one process is using shared modifiable data or resources then the other processes will be excluded from doing the same thing at the same time. A number of mutual exclusion algorithms are available in the literature, with different performance metrics and with different techniques. The Selection for a “good” mutual exclusion algorithm is a key point. These mutual exclusion algorithms can be broadly classified into token and non-token based algorithm. This paper surveys the algorithms which have been reported in the literature for Mutual exclusion in distributed systems and their comparison.

Keywords

Mutual Exclusion (MUTEX), Critical Section (CS), Timestamp.

1. INTRODUCTION

A distributed system is an assemblage of computers that are geographically separated and do not share memory and clock. The processes running on these computers converse with one another by exchanging messages over communication channels. The major benefit of distributed system is resource sharing and cost effective. Mutual exclusion (MUTEX) is a primary issue in distributed computing systems; this issue arises in distributed systems (DS) whenever concurrent access to shared resources by numerous sites is involved. MUTEX states that concurrent access of processes to a shared resource or data is serialized to protect the integrity of data. when a process is accessing a shared resource/data then it is said to be in a CS. In distributed mutual exclusion, the requirement is to serialize the access to CS in the absence of shared memory and common clock. It ensures that action performed by a user on a shared resource must be atomic [9]. ME in a distributed system states that only one process is permitted to execute the critical section (CS) at any given time. The design of distributed MUTEX algorithms is complex because these algorithms have to handle changeable message delays and partial knowledge of the system state. Mutual Exclusion algorithms are categorized as [12]:

- Centralized Algorithm
- Distributed algorithms

Under Distributed algorithms, there is further categorization:

- Contention based solutions
- Control based solutions

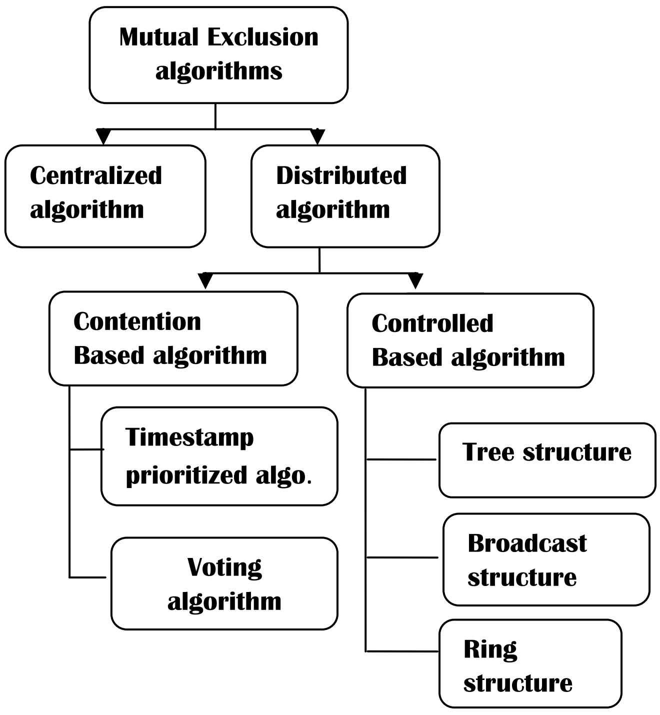

Mutual Exclusion algorithms
Centralized algorithm
Distributed algorithm
Contention Based algorithm
Controlled Based algorithm
Timestamp prioritized algo.
Voting algorithm
Tree structure
Broadcast structure
Ring structure

Fig.1.1: Classification of Mutual Exclusion Algorithms

1.1 System model [9]

The DS comprises of n (P1, P2, P3……, PN) distinct sites (processes or computers), which converse with each other by message passing over connected network. The messages consume finite but random time to reach at the receiving servers. All processes in the DS are assigned distinct identification numbers (PID). If any process requires executing the CS, then it sends request message to every other process. It enters the CS if all process sends reply.

1.2 Requirements of Mutual Exclusion Algorithms [9]

An ME algorithm should assure the subsequent properties:

1. **Safety property:** It ensures that at any instant, only one process can execute the CS. It is a necessary property of a MUTEX algorithm.
2. **Liveness property**: This property states the nonexistence of deadlock and starvation. Two or more sites should not repetitively wait for messages that will never turn up. In addition, a site must not wait forever to execute the CS while other sites are again and again executing the CS. That is, every requesting site should get an opportunity to execute the CS in finite time.

3. **Fairness:** Fairness in the context of MUTEX means that each process gets a reasonable opportunity to execute the CS. The fairness property normally means that the CS execution requests are executed in order of their arrival in the system (the time is determined by a logical clock) [2, 3].

2. CLASSIFICATION OF MUTEX ALGORITHMS [12]

2.1 Centralized Algorithm [9, 12, 14]

As its name indicates, there is one coordinator which handles all the requests to access the shared resource/data. Every process takes permission to the coordinator, if coordinator will give permission only then a particular process can enter into CS. Coordinator will maintain a queue to keep all the requests in order.

Algorithm

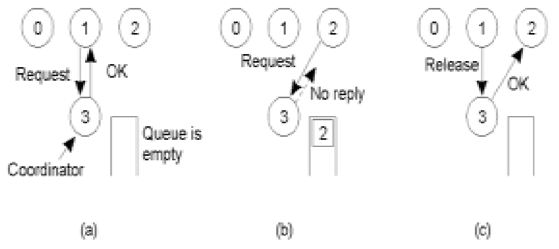


Fig.2.1: Working of Centralized algorithm

a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted because queue is empty and no pending request is there so coordinator will give permission.

b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply because P1 is not exited from CS till now.

c) When process 1 exits the critical region, it tells the coordinator, after that Coordinator will give permission to P2.

Advantages

- Fair algorithm; follow FIFO order for to give permission.
- Easy to implement
- Scheme can be used for general resource allocation.

Shortcomings

- Single point of failure, No fault tolerance.
- Confusion between No-reply and permission denied.
- Performance bottleneck because of single coordinator in a large system.

2.2 Distributed algorithm [13, 15, 16]

In this scheme, there is no coordinator. Every process asks to other process for permission to enter into CS. These algorithms divided into two parts

1. Contention(Non-Token) based algorithms
2. Controlled (Token) based algorithms

2.2.1 Contention (Non-Token) based algorithms [9, 12, 18]

In this type of algorithms, sites converse with a set of other sites to choose who should execute the CS first. These algorithms also divided into 2 parts:

1. Timestamp based
2. Voting scheme

Two basic Timestamp-based Contention algorithms are:

LAMPORT'S ALGORITHM [1, 9, 17] Lamport was the first who designed a distributed MUTEX algorithm on the basis of his logical clock concept. This algorithm is a non-token based scheme. Non-token based protocols use timestamps to order requests for CS. Request message Contain following:

- Identifier (Machine id, process id)
- Name of resource
- timestamp

Timestamp is a distinct integer value which is given by the operating system to the sites when they produce requests for CS. Timestamp is monotonically increased every time when a request is arrived. Smaller timestamp requests have higher precedence rather than large timestamps requests. In lamport's scheme, for a site Pi, request set Ri= {P1, P2, P3…….Pn}. It comprises of all those sites from which Pi must require authorization before entering the CS. Every process maintains a queue of awaiting requests for entering CS in ascending order of timestamps. This algorithm assumes that channels are FIFO.

Algorithm:

Requesting the critical section:

When a process wants to enter into CS, it takes the subsequent steps:

1. Enters its request in its own queue (ordered by time stamps).
2. Sends a request to every node.
3. Wait for replies from all other nodes.
4. When another site receives this request message, it sends a timestamp reply message to the requesting site and keeps this request in its own request_queue.

Executing the critical section

A site can enter into CS when these two conditions are satisfied:

[L1]: Pi has not received a message with timestamp larger than (tsi , i) from all other sites.

[L2]: Pi's request is at the top of request_ queue_i.

Releasing the critical section

1. Upon exiting the critical section, it removes its request from the queue and sends a release message to every process.

2. Upon receiving release message, then other sites removes the related entry from its own request_queue.

3. If own request is at the head of its queue and all replies have been received, enter CS.

Performance

- **Message complexity:** (N-1) number of messages necessary for requesting CS. (N-1) number of messages required for reply. (N-1) number of messages necessary for release. Total 3 (N-1) numbers of messages required in heavy load as well as in case of lightly loaded.
- **Synchronization delay:** Average message delay for sending a message from one process to another process. T time takes place in synchronization.

RICART-AGRAWALA ALGORITHM [2, 9]

Ricart-agrawala algorithm is an expansion and optimization of Lamport's protocol. This algorithm is also for MUTEX and it is a non-token based algorithm. This algorithm combines the RELEASE and REPLY message of lamport's algorithm and decreases the complexity of the algorithm by (N-1).

In this algorithm, there is a request set Pi= (P1, P2, P3……Pn). It comprises of all the sites from which a site needs to acquire authorization before entering to CS. The Algorithm proceeds as follows.

Requesting the critical section

1. When a site desires to execute into CS, it sends a request along with its timestamp to all sites. This message includes the site's name, and the current timestamp of the system according to its logical clock.
2. Upon reception of a request message, another site will immediately sends a time stamped reply message if and only if:
    - The receiving process is not currently interested in the critical section.
    - The receiving process desires to enter into CS but its own timestamp value is higher than requesting site.
    - Otherwise, the receiving process will suspend the reply message. This means that a reply will be sent only after the receiving process has completed using the CS itself.

Executing the critical section

- Requesting site enters its CS only after receiving all reply messages.

Releasing the critical section

1. Upon exiting the critical section, the site sends all deferred reply messages.
2. In this algorithm, all the CS requests are executed in their timestamp order.

Performance

- **Message complexity:** (N-1) number of messages required for requesting CS. (N-1) number of reply messages merges with release. Total 2 (N-1) numbers of messages required in heavy load as well as in case of lightly loaded.
- **Synchronization delay:** Average message delay for sending a message from one process to another process. T time takes place in synchronization.
- Reply messages are combined with release messages because reply messages are send to only those sites whose timestamp is greater than executing site.

Disadvantage: Failure of a node – May result in starvation.

VOTING SCHEMES[12]: In this scheme, If there are n no. of processes and suppose process p1 wants to enter into critical section then it will send a request message to (n-1) processes and more than n/2 processes send reply message then p1 will enter into CS.

Requestor

- Send a request to all other processes.
- Enter critical section once REPLY from a majority is received
- Broadcast RELEASE upon exit from the critical section.

Other processes

- REPLY to a request if no REPLY has been sent. Otherwise, hold the request in a queue.
- If a REPLY has been sent, do not send another REPLY till the RELEASE is received. [1]

Drawback: Possibility of deadlock.

2.2.2 *Controlled (TOKEN) BASED ALGORITHMS [12, 22]*

In token-based algorithms, a unique token is shared among the sites. A site is allowed to enter its CS if it possesses the token. Token-based algorithms use sequence numbers instead of timestamps to distinguish between old and current requests. Generally do not assume FIFO message delivery. Also their Proof of correctness is trivial.

Issues: how to find and get the token. This distinguishes various algorithms.

These algorithms are divided into 3 parts on the basis of structure in which process are connected:

Ring Structure [21]: In this structure all processes are connected in the form of a ring in which each process is assigned a position as shown in the Following Fig. The ring positions may be allocated in numerical order of network addresses or some other means. Way of ordering is not much important , while the important thing is that each process knows who is next in line after itself.

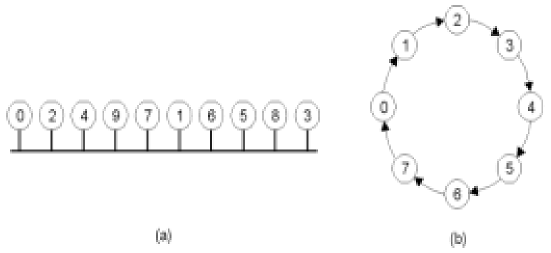


Fig.2.2: Ring Structure of processes

Advantages: simple, deadlock-free, fair.

Disadvantages:

- The token circulates even in the absence of any request (unnecessary traffic).
- Long path (O(N)) – the wait for token may be high.

Broadcast Structure (Suzuki-Kasami Algorithm) [3, 9, 28]

In Suzuki-kasami algorithm, if a site wants to enter the CS and in case if it do not possess the token, it broadcasts a

REQUEST message for the token to all other sites. A site which possesses the token sends it to the requesting site upon the receipt of its REQUEST message. If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of its CS.

Token Consist of:

– **Q:** Queue of the requesting processes, at most n.

– **LN [1...n]:** array of integers, LN[j] is the sequence number of the request that Pj executed most recently.

Data Structures:

• **REQUEST (j,n):** REQUEST message from Pj for its nth request to enter the CS.

• **RNi[1..N]:** RNi[j] is the largest sequence number in a REQUEST message from Pj received by Pi.

• On receipt of REQUEST (j,n), Pi sets RNi[j] to be max(RNi[j],n).

• If **RNi[j] >n**, the message is outdated.

This algorithm must efficiently address the following two design issues:

(1) How to distinguish an outdated REQUEST message from a current REQUEST message: Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied. If a site can not determined if the request corresponding to a token request has been satisfied, it may dispatch the token to a site that does not need it .this will not violate the correctness, however, this may seriously degrade the performance.

(2) How to determine which site has an outstanding request for the CS: After a site has finished the execution of the CS, it must determine how many sites have an outstanding request for the CS so that the token can be dispatched to one of them.

The first issue is addressed in the following manner: A REQUEST message of site Pj has the form REQUEST (j, n) where n (n=1, 2 ...) is a sequence number which indicates that site Pj is requesting its nth CS execution. A site Pi keeps an array of integers RNi[1..N].where RNi[j] denotes the largest sequence number received in a REQUEST message so far received from site Pj .When site Pi receives a REQUEST(j, n) message, it sets RNi[j]:=max(RNi[j], n).When a site Pi receives a REQUEST(j, n) message, the request is outdated if RNi[j]>n.

The second issue is addressed in the following manner: The token consists of a queue of requesting sites Q, and an array of integers LN [1...N]; where LN[j] is the sequence number of the request which site Pj executed most recently. After executing its CS, a site Pi updates LN[i]:=RNi[i] to indicate that its request corresponding to sequence number RNi[i] has been executed .At site Pi if RNi[j]=LN[j]+1, then site Pj is currently requesting token.

Algorithm:

Requesting the CS:

– If the requesting site Pi does not have the token, it increments its sequence number RNi[i], and sends a REQUEST (i,sn) message to all other sites.

– When Pj receives the message, it sets RNj[i] to max (RNj[i], sn). If Pj has the idle token, it sends the token to Pi if RNj[i] =LN[i] +1.

Executing the CS: Enter CS when gets token.

Releasing the CS: Having finished the execution of the CS, site Pi takes the following actions:

– Sets LN[i] to Rni[i].

– For every site Pj whose ID is not in the token queue, it appends its ID to the token queue if RNi[j] = LN[j] +1.

– If token queue is nonempty after the above update, it deletes the top site ID from the queue and sends the token to the site indicated by the ID.

Performance

- **Message complexity:**Requires 0 messages if the requesting site holds the idle token.N messages otherwise (N-1 broadcast and 1 to send the token).
- **Synchronization delay:** 0 or T based on if the site holds the token at the time of request.
- **No Starvation:** Token request messages reach all other sites in finite time. Since one of these sites posses the token, the request will be placed to the token Q in finite time. Since there are at most N-1 other requests in front of this request, the request will be granted in finite time.

Tree structure (Raymond's Algorithm) [4, 9, 19]

Basically this algorithm uses a spanning tree to reduce the number of messages exchanged per CS execution. The network is viewed as a graph; a spanning tree of a network is a tree that contains all the N network nodes (or sites). The algorithm assumes that the underlying network guarantees the delivery of message. All nodes in the network are completely reliable. A node (or site) needs to hold information about and communicate only to its immediate-neighboring nodes (or sites). Sites (or nodes) are arranged in a logical directed tree. Root holds the token. Edges are directed towards the root node or towards node currently possessing the token. Every site (or node) has a holder variable that points to an immediate neighbor node(or site) on the directed path towards root (Root's holder point to itself). A FIFO queue called request_p that holds its requests for the token, as well as any requests from neighbors that have requested but haven't received the token if request_p is non-empty that implies the node (or site) has already sent the request at the head of its queue toward the holder.

Algorithm:

Requesting for CS:

– Send REQUEST to parent on the tree, provided i do not hold the token currently and its request_p is empty. Then place request in its request_p.

– When a non-root node j receives a request from i:

a) Place request in its request_p.

b) Send REQUEST to parent if no previous REQUEST sent.

– When the root r receives a REQUEST:

a) Place request in its request_p.

b) If token is idle, follow rule for releasing critical section (shown later).

– When a node receives the token:

a) Delete first entry from the request_p.

b) Send token to that node.

c) Set Holder variable to point to that node.

d) If request_p is non-empty, send a REQUEST message to the parent (node pointed at by Holder variable).

Executing the CS:

— Enter if token is received and own entry is at the top of its request_p; delete the entry from the request_p.

Releasing the CS:

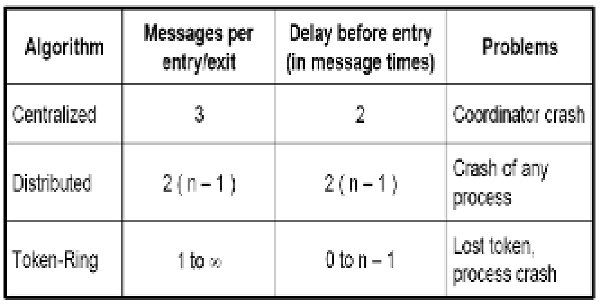
– If request_p is non-empty, delete first entry from the request_p, send token to that node and make Holder variable point to that node.

– If request_p is still non-empty, send a REQUEST message to the parent (node pointed at by Holder variable).

Performance:

- **Message complexity:** Average messages: O (log N) as average distance between 2 nodes in the tree is O (log N).
- **Synchronization delay:** (T log N) / 2, as average distance between 2 sites to successively execute CS is (log N) / 2.
- **Greedy approach:** Intermediate site getting the token may enter CS instead of forwarding it down. Affects fairness, may cause starvation.

Table 1: Comparison of algorithms on the basis of No. of message require for entry/exit, Delay & their problems



Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token-Ring	1 to ∞	0 to n – 1	Lost token, process crash

PERFORMANCE METRICS [9]
The performance is usually calculated by the subsequent four metrics:

Message complexity: The number of messages communication per CS execution by a site.

Synchronization delay: it is a duration time when a site exits the CS and next site enters the CS.

Response time: The time duration a request waits for its CS execution to be over after its request messages have been sent out.

System throughput: The rate at which the system executes requests for the CS**. (System Throughput=1/ (SD+E)**), where SD is the Synchronization Delay and E is the average CS Execution Time.

Low and High Load Performance: We often study the performance of MUTEX schemes under two special loading conditions, which is, "low load" and "high load". The load is determined by the arrival rate of CS execution requests. Under low load conditions, there is seldom more than one request for the CS present in the system simultaneously. Under heavy load conditions, there is always an awaiting request for CS at a site. The MUTEX is very essential for the design of distributed systems. The design of distributed MUTEX schemes is difficult because these algorithms have to deal with irregular message delays and partial knowledge of the system state.

Table 2: Comparative Performance Analysis A comparison of performance (LL= Light Load, HL = Heavy Load)

Algorithm	Resp. Time (LL)	Sync. Delay	Messages (LL)	Messages (HL)
Lamport's	2T+E	T	3(N − 1)	3(N−1)
Ricart-Agrawala	2T+E	T	2(N−1)	2(N−1)
Suzuki-Kasami	2T+E	T	N	N
Raymond's	T(logN)+E	T(log N)/2	Log(n)	4

3. CONCLUSION
In Non-Token based approach requests for access to the critical section are satisfied in the order of their timestamps, therefore fairness is guaranteed. More no. of messages require for Non-token based algorithms in their communications in comparison with the token-based algorithms. No one algorithm is perfect because everyone has their own advantages and disadvantages. Non-Token based algorithms are called permission based algorithms so no token is required to enter into CS. Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next. Tree structure based algorithm uses a spanning tree to reduce the number of messages exchanged per critical section execution. The algorithm assumes that the underlying network guarantees message delivery. All nodes of the network are 'completely reliable. The algorithm provides the following guarantees: Mutual exclusion is guaranteed, Deadlock is impossible, Starvation is impossible.

4. RESEARCH GUIDELINES
- A mutual exclusion algorithm should be fault tolerant. If any site fails due to some network failure or any hardware failure then mutual exclusion algorithm should gracefully handle the situation.

- We need to design the algorithm for NON-FIFO channels.
- It should be adaptive to mobile distribution system.

- Priority should be there. Higher priority site should get chance to access critical section first.

5. REFERENCES

[1] L. Lamport. Time, clocks and the ordering of events in a distributed system. Communications of the ACM, 21(7):558-565, 1978.

[2] G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 24(1):9-17, 1981.

[3] Trehel and M. Naimi. A distributed algorithm for mutual exclusion based on data structures and fault tolerance. In Proc. IEEE 6th International Conference on Computers and Communications, pages 35-39, 1987.

[4] M.G.Velaquez. "A Survey of Distributed Mutual Exclusion Algorithms". Technical Report CS. Colarido state university, September 1993.

[5] P.C. Saxena, J. Rai, A survey of permission-based distributed mutual exclusion algorithms, Computer Standards & Interfaces, Volume 25, Issue 2, May 2003, Pages 159-181

[6] SRIMANI, P.and REDDY, R., "Another distributed algorithm for multiple entries to a critical section," Information Processing Letters, vol. 41, no. 1, jan. 1992, pp. 51-57.

[7] KAKUGAWA, H.; FUJITA, S.; YAMASHITA, M. and AE, T., "Availability of k- Coterie," IEEE Transactions on Computers, vol 42, no. 5, may 1993, pp. 553-558

[8] Pranay Chaudhuri, Thomas Edward "An O(√n) Distributed Mutual Exclusion Algorithm Using Queue Migration1" Journal of Universal Computer Science, vol. 12, no. 2 (2006), 140-159

[9] M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.

[10] I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems, 3(4):344-349, 1985.

[11] L. Lamport, "Time, clocks and ordering of events in a distributed system" Comm. ACM, vol.21, no.7, pp. 558-565, July 1978.

[12] "Distributed Mutual exclusion" ppt. by Rajnitha Shivarudraiah

[13] A. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, Upper Saddle River, NJ, Prentice-Hall, 2003.

[14] Randy Chow,Theodore Johnson "Distributed Operating system and algorithm analysis".

[15] Abhishek swaroop, Awadesh kumar singh "a STUDY BASED ALGORITHMS FOR Distributed mutual exclusion".

[16] K.Raymond,"A distributed algorithm for multiple entries to a critical section",Information processing letter, vol.30, pp.lg9- 193,1989.

[17] L.Lamport ,'A fast mutual exclusion algorithm", ACM Transaction on computer Systems,vol. 5,no. 1,p. 1-11,1987.

[18] D.agarwal,A.El Abbadi,"A token baesd fault tolerant Distributed mutual exclusion algorithm,journal of parallel and distributed computing,24,pp.164-176,1995.

[19] k.Raymond,"a tree based algorithm for distributed mutual exclusion,ACM Transcaction on computer systems,computer standard & interfaces,vol. 21 ,pp.33-50,1999.

[20] M.Mizuno,M.L Neilson,R.rao,"A token based Distributed mutual exclusion algorithm based on quorum agreements.Conference on distributed computing system,pp.361 368,1991.

[21] "Several-tokens Distributed Mutual Exclusion algorithm in a logical ring network" by Ousmane.

[22] I.Suzuki and T.Kasami, "A distributed Mutual exclusion algorithm". ACM Transaction on computer systems. Vol.3,no. 4,pp. 344-349,1985.

[23] KAKUGAWA, H.; FUJITA, S.; YAMASHITA, M. and AE, T., "Availability of k- Coterie," IEEE Transactions on Computers, vol 42, no. 5, may 1993, pp. 553-558

[24] Rahul Garg, Vijay K Garg, Yogish sabharwal "Scalable *algorithms for global snapshots in distributed systems* " ACM 2006.

[25] "Shared memory mutual exclusion exclusion": Major Research trends since 1986 by James H.Anderson and yong-jik kim.

[26] Ichiro Suzuki, Tadao kasmi," A distributed Mutual exclusion Exclusion algorithm".

[27] Trehel and M. Naimi. A distributed algorithm for mutual exclusion based on data structures and fault tolerance. In Proc. IEEE 6th International Conference on Computers and Communications, pages 35-39, 1987.

[28] SUZUKI, I., AND KASAMI, T. An optimality theory for mutual exclusion algorithms in computer networks.In oroceedings of the 3rd international conference on distributed computing systems, IEEE,N.Y.,365-370.

[29] M.G.Velaquez. "A Survey of Distributed Mutual Exclusion Algorithms". Technical Report CS. Colarido.

[30] P.C. Saxena, J. Rai, "A survey of permission-based distributed mutual exclusion algorithms", Volume 25, Issue 2, May 2003, Pages 159-181