Various Mutual Exclusion Algorithms in Distributed Systems

Sindhu Daniel
Department of MCA, Mount Zion college of Engineering, Kadammanitta

Abstract--A mutual exclusion is a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section, a piece of code in which processes or threads access a shared resource. 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 permission 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.

I. INTRODUCTION

The paper proposes a recovery scheme that provides peer-to-peer recovery approach which communicates each member to recover the lost packets. With or without the wired infrastructure, it can establish an instant communication structure for civilian and military applications. Peer-to-Peer recovery tends to evenly distribute recovery overhead to the entire group instead of centralizing at certain nodes. A receiver attempts to recover lost packets with the aid of a random set of members in the group. Collaborative Opportunistic Recovery Algorithm (CORA) achieves the recovery which maximizes efficiency within latency. The contributions are: First, a localized peer-to-peer recovery strategy; Second, a deterministic Cached Packet Distance Vector (CPDV); Third, a tradeoff study between localized recovery benefits versus memory and processing overhead. It extends the collaboration of non-member nodes in order to locate local peers, allows a node to acquire one-hop neighbors caching status and/or CPDV entries with zero transmission overhead. It also reduces the communication overhead caused by recovery traffic and energy consumption

- Centralized algorithms.
- Decentralized algorithms.
- Distributed algorithms.
- Token Ring Algorithms.

A. System model

The paper proposes a recovery scheme that provides peer-to-peer recovery approach which communicates each member to recover the lost packets. With or without the wired infrastructure, it can establish an instant communication structure for civilian and military applications. Peer-to-Peer recovery tends to evenly distribute recovery overhead to the entire group instead of centralizing at certain nodes. A receiver attempts to recover lost packets with the aid of a random set of members in the group. Collaborative Opportunistic Recovery Algorithm (CORA) achieves the recovery which maximizes efficiency within latency. The contributions are: First, a localized peer-to-peer recovery strategy; Second, a deterministic Cached Packet Distance Vector (CPDV); Third, a tradeoff study between localized recovery benefits versus memory and processing overhead. It extends the collaboration of non-member nodes in order to locate local peers, allows a node to acquire one-hop neighbors caching status and/or CPDV entries with zero transmission overhead. It also reduces the communication overhead caused by recovery traffic and energy consumption

B. Requirements of Mutual Exclusion Algorithms

An ME algorithm should assure the subsequent properties:

1. Safety Property: At any instant, only one process can execute the critical section.
2. Liveness Property: This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.
3. Fairness: Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

II. CLASSIFICATION OF MUTEX ALGORITHMS

A. Centralized Algorithm

In centralized algorithm, one process is elected as the coordinator. Whenever a process wants to access a shared resource, it sends a request message to the coordinator stating which recourse it wants to access and asking for permission, if no other process is currently accessing that resource, the coordinator sends back a reply granting permissions.

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.
Fig. 1: Working of Centralized algorithm

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.

B. Decentralized algorithm

- Decentralized algorithm based on the Distributed Hash Table (DHT) system structure.
  - Peer-to-peer
  - Object names are hashed to find the successor node that will store them
- Here, we assume that \( n \) replicas of each object are stored. The resource is known by a unique name: 
  \( \text{name} \) Replicas: \( \text{name}-0, \text{name}-1, ..., \text{name}-(n-1) \). \( \text{name}-i \) is stored at \( \text{succ(\text{name}-i)} \), where names and site names are hashed as before
  - If a process knows the name of the resource it wishes to access, it also can generate the hash keys that are used to locate all the replicas
- Every replica has a coordinator that controls access to it (the coordinator is the node that stores it)
- For a process to use the resource it must receive permission from \( m > n/2 \) coordinators.
- This guarantees exclusive access as long as a coordinator only grants access to one process at a time. The coordinator notifies the requester when it has been denied access as well as when it is granted.
- Requester must “count the votes”, and decide whether or not overall permission has been granted or denied.
- If a process (requester) gets fewer than \( m \) votes it will wait for a random time and then ask again.
- If a resource is in high demand, multiple requests will be generated.

Shortcomings

- It’s possible that processes will wait a long time to get permission
- Deadlock
- Resource usage drops
- More robust than the central coordinator approach and the distributed approaches. If one coordinator goes down others are available.
- If a coordinator fails and resets then it will not remember having granted access to one requestor, and may then give access to another.

C. Distributed algorithm

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

1) Contention (Non-Token) based algorithms

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:** 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.

**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:** 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**: 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.
- **Drawback**: Possibility of deadlock.

2) **Controlled (TOKEN) BASED ALGORITHMS**
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**: 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.

**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)**
In Suzuki-kasami algorithm, if a site wants to enter the CS and in case if it do not possess the token, it broadcasts aREQUEST 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 RN[i][1..N], where RN[i][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 RN[i][j]:=max(RN[i][j], n). When a site Pi
receives a REQUEST (j, n) message, the request is outdated if
RN[i][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 Pj updates LN[i]=RN[i] to indicate
that its request corresponding to sequence number RN[i] has
been executed. At site Pi if RN[i][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 RN[i][j], and sends
a REQUEST (i, n) message to all other sites.
– When Pi receives the message, it sets RN[j][i] to max (RN[j][i],
sn). If Pi has the idle token, it sends the token to Pi if RN[j][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 RN[i]. – For every site Pj
whose ID is not in the token queue, it appends its ID to the
token queue if RN[j][i] = LN[j][i] +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)

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message per entry/exit</th>
<th>Delay before entry/in message times</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Decentralized</td>
<td>$3m_k,k=1,2,...$</td>
<td>2 m</td>
<td>Starvation, low efficiency</td>
</tr>
<tr>
<td>Distribute</td>
<td>$2(n-1)$</td>
<td>$2(n-1)$</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token Ring</td>
<td>$1 \text{ to } \infty$</td>
<td>$0 \text{ to } n-1$</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>

**PERFORMANCE METRICS**

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)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Resp.Tim</th>
<th>Sync.Dela</th>
<th>Messages(LL)</th>
<th>Messages(HL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamport’s</td>
<td>$2T+E$</td>
<td>$T$</td>
<td>$3(N-1)$</td>
<td>$3(N-1)$</td>
</tr>
<tr>
<td>Ricart Agrawala</td>
<td>$2T+E$</td>
<td>$T$</td>
<td>$2(N-1)$</td>
<td>$2(n-1)$</td>
</tr>
<tr>
<td>Suzuki kasami</td>
<td>$2T+E$</td>
<td>$T$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Raymond’s</td>
<td>$T(\log N)+E$</td>
<td>$T(\log N)/2$</td>
<td>$Log(n)$</td>
<td>4</td>
</tr>
</tbody>
</table>

**III. 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.
REFERENCES


[13]. Randy Chow, Theodore Johnson “Distributed Operating system and algorithm analysis”.

[14]. Abhishek swaroop, Awadesh kumar singh “a STUDY BASED ALGORITHMS FOR Distributed mutual exclusion”.

www.ijcsws.com 244