The Message Passing Group Mutual Exclusion: A Review

Abhishek Swaroop*
Department of Computer Science & Engg
G. P. M. College of Engineering
New Delhi 110036 India
abhi_pu1@yahoo.co.in

Awadhesh Kumar Singh
Department of Computer Engineering
National Institute of Technology
Kurukshetra 136119 India
aksinreck@rediffmail.com

Abstract
The group mutual exclusion (GME) problem is an interesting generalization of the mutual exclusion problem. The group mutual exclusion problem deals with two contradictory issues of mutual exclusion and concurrency. The purpose is to allow concurrent access to the processes requesting for the same resource. However, the processes requesting for different resources must access requested resources in mutually exclusive way. GME is also advantageous in the scenarios, when several users share large data object stored on some secondary storage. The users, interested in currently buffered object, may access it concurrently; however, users interested in some other object(s) have to wait. Various solutions for GME problem have been proposed in the literature both for shared memory as well as message passing systems. However, a good review of the problem is still missing in the literature. In order to limit the present exposition, we could accommodate GME algorithms only for the message passing systems. Nevertheless, in addition to token-based as well as non token-based algorithms, two variants of GME problem, namely group-k mutual exclusion and extended group mutual exclusion, have also been covered.

Key words: Group Mutual Exclusion, Algorithms, Concurrency, Token, Quorum

1 Introduction

A distributed system is a collection of independent computers, which appears to it’s user as a single coherent system, and which are capable of collaborating on a task. The distributed systems provide a better price/performance ratio due to the sharing of resources. The problem of mutual exclusion is one of the most fundamental problems in distributed systems. When some resource is shared between several processes, only one process is allowed to use it at one time in. In [2] Joung proposed a generalization of classical mutual exclusion problem called group mutual exclusion (GME). The group mutual exclusion problem has been modeled as congenial talking philosopher (CTP) problem in [3]. In CTP problem there are \( n \) philosophers and \( m \) forums, however, there is only one meeting room. A philosopher may be in any one of the following three states - thinking, waiting and talking. A philosopher interested in a forum may enter the meeting room, if the meeting room is empty or some philosopher interested in the same forum is already in the meeting room, otherwise it has to wait.

*Corresponding author- Abhishek Swaroop, 360/7a, street no.6, Bholanath nagar, Shahdra, Delhi-110032 email- abhi_pu1@yahoo.co.in, Phone-number-91-11-22300003
The GME problem deals with two contradictory issues of mutual exclusion and concurrency. The concept of GME can be applied to a situation, when several users share large data objects stored in secondary memory (such as CD’s) and only one data object can be loaded in the buffer at a time. Users interested in the data object currently loaded in the buffer are allowed to access it concurrently; however, users trying to access different object(s) have to wait. The readers-writer problem can be considered as a special case of GME problem. In order to achieve this we can use a common read session for all processes and a unique write session for each individual process.

Any solution to the GME problem must satisfy following requirements:

- Mutual Exclusion: No two processes requesting for a different session can be in their critical sections concurrently.
- Deadlock freedom: A process attempting to attend a forum will eventually succeed.
- Concurrent Occupancy: If some process P has requested for a session X and no philosopher is currently attending or requesting a different session, then P can attend X, without waiting for any other process to leave the session.

The first solution for GME problem was given by Joung [2] for shared memory model. Other group mutual exclusion algorithms for shared memory are given by Hadzilacos [7], Kean and Moir [8], and Jayanti et al. [18]. The algorithms designed for shared memory model can be mechanically transformed to the algorithms for message passing systems, however, such transformations are generally found not to be efficient.

The GME algorithms for message passing systems can be classified into token-based algorithms and non token-based algorithms. In non token-based algorithms, a process wishing to enter CS, sends its request to some or all other processes in the system and waits for their permission. The concept of logical clocks [1] is used for prioritizing requests. Joung [3] proposed two non-token based algorithms based upon Ricart-Agarawala’s algorithm [19] to solve the GME problem. Some non token-based algorithms, which uses the concept of quorum systems to solve the GME problem are [9,10,20]. In token-based algorithms, a privilege message called token, is shared among processes and only the process possessing the token, can start a session as a captain. However, other processes requesting for the same session may enter as follower. The examples of token based algorithms are [6,12,15,16].

The rest of the paper is organized as follows. In section 2 we discuss performance metrics relevant for group mutual exclusion algorithms in message passing systems, Section 3 contains discussion of non token-based algorithms for group mutual exclusion, Section 4 covers token-based algorithms and concluding remarks are given in section 5.

2. Performance metrics

The performance metrics such as message complexity, waiting time, synchronization delay and average message size, which are used to compare the performance of the algorithms for classical mutual exclusion problem in message passing systems, are also applicable to the group mutual exclusion algorithms. However, concurrency and forum switch complexity are two other performance metrics, which are specifically applicable to group mutual exclusion algorithms.

a) Concurrency – Concurrency can be defined by the maximum number of processes that can attend a session simultaneously. The utilization of resources can be improved if more processes are allowed to share a resource simultaneously. Therefore, a higher degree of concurrency implies better resource utilization.
b) Forum switch complexity- The forum switch complexity is measured by the maximum number of round of passages a process may wait before it can access the requested resource.

A passage by $P_i$ through a session $F$ is an interval $[t_1, t_2]$, where $t_1$ is the time $P_i$ enters the session and $t_2$ the time when $P_i$ leaves the session. Let $S$ be a set of passages through $F$, where $\text{ts} = \min \{ t \mid [t, t'] \in S \}$ and $\text{tf} = \max \{ t' \mid [t, t'] \in S \}$, then $S$ is a round of passage through $F$, if following conditions are satisfied

i) Only those passages which are in $S$, are initiated between $t_s$ and $t_f$

ii) The last passage before $t_s$ and the first passage after $t_f$, are for a session other than $F$.

Forum switch complexity is particularly significant in applications, where changing a session is time consuming, such as applications which require unloading and loading of disk during a session switch.

3 Non token-based algorithms for GME

In non token-based algorithms, a process wishing to enter CS sends its request to some or all other processes in the system and waits for their permission. The logical clocks [1] are used for prioritizing requests of different processes.

3.1 Algorithms based on Riacrt-Agarawala’s algorithm

The first solution of GME problem for message passing systems was given by Joung[3]. He proposed two algorithms $RA1$ and $RA2$ for GME problem based upon Ricart-Agarwala’s algorithm[19]. In $RA1$ a process $P_i$ wishing to attend a session sends the request $\text{Req} (i, SN_i, X)$ to all other processes, where $i$ is the process identifier, $SN_i$ is the sequence number and $X$ is the requested session. The pair $<i, SN_i>$ is the priority of the request. A process $P_i$ upon receiving a request, replies immediately, if its priority is lower than that of $P_i$ or it is not interested in a different session. Otherwise the reply is delayed. A process can enter the session, if all other processes have replied to its request. The message complexity of $RA1$ is $2*(n-1)$ in worst case. $RA1$ exhibits poor concurrency because; no process $P_i$ can access a resource, if there is a higher priority request from a different process $P_j$, which wants to access a different resource (even if some other process $P_k$, having higher priority than $P_j$, is requesting/accessing the same resource, which was requested by $P_j$).

Joung removed this shortcoming in $RA2$ by modifying the entry policy as follows: “No process $P_i$ can attend a session $X$, if there is a higher priority request from some $P_j$ willing to enter a different session, unless there is another process $P_k$ already in $X$”. In $RA2$ two types of philosophers may exists in a session: captain – which starts a session and successor-which attends the session, because, there is a captain already attending the session. A captain can capture a successor process by sending a start message to that process. On receiving a start message a successor process enters the session without waiting for acknowledgements from other processes. The forum switch complexity of $RA2$ is $2*(n-1)$ in worst case. The message complexity of $RA2$ is $2*(n-1)$ in worst case. The simulation results show that $RA2$ shows better concurrency than RA1 because of weakened entry policy.
3.2 Quorum based algorithms

A quorum is a subset of processes. A quorum system \( C \) also referred as coterie is a set of quorums satisfying following two properties:

a) Intersection \( \forall P, Q \in C \rightarrow P \cap Q \neq \phi \)

b) minimality \( \forall P, Q \in C, P \neq Q \rightarrow P \subsetneq Q \)

Many quorum based algorithms exist in the literature for classical mutual exclusion problem [14, 21, 22]. Joung[20] gave the notion of surficial quorum systems to solve the GME problem. Let \( P = \{1, \ldots, n\} \) be the set of processes. An m-group quorum system or cartel \( C=(C_1,C_2,\ldots,C_m) \) over \( P \) consists of \( m \) sets, where each \( C_i \) is a set of subsets of \( P \). Each \( C_i \) satisfies following properties:

Intersection: \( \forall Q_1, Q_2 \in C : Q_1 \cap Q_2 \neq \phi. \)

Minimality \( \forall Q_1, Q_2 \in C, Q_1 \neq Q_2 : Q_1 \subsetneq Q_2 \)

Joung proposed two algorithms based upon Maekawa’s algorithm [] namely Maekawa_M and Maekawa_S. Both of these algorithms use concept of surficial quorum systems. In Maekawa_M a process \( P_i \) which wishes to enter CS as member of group \( g \), chooses a quorum from cartel \( C_g \) and enters CS only after locking all members of the quorum. In order to improve the concurrency, a node is allowed to be locked by more than one processes of the same group. However, in case of arrival of a higher priority request from a different group all granted locks have to be retrieved. The synchronization delay of the algorithm is \( 2T \) (\( T= \)maximum time delay), while the message complexity is \( 3c+3c.\max[g] \). Here \( c=\max \) quorum size, and \( \max[g] \) is the number of processes in the largest group.

In order to reduce the high message complexity of Maekawa_M, joung proposed another algorithm Maekawa_S. In Maekawa_S each process locks quorum members in some fixed order (with increasing process ID’s). Therefore, if a process \( P \) is waiting for a lock \( i \), than every lock held by \( P \) must be from some \( j \) such that \( j<i \). Further every process \( Q \) holding lock \( i \) has either locked all members of its quorum, or is waiting for some \( k \) such that \( k>i \). The number of messages required per CS entry reduces to \( 2c+1 \) in Maekawa_S, however, the synchronization delay is \( c+1 \).

The major drawbacks of Maekawa_M and Maekawa_S are that the quorum system depends on the set of resources and the maximum number of processes that can share a resource is limited to at most \( \sqrt{2n/m(m-1)} \), where \( m \) is the number of resources and \( n \) is the number of processes.

Toyomura et al. [11] presented an algorithm using coteries, that support any finite number of resources, and any number of processes can share a resource simultaneously. A process \( P \) wishing to use a resource; send \( s \) request messages to each process \( P_j \) in an arbitrarily selected quorum \( Q \in C \) and waits for their permission. Permissions may be preempted by inquire message in order to avoid deadlock and starvation. If a process \( P_j \) has granted permission to a process requesting for resource \( g \) than \( P_j \) can grant permission to another process \( P_k \) requesting the resource \( g \), provided that there is no pending request in its queue, whose priority is lower than \( P_k \). The message complexity of the algorithm is \( O(|Q|) \) in best case and \( O(n|Q|) \) in the worst case, where \( |Q| \) is the maximum quorum size.

Attreya and Mittal[1] suggested a quorum based solution for GME problem and introduced the notion of surrogate quorums. They followed the leader-follower approach. A leader process enters the forum by locking all its quorum members. The quorum members send the requests for the same forum, which are stored in their queue, with the locks. On successfully locking all
members of the quorum a leader enters the forum and sends the *invite* message to the processes requesting for the same forum. On receiving an *invite* message the process unlocks all its quorum members and enters the forum as follower. The leader does not release its quorum until, all its followers have left the session. The algorithm has a message complexity of \( O(Q) \), a synchronization delay of \( 3T \), where \( T \) is the maximum message delay. In order to reduce the synchronization delay author’s suggested a modification, in which each follower directly sends *release* message to its quorum members, thus reducing the synchronization delay to \( 2T \). However, for each stale *Invite* message, \( Q \) *release* messages are required, hence increasing the message complexity to \( O(Q^2) \).

Manabe and Park [9] proposed a quorum based solution for extended group mutual exclusion problem, in which a process can request more than one group at a time and can join any one of these groups. The problem can be mapped to the CD juke box example. A process may be interested in some data, which is available in more than one CD, say A and E. If either CD A or CD E is currently loaded than the process may access the data. The number of messages required by Manabe and Park’s algorithm is \( 9|Q| \) per CS request. However, the algorithm avoids unnecessary blocking [9].

### 3.3 Non token-based algorithms for ring networks

Wu and Joung [4] presented two solutions (*CTP-Ring1* and *CTP-Ring2*) for GME problem in ring networks, where each process can communicate directly with its two neighboring processes. In both the algorithms each process maintains a variable \( SN \), storing the maximum sequence number seen by the process. In *CTP-Ring1* a process wishing to attend a session sends its request \( Req(I,Sni,X) \) to its successor \( P_{i+1} \) and waits until its request is returned. The successor \( P_{i+1} \) on receiving the request, forwards it to its successor, if it is not interested in any session other than \( X \) or its priority is lower than the priority of the request. The message complexity of *CTP-Ring1* is \( n \). However, the concurrency of the algorithm *CTP-Ring1* is very poor.

In *CTP-Ring2* a process \( P_i \), wishing to attend a session \( X \), sends its request \( req(i,SN,X) \) to its successor \( P_{i+1} \). However, when the request is returned to \( P_i \), \( P_i \) enters state checking and sends a *conf/capt* message to its successor. \( P_i \) attends the session \( X \), when the message is returned to \( P_i \). When \( P_j \) receives *conf/capt* message it forwards the message, if it is also interested in \( X \) or has a priority lower than \( <i,SN> \); otherwise \( P_j \) detains the message. If \( j \) is also interested in \( X \) and has a priority lower than \( <i,SN> \), then \( P_j \) is captured by \( P_i \) and \( P_j \) also enters the state checking by issuing a *conf* message to its successor. The message complexity of *CTP-Ring2* is \( 2n \) and forum switch complexity is \( O(n^2) \).

### 4 Token-based algorithms for GME

In token-based algorithms a privilege message called token exists in the system and only the process holding the token can start a session. Token-based algorithms are generally faster than non token-based algorithms, produce less message traffic and are not deadlock prone. However, their resiliency to failure is poor. In the following subsections we discuss some major token-based algorithms for GME problem.
4.1 Mittal and Mohan’s algorithm

Mittal and Mohan [6] presented a token based algorithm for GME problem, useful for applications in which a small no of groups are requested more frequently than others. For example in a CD-juke box of 500 CD’s, 5-10 CD’s may be in high demand, or in multiple readers single writer problem, read requests are more common than write requests. Mittal and Mohan’s algorithm is based on Suzuki – Kasmi’s algorithm[17].

In this algorithm two type of tokens, primary token and secondary tokens are used. At a time there can be only one primary token, while there may be multiple secondary tokens. A process holding primary token, is allowed to issue secondary tokens to other processes requesting the same session. The primary token stores the number of secondary tokens issued. If the process wishing to enter CS holds the token of the same type, it enters CS. Otherwise it sends request to all other processes. A process holding a secondary token, when learns about conflicting pending requests, it sends release message to all other processes after coming out of its CS. The process holding the primary token after coming out of its CS, selects next session and passes primary token to one of the processes having pending request of the selected type. The next request is selected based upon the age of the requests and the number of requests for a particular session.

The message complexity of the algorithm is \((2n-1)\) messages / CS, where as amortized message overhead is \(O(1)\). Synchronization delay of the algorithm is \(T\) and waiting time is \(2T\). The maximum concurrency of algorithm is \(n\).

4.2 Token-based algorithms for ring networks

Cantarell et al. [5] presented a token based algorithm for GME problem in unidirectional ring networks. The algorithm uses a token to open and close the session only. The token is initiated by a process and contains the ids of the current session the next session. The first process entering a session \(X\) is called the leader of the current session and it advertises other processes that session \(X\) is open. The token may be in any one of the following states: Closed with No leader (CNL), Open with no request (ONR), Open with request (OR), Closed with request (CR), Open with no Leader (ONL).

When a process \(P\) requesting for session \(X\), receives the token from its predecessor, it becomes the leader and sends token \((X, \text{NULL})\) to tell that session \(X\) is open (state= ONR). Each process requesting session \(X\) may enter the critical section concurrently. When a process say \(Q\), requests for a session \(Y\) \((Y \neq X\) ) the state of token changes from ONR to OR. When process \(P\), leader of session \(X\) receives the token in OR state, it closes the session and changes token state from OR to CR. When a process, say \(R\) receives the token in CR state, \(R\) can be in critical section. In that case, \(R\) holds the token and releases the token on exiting from the critical section. When process \(P\) (leader of session \(X\) ) again receives the token in CR state, it closes session \(X\) and changes the state of the token from CR to ONL, meaning that session \(Y\) is now open. The main advantage of the above mentioned algorithm is that it does not require process id and maintains no data structure for implementing any queue. The size of the message is bounded and is \(2\log(m+1)\) bits. The worst case message complexity is \(O(n^2)\) per resource request and zero messages in the best case. This algorithm is starvation free. However, in this algorithm the sessions are not opened in a FCFS manner and the synchronization delay is quiet high.
David Lin et al. [12] announced a token based GME algorithm for ring networks, which provides an upper bound to the number of messages and to the message size also. The message complexity and forum switch complexity of the algorithm is $O(n)$ while the time complexity of the algorithm is $O(n^2)$.

4.3 Other algorithms

Q.E.K. Mamun and H. Nakazato’s [12] algorithm has the message complexity between $\theta$ and $n+2$. The algorithm is starvation free and the maximum concurrency of the algorithms is $n$. Another interesting characteristic of the algorithm is that the concurrency, throughput and waiting time can be regulated by adjusting the time period for which a session is declared. However, the algorithm assumes the availability of the synchronized logical clocks among processes and also requires the information about the duration for which the requesting process needs the resource.

O. Thiare, M. Gueroui and M. Naimi. [15] presented a token-based algorithm for GME problem, using the concept of rooted tree. The number of messages required is between $\theta$ and $m$, where $m$ is the number of sessions in the system. The average message complexity of the algorithm is $O(\log m)$. The maximum concurrency is $n$. However, the synchronization delay of the algorithm is high because, the request message follows the path from the requesting process to the root process.

J.R. Jiang [13] proposed a token based algorithm for the group $k$-mutual exclusion algorithm problem in distributed systems. The proposed algorithm does not use process identifiers for process identification. The delay of the algorithm is $O(n^2)$ and forum switch complexity is $O(n)$.

5 Conclusions

The group mutual exclusion problem has attracted a lot of researchers because of its applicability to various practical situations. Many algorithms for message passing systems to solve GME problem exists in the literature. Various variants of GME problem have also been proposed by the researchers like group-k mutual exclusion, extended group mutual exclusion, and group mutual exclusion with bounded capacity. However a good survey of solutions of GME problem in message passing system was missing. In the present paper we tried to cover the message passing group mutual exclusion problem and its variants. Due to the limitation of the space detailed discussion of the algorithms was not possible, however we presented brief description of important token-based as well as non token-based algorithms for GME problem in message passing systems.

References


9. Y.Manabe, and J.Park, “ A quorum based extended group mutual exclusion algorithm without unnecessary blocking”, In proceedings of 10th international conference on parallel and distributed systems(ICPADS’04), 2004
15. Q.E.K Mamun, H.Nakazato, “A new token based group mutual exclusion in distributed systems” In the proceedings of the fifth international symposium on parallel and distributed computing (ISPDC’06), 2006
16. O.Thiare, M.Gueroui, and M.Naimi, “ Distributed group mutual exclusion based on client/servers model” In the proceedings of the seventh international conference on parallel and distributed computing, applications and technologies(PDCAT’06), 2006.