RESEARCH ARTICLE OPEN ACCESS

Comparative Study of Mutual Exclusion Algorithms in Distributed Systems

Jijnasa Patil[1], Radhika Naik[2]
Department of Computer Engineering and Information Technology Veermata Jijabai Technological Institute Matunga, Mumbai, India

Abstract
Mutual Exclusion is an important phenomenon in distributed systems. In this paper, we analyze and compare various mutual exclusion algorithms in distributed systems. In permission based mutual exclusion process waits for permission from other processes to enter into a critical section. In token based mutual exclusion, a special message called token is passed over the system and process holding the token can enter into the critical section. We present a comparative study of quorum based, token ring token asking and multiple token algorithms for mutual exclusion in distributed systems.
Index Terms—distributed mutual exclusion, group mutual exclusion, mutual exclusion,

I. INTRODUCTION

In distributed systems multiple processes run concurrently and collaboratively. There are a few numbers of resources as compared to that of processes. So, these resources need to be shared among the processes. A situation may occur quite often that multiple processes want to access the same resource simultaneously. This may cause inconsistency. To prevent this some mechanism must be present to grant access of the shared resource to the processes in mutually exclusive manner. In this paper we study the various mutual exclusion algorithms. In distributed systems, mainly there are two types of algorithms for mutual exclusion, viz., Token based and permission based. In the case of former, a message, called token, is passed between the processes. The tokens are available one per shared resource. The process which has the token is allowed to access the shared resource. When process finishes its critical section, it passes the token to the next process. While in the latter case, a process which wants to access the shared resource requests other processes' permission. There are many ways to grant the permission. One of the methods is quorum based mutual exclusion algorithms. In quorum based mutual exclusion algorithms a process chooses its quorum and waits for each quorum member to grant its request to access the shared resource.

A group mutual exclusion is a special case of mutual exclusion. In this, every shared resource is associated with a type. All the processes which want the same type of shared resource can run concurrently, while processes wanting to access different type of shared resource should run in mutually exclusive manner.

In this paper we have a look at some of the distributed mutual exclusion algorithms such as quorum based token ring token asking and multiple token algorithms.

II. RELATED WORK

There are many ways by which mutual exclusion can be achieved. A centralized approach to mutual exclusion [1] mimics single processor system. One of the processes in the system is elected as coordinator. Other processes request for the resource, the coordinator grants or queues the request. When request is granted by the coordinator process starts using the resource and when done it releases the resource. Although it is very easy to implement has very less message complexity, the dead coordinator cannot be detected. And the central server becomes the bottleneck. A distributed approach is followed by Ricart Agrawala algorithm [1], [7] in which a process which wants the shared resource sends request to all the processes and waits for ok reply from every process. It accesses the resource when it receives ok message from all other processes.

III. MUTUAL EXCLUSION IN DISTRIBUTED SYSTEMS

In distributed systems multiple processes run concurrently and collaboratively. There are a few numbers of resources as compared to that of processes. So, these resources need to be shared among the processes. A situation may occur quite often that multiple processes want to access the same resource simultaneously. This may cause inconsistency. To prevent this some mechanism must be present to grant access of the shared resource to the processes in mutually exclusive manner. In this paper we study the various mutual exclusion algorithms. In distributed systems, mainly there are two types of algorithms for mutual exclusion, viz., Token based

and permission based. In the case of former, a message, called token, is passed between the processes. The tokens are available one per shared resource. The process which has the token is allowed to access the shared resource. When process finishes its critical section, it passes the token to the next process. While in the latter case, a process which wants to access the shared resource requests other processes' permission. There are many ways to grant the permission. One of the methods is quorum based mutual exclusion algorithms. In quorum based mutual exclusion algorithms a process chooses its quorum and waits for each quorum member to grant its request to access the shared resource.

A group mutual exclusion is a special case of mutual exclusion. In this, every shared resource is associated with a type. All the processes which want the same type of shared resource can run concurrently, while processes wanting to access different type of shared resource should run in mutually exclusive manner.

A. Quorum based Distributed Mutual Exclusion Algorithm

A quorum is a subset of nodes or processes. Although nodes and processes are identical, following the convention in, we use the term node specifically when referring to the role of a process as a quorum member. A quorum system C, also referred to as a coterie, for a (traditional) mutual exclusion is a set of quorums satisfying the following properties:

Intersection: $\forall P, Q \in \mathcal{C} :: P \cap Q \neq \emptyset.$
Minimality: $\forall P, Q \in \mathcal{C} : P \neq Q : P \not\subseteq Q$

If a process enters its critical section only after it has successfully locked all nodes in some quorum, then the intersection property ensures that no two processes can execute their critical sections concurrently. The minimality property ensures that no process is required to lock more nodes than necessary to achieve mutual exclusion.

Maekawa's algorithm [2], [3], [6] implements mutual exclusion by using a coterie. Lamport's logical clock is used to assign a time stamp to every request for a critical section. A request with a smaller time stamp has a higher priority than a request with a larger time stamp (ties are broken using process identifiers).

Maekawa's algorithm works as follows:

- When a process wishes to enter a critical section, it selects a quorum and sends a REQUEST message to all the quorum members. It enters the critical section once it has successfully locked all its quorum members. On leaving the critical section, the process unlocks all its quorum members by sending a RELEASED message.
- A node, on receiving a REQUEST message, checks to see whether it has already been locked by some other process. If not, it grants the lock to the requesting process by sending a LOCKED message to it. Otherwise, the node uses time stamps to determine whether the process currently holding a lock on it (hereafter referred to as the locking process) should be preempted. If the node decides not to preempt the locking process, it sends a FAILED message to the requesting process. Otherwise, it sends INQUIRE message to the locking process.
- A process, on receiving an INQUIRE message from a quorum member, unlocks the member by sending a RELINQUISH message as and when it realizes that it will not be able to successfully lock all its quorum members. This is ascertained when a FAILED message is received from one of the quorum members.
- A node, on receiving a RELINQUISH or RELEASED message, grants the lock to the process whose request has the highest priority among all the pending requests, if any.

Maekawa proved that the message complexity of the above algorithm is O(n), where q is the maximum size of a quorum. Further, its synchronization delay and waiting time are both two message hops. The synchronization delay is two message hops because once a process leaves the critical section, another process can enter the critical section after all the quorum members of the former process have received a RELEASED message and the latter process has received a LOCKED message from its quorum members.

B. A Surrogate-quorum-based Algorithm

The group mutual exclusion problem (GME) [2] was first proposed in as an extension to the traditional mutual exclusion problem. In this problem, every request for a critical section is associated with a type or a group. An algorithm for group mutual exclusion should satisfy the following properties:

Group mutual exclusion is at any time, no two processes, which have requested critical sections belonging to different groups, are in their critical sections simultaneously. Starvation freedom is a situation when a process wishing to enter critical section succeeds eventually.

So, an algorithm [2] solving traditional mutual exclusion problem can solve this group mutual exclusion problem indeed, but they are not optimal for this problem. As they consider all critical section s as same. They require that all critical sections must be executed in mutually exclusive manner.

- A node, when sending a LOCKED message to a process, piggybacks all requests currently in its queue, which are compatible with the request by the locking process.
- A process, on receiving a LOCKED message, stores all the requests that were piggybacked on the message. Once it has successfully locked all

its quorum members, it sends an INVITE message to processes that made these requests.

- A process, on receiving an INVITE message for its current request, sends a CANCEL message to all its quorum members. It then enters the forum.
- A node, on receiving a CANCEL message from a process, removes its request from the queue, if it exists.
- Once a follower exits the forum, it sends a LEAVE message to its leader.
- A leader maintains the lock on its quorum members until it has received a LEAVE message from all its followers and has itself left the forum. It then sends a RELEASED message to its quorum members.
- A node, on receiving a RELEASED message from a process, removes all those requests from its queue that it piggybacked on the last LOCKED message that it sent.
- If a leader leaves its forum and has not received a LEAVE message from all its followers, then the leader is called a surrogate-leader.

C. Ricart Agrawala Algorithm

This algorithm [1], [3], [7] is an efficient way of achieving mutual exclusion in distributed systems. When a process want to access the shared resource it sends a message containing a unique identifier, name of the resource and timestamp. When a process receives a request message it compares timestamp and the earliest wins. The algorithm is illustrated as below:

- Process wants to enter critical section:
  Compose message containing:
  Identifier (machine ID, process ID)
  Name of resource
  Timestamp (totally-ordered Lamport)
  Send request to all processes in group
  Wait until everyone gives permission
  Enter critical section / use resource
- When process receives request:
  If receiver not interested:
  Send OK to sender
  If receiver is in critical section
  Do not reply; add request to queue
  If receiver just sent a request as well:
  Compare timestamps: received & sent messages
  Earliest wins
  If receiver is loser, send OK
  If receiver is winner, do not reply, queue
- When done with critical section
  Send OK to all queued requests

D. Token Based Distributed Mutual Exclusion Algorithm

In Token based mutually exclusive systems multiple processes are in some logical structure. A message called token is circulated. Process having the token can access the shared resource. There are two types of token based systems[3], 1) Perpetual mobility and 2) token-asking method. In the perpetual mobility, the token travels from one process to another to give them the right to enter their critical sections exclusively, without paying attention to whether that process needs the token or not. Therefore, additional processing and communication are imposed on the system as overhead, especially in the light load situations in which very few numbers of processes attempting to invoke their critical sections, simultaneously. But the perpetual mobility of the token is very effective on the high load situations. Token-ring algorithm is one of these algorithms.

E. Token Ring Algorithm

In Token Ring Systems [1], [3] nodes are logically ordered in a ring fashion. A special message called token is passed between the nodes one by one. This is shown in Fig. 1. When a process holding the token completes its critical section it passes the token to the next process in the ring and so on.

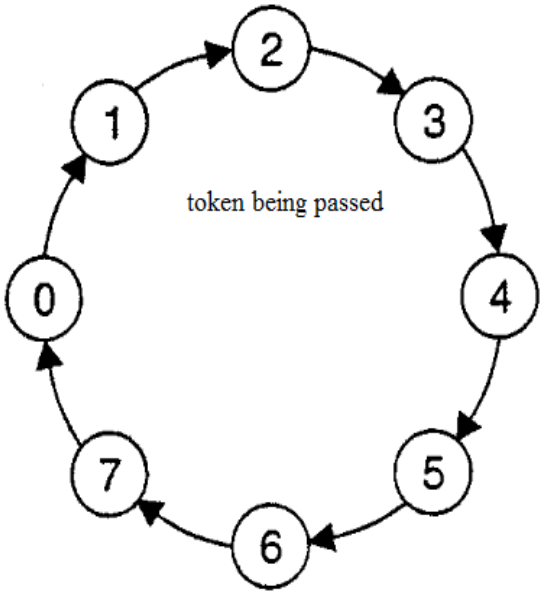

Fig. 1 Token Ring Algorithm

A token ring algorithm is illustrated below:

- Initialization
  Process 0 gets token for resource R
- Token circulates around ring
  From Pi to P(i+1)mod N
- When process acquires token
  Checks to see if it needs to enter critical section
- If no, send ring to neighbor
- If yes, access resource
  Hold token until done

Some of the advantages of the token ring algorithm are 1) It is very simple to understand and

implement. 2) As all the processes wanting to enter into critical section get equal chance, no processes will starve. Along with the advantages there are some disadvantages also: 1) since there are N-1 messages for a token communication overhead may be caused.2) If the token is lost, it is very difficult to regenerate it. 3) If a process holding the token crashes then the token needs to be regenerated.4) this system is not scalable. If number of processes in the system increases, the waiting time per process will also increase.

F. Information based token passing algorithm

In information based token asking method [3], the processes are structured in a wrap around 2-dimensional d x d array of size N2 where N is the number of processes N=d2. There may be three type of processes viz. 1) Token holding process- this is only one process, 2) informed process- processes in the row of token holding process, 3) token requesting process- sends request to the process below it.

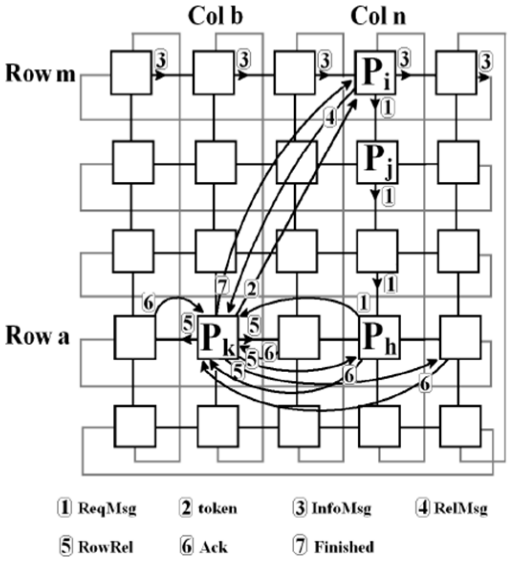

Fig. 2 Information based- based token asking algorithm

By info-based it is meant that from total nodes in a distributed system, some nodes know the current location of the token and forward CS entry requests to the token-holding node, directly. Therefore, in our info-based algorithm, CS entering requests are led to the token-holding process directly through informed-nodes and the token is also sent from the token-holding process to the requesting process directly. As a result, knowing that some nodes have requests to enter their critical sections and the current location of the token are very important concepts in our info-based algorithm.

We assume that in the beginning of the algorithm, Pk is the token-holding process (which is in row a and column b of the wrap around topology) and it is executing its CS. To simplify, assume there is only one non-token-holding process, say process Pi in row m and column n, which is attempting to invoke its CS. The given position of these two nodes and messages exchanged between them in the following scenario is shown in Fig. 2. The request message (ReqMsg) of process Pi for entering its CS is sent to the node below Pi, suppose it is process Pj. If process Pj doesn't know who the token-holding node is, it sends ReqMsg of process Pi to the next node below. This action continues until ReqMsg of process Pi eventually arrives to one of the informed-nodes, a node in row a that knows process Pk is the token-holding node, e.g. process Ph in Fig. 2. Now, ReqMsg of process Pi is sent directly to process Pk. Therefore, up to this step of the algorithm, the request message of process Pi for entering its CS has arrived to the token-holding node. Process Pk, after releasing its CS, sends the token directly to process Pi. Process Pi, after receiving the token, informs all the nodes in its row that Pi is the tokenholding process by an InfoMsg message. Hence, all nodes in row m know that Pi is the token-holding process. In this case, process Pi through sending RelMsg message to process Pk, asks process Pk to inform all nodes in Pk's row that Pk does not hold the token anymore. To do this, process Pk after receiving the RelMsg message, multicasts RowRel messages to all nodes in its row, except itself. Process Pk waits until receiving Ack messages from all these nodes. By arriving each Ack message from any process (say process Pf), process Pk knows for certain that it has received all ReqMsgs sent through process Pf already. When process Pk receives the Ack messages from all these nodes (i.e. all nodes in Pk's row know that Pk is a nontoken- holding process anymore), it sends Finished message to process Pi. Therefore, the responsibility of process Pk in managing the token and CS entering requests is finished. Process Pi after receiving the Finished message, executes its CS and after completing its CS becomes the manager of the token with the power to decide whether remain the idle token holding process or send the token to another node.

This algorithm runs in worst case in terms of message in the case of heavy load. When all the nodes attempt to enter CS, there will not be any downward forwarding of request. Table 1 shows the comparison of token ring and info-based token asking algorithms

G. Multiple Tokens Algorithm

It is based on token ring algorithm. Every process is associated with a unique identifier. There are multiple resources to be used in mutual exclusive manner by the processes. Multiple tokens are circulated in the system simultaneously. A process which wants to enter into CS generates a token and passes in the ring as shown in Fig. 3. When the same token is back it can enter the CS.

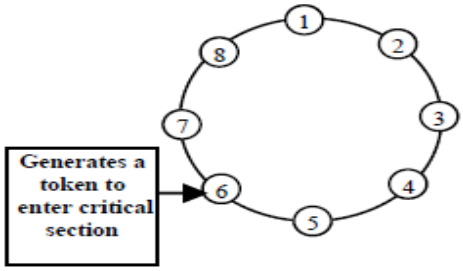

Fig. 3 Multiple Tokens Algorithm

Token structure is shown below:

TOKEN {
TokenID; //includes address of a node
ResourceID; //resource for which the request ismade
Resend; //0 for initial token, 1 for retransmitted tokens
};

Where,
TokenID = (*SeqNum, PID)*
SeqNum = locally assigned unique sequence number
PID = process identifier
HighestSeqNumSeen = stored by each process
T_i^r = token generated by node i for resource r
Queue for each resource
The algorithm [4] works in four steps:
Step-1: A node Ni wants to enter the critical section, CS(r)

- It Generates a token, T_i^r
- And passes it to the next node

Step-2: Any node Nj receives a token for CS(r)

1. If Nj has no intention to enter the CS(r),
   - It replaces the sequence number by HighestSeqNumSeen
   - passes the token to the next node.
2. If Nj is now in the CS(r),
   - it puts the incoming token in Q_i^r .
   - When Nj exits the CS, it releases the tokens to the next node sequentially (if any) from Q_i^r
3. If Nj has already generated a token but not yet received that back it compares the incoming token's priority with it's generated token's priority. If **Pri(T_i^r)<*Pri*(T^r)**, it passes the token to the next node; otherwise it puts the token in r .

Step-3:

- If the node Ni receives its own generated tokenT_i^r,
- i.e. all other nodes allowed T_i^r to enter the CS(r), and then it enters the CS.
- On exiting from the CS(r) it sends the queued tokens to the next node sequentially (if any) and deletes the associated copy and original token.

Step-4:

- If node Ni does not receive its own generated token, T_i^r, within a certain timeout period (because, either token is lost or held by some other died node),
- Ni retransmits the token with the initial priority identifier and Resend field value of 1.
- Upon reception of a token, if its Resend field is found to be 1, a simple query in the request queues is executed to find that token.
- If it is not found, the token is added in the request queue; otherwise it is simply discarded.
- Takes care of lost tokens

It is very easy to see that the message complexity of this algorithm is N-1.

IV. ANALYSIS

Comparison of different mutual exclusion algorithms in distributed systems is given below in TABLE I, where N denotes number of nodes in the system and q denotes number of nodes in the quorum.

TABLE 1: Comparison of token ring algorithm, info-based token asking algorithms and multiple token algorithm

Algorithm	Heavy Load	Light Load
Token Ring	1	O(N)
Information based Token Passing	2	$4\sqrt{N}+1$
Multiple token Algorithm	N-1	N-1
Ricart-Agrawala	2(N-1)	2(N-1)
Maekawa	$5(\sqrt{N}-1)$	$5(\sqrt{N}-1)$
Surrogate Quorum Based	O(q)	O(q)

V. CONCLUSION

This paper analyzes different algorithms for achieving mutual exclusion in distributed systems. The types algorithms are broadly categorized as permission based and token based. Quorum based mutual exclusion is a special case of permission based. And for token based we have described token asking and multiple token algorithms. The comparison of these algorithms is presented. The future scope of this paper is to design a new hybrid algorithm combining the benefits of both permission and token based mutual exclusion techniques.

REFERENCES

[1] Andrew S.Tanenbaum, Maarten Van Steen, "*DISTRIBUTED SYSTEMS, Principals and Paradigms*", Pearson Prentice Hall Pearson

Education, Inc. Upper Saddle River, NJ 07458

[2] Ranganath Atreya, Neeraj Mittal, Member, IEEE Computer Society, and Sathya Peri, "*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, OCTOBER 2007

[3] Peyman Neamatollahi, Hoda Taheri, Mahmoud Naghibzadeh, "*A Distributed Token-based Scheme to Allocate Critical Resources*", IEEE 2011

[4] Md. Abdur Razzaque, Choong Seon Hong, "*Multi-Token Distributed Mutual Exclusion Algorithm*", 22nd International Conference on Advanced Information Networking and Applications

[5] Samad Paydar, Mahmoud Naghibzadeh, Abolfazl Yavari, "*A Hybrid Distributed Mutual Exclusion Algorithm*", IEEE—ICET 2006.

[6] MAMORU MAEKAWA, "*A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems*", ACM Transactions on Computer Systems, Vol. 3, No. 2, May 1985

[7] Glenn Ricart, Ashok K. Agrawala," *An Optimal Algorithm for Mutual Exclusion in Computer Networks*", Communications of the ACM, January 1981, Volume 24, No.1.