ROLE PROCESSES FOR SHARING A RESOURCE BETWEEN A GROUP OF PROCESSES

Karina M. Cenci
Laboratorio de Investigación en Sistemas Distribuidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253, Bahía Blanca, Argentina
kme@cs.uns.edu.ar

Jorge R. Ardenghi
Laboratorio de Investigación en Sistemas Distribuidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253, Bahía Blanca, Argentina
jra@cs.uns.edu.ar

ABSTRACT
Distributed systems are in continuous growth and new requirements emerge. Initially, in multiprogramming, the mutual exclusion problem was the aim to solve. Then, different extensions and variations come up, like k-mutual exclusion, and group mutual exclusion. In this paper, the proposal is based on an extension to group mutual exclusion (GME) where processes join a group with a role (shared, exclusive) in each stage. The properties that must guarantee a solution to GME are: mutual exclusion, bounded delay, progress and concurrency. For this extension, it requires a new property: role mutual exclusion. A general model to solve the problem is composed of two players: groups and processes. This model can be applied to the different communication mechanisms (shared memory, messages). Examples of implementations based on messages, and shared memory are presented. The shared memory proposed solution is based on an adaptive bakery algorithm.

KEY WORDS
Mutual Exclusion - Group Mutual Exclusion - Concurrency - Distributed Systems

1 Introduction
Applications use resources to give users service. These applications may require the exclusive use of resources. The use of protocols that guarantee mutual exclusion provides a solution to this problem. Several authors proposed different options using the shared-memory or message-passing model ([12], [3], [15], [5], [12]). In distributed systems, applications need resources too, but there may be two different options: some processes compete and some processes collaborate to give users service. There may also be another situation where a resource can be shared by processes with common property, i.e., they belong to the same group or they will not be in conflict while using the resource. Processes with different properties must use the resource in exclusive mode. For example, database applications require a mutual exclusion property for data inserting operation and concurrency property for data selecting operation. This situation is similar to the readers-writers problem (multiple readers-single writer). This type of problem is solved by using protocols that guarantee group mutual exclusion (GME). Properties of mutual exclusion and concurrency are important at the time of the design.

The GME problem was first introduced by Joung [10]. The solution presented was an asynchronous algorithm for shared memory parallel computer system. Several quorum-based algorithms [11] [19] [13] have been proposed for asynchronous message passing. The Manabe-Park [13] algorithm prevents the unnecessary blocking, defined as the case in which two processes are prevented from entering a critical section simultaneously even if they are capable of doing so. Singh-Su [16] proposed a solution to the region synchronization problem (such as mutual exclusion, group mutual exclusion, readers/writers) using messages and satisfying the property of absence of unnecessary blocking.

In this paper, section 2 presents previous solutions to GME with priority. Section 3 introduces concepts of the model of two actors that are used in the different solutions. Section 4 shows the characteristics of the group mutual exclusion where processes join a group with a role (shared, exclusive) (GME-RP). In section 5 a model to the GME-RP is shown. Section 6 and 7 present different implementations for this new situation. Section 8 shows some comparisons of the algorithms. Finally, section 9 draws some conclusions.

2 Previous Work
Mittal and Mohan [14] proposed a token-based algorithm of GME, with two kinds of token: primary (only one at all time) and secondary (multiple). The algorithm is derived from Suzuki and Kasami’s [17] token-based algorithm for mutual exclusion.

A token has a group associated with it and it can only be used to access the critical section of that group. A process can access the critical section in a group if it holds a token -primary or secondary- of that group. At that moment, a process p_i requires access to a critical section in a group, and checks whether it has a token with the same group. If it has a token with this condition, the process can access the critical section without delay. Otherwise, it sends a REQUEST message to all processes in the system.

To select the next primary token holder, first it determines the group for the next session. Then, one of the processes that has requested a critical section of the selected
group is chosen to become the next primary token holder. All other processes with pending request for the selected group are issued secondary tokens.

The concept of priority is used to enhance the resource utilization, the algorithm does not allow individual processes to assign level (priority) to their request. The selection for the next token holder uses a priority-based scheme. This scheme selects the next group with the maximum number of pending requests in the token queue for minimizing the waiting time. Priority schemes may lead to starvation. To avoid starvation, the algorithm uses aging techniques. It measures the number of sessions that have been initiated since the request was added to the queue.

Synchronization delay and waiting time is measured in terms of number of message hops rather than in terms of time. Message complexity is of 2n-1 messages per request for critical section (CS). It has low synchronization delay at most \( t \) and low waiting time of at most \( 2t \). Maximum currency is \( n \).

Swaroop-Singh [18] propose a token based scheme, with a unique token, a captain process (only one at all time) and follower processes. A captain process is responsible for the session initiation and sending a start message to other processes, called followers, requesting for the same session in order to allow them to enter the CS.

A process \( P_i \) wants to attend a session of group \( X \) with priority \( Z_i \) and does not have the token, it sends its request to all members in its request set and waits for the token or a start message. If it receives the token, \( P_i \) initiates the new session as a captain. Otherwise, it receives a start message and it enters the CS as a follower.

A priority queue is associated with the token to store the pending requests. The same priority is ordered in a FCFS approach. To remove the possibilities of request starvation, the priority level of all entries in the token queue are incremented by one before sending the token to the next captain. The lowest priority level is one and the highest priority level is \( k \). The process having the lowest priority level will also be able to attain the highest priority level after \( k-1 \) session switches.

Exchange Messages: - In the best case, no message needs to be exchanged. - In the worse case, if a process enters as captain, it requires \( n \) messages. If it enters as follower, it requires \( n+1 \) messages.

The maximum concurrency of the algorithm is \( n \). Synchronization delay - the heavy load synchronization delay of the algorithm is \( 2T \) in the worst case and \( T \) in the best case, where \( T \) is the maximum message propagation delay.

In [8], the proposed algorithm is a distributed solution to the problem of group mutual exclusion coordination, considering that the processes require some time to share the resource in a group with a maximum concurrency of \( n \) processes.

### 3 Concepts of the Model

Let be a set of \( n \) processes \( P_0, P_1, ..., P_{n-1} \); a set of \( m \) groups \( G_0, G_1, ..., G_{m-1} \) and a unique, non shareable, resource among the \( m \) groups. The processes may work alone or in cooperation with other processes in a group. Any of the \( n \) processes is able to participate in a group. Only one group at a time is allowed to use the resource.

Initially each process works alone. When the process wants to work in a team, it selects a group. Each process may select any of the different groups with a finite working time in the team. Figure 1 shows an example of the relation between the groups and the processes. Where \( P_1 \), \( P_2 \) and \( P_7 \) are linked to the group \( G_1 \), the latter is active and has the permission to use the resource. That means that all the processes are using the resource concurrently. Processes \( P_0 \) and \( P_8 \) are linked to the group \( G_0 \) that is competing to gain access to the resource.

The model of two players, posed in [6], proposed a general solution to this problem using two players: groups and processes. Figure 2 shows the communication between the players. The processes are active players and the groups are passive players and the relation between the players is temporary. When the player group is activated, the competition to access the resource begins.

The design of a solution for this problem requires an algorithm that satisfies the following requirements.

- **Mutual Exclusion**: if some process is in a group, then no other process can be in a different group using the resource simultaneously.
- **Bounded Delay**: a process attempting to participate in a group will eventually succeed.
- **Progress**: when the resource is available (the critical section is empty), and some groups are waiting, one group gains access to the resource at some later point.
- **Concurrent Entering**: if some processes are interested in
a group and there is no other process interested in a different group, the processes can participate in the group concurrently, without waiting for other processes to leave the group.

Figure 3 shows an example of concurrency among processes. Group $G_1$ is in the critical section; $P_i$, the first associated process to the group, and $P_k$ access together to the critical section. At the moment process $P_j$ selects group $G_i$, the latter is in the critical section and the first associated process ($P_l$) is still working. Then process $P_j$ can access without waiting and work concurrently.

4 Definition of GME-RP

This section presents a variation of GME defined as GME-RP (Group Mutual Exclusion - Role Process), posed in [7].

Let be a set of $n$ processes $P_0, P_1, \ldots, P_{n-1}$; a set of $m$ groups $G_0, G_1, \ldots, G_{m-1}$ and a unique, non-shareable, resource among the $m$ groups. The processes may work alone or in cooperation with other processes in a group. Every time that a process works in cooperation, it selects a role to participate in the group.

The roles are exclusive and shared. The exclusive role means that a process requires some time to use the resource exclusively with respect to other exclusive processes of the same group and concurrently with shared processes. Examples that correspond to this situation are the following: (1) In a virtual class, students participate in shared role in the class and tutors in exclusive role. The virtual class is the group that all members want to join. Some processes access the virtual class (resource) as student role (shared role) and others as tutor role (exclusive role). (2) In a political debate, candidates participate in exclusive role and the public in shared role. The political debate is the group that all members want to join.

Initially, each process works alone. Figure 4 (a) shows this situation. When the process wants to work in a team, it selects a role and a group. Each process may select any of the different groups with a finite working time in the team. The selected role depends on the activities of the process in the selected group for this stage. Figure 4 (b) shows the case in which process $P_k$ is linked to group $G_1$ with rolE (exclusive role).

Figure 4 (c) shows process $P_k$ with rolE and process $P_l$ with rolS (shared role) associated to group $G_1$. Figure 4 (d) shows four associated processes to group $G_1$, where two processes, $P_k$ and $P_l$, have rolE and two processes, $P_i$ and $P_j$, have rolS.

A solution to this problem requires an algorithm that satisfies the requirements of GME: Mutual Exclusion, Bounded Delay, Progress, Concurrent Entering and Role Mutual Exclusion.

Role Mutual Exclusion guarantees that when some process with exclusive role is in a group, no other process with exclusive role in the same group can be using the resource simultaneously.

5 Model GME-RP

This section presents a model to solve the problem of GME-RP. The proposed pattern is based on the model of two players. The behaviour of the players is introduced first and then, a design of an algorithm applying these concepts. The behaviour of the player process is the following:

- When the player process wants to participate in a group, it first specifies its role, time and finally the group.
- The process waits until the group allows the access.
- When the process finishes, it releases the group.

The behaviour of the player group is the following:

- The moment the player group becomes active, a time to use the resource is assigned to it. This time is that of the first process.
- While the player group is waiting to access the resource (entry section)
  - A request from a player process may happen. The request is added to the active list. The group checks the process role of the request.
  - If it is shared, the group compares the duration of the process with its own duration. If it is longer, it then sets the associated time to the maximum duration of the new player process.
  - If it is exclusive, the group adds the duration of the process to its associated time.
- While the player group is using the resource (critical section)
  - A request from a player process may happen. The process role is checked. If it is shared, the duration of this process is compared to the remainder (group duration - elapsed duration). If it is not longer the request is added
to the active list and accepted. Otherwise, this request is pending until the next stage.

If it is exclusive, the duration of this process is added to the sum of the duration of the requests with exclusive role of the active list, and the total is compared with the remainder (group duration - elapsed duration). If it is not longer the request is added to the active list and accepted. Otherwise, this request is pending until the next stage.

When the player group is in a critical section and a new request happens, an acceptance test (1) is done. Where \( \text{tpodur}_{\text{nreq}} \) means the time associated to the process, \( \text{timegroup} \) means the time associated to the group, \( \text{tpouse} \) means the elapsed time, \( \text{Req}_{i,E} \in \text{LP} \) means all the accepted requests with exclusive role to use the resource in this stage.

### 6 Algorithm Message GME-RP

This algorithm was presented in [7]. It was the first solution to GME-RP. It uses messages for the communication. In a distributed environment, the communication time (delay time) has to be considered. A reliable network is required, with an estimated communication time \( tc \), and a finite resource time use. The communication time is necessary to adjust the remainder time, to accept a new player process while the player group is in the critical section or not. We define the following variables:

- \( tc_{i,k} \): Delay estimation of the communication between group \( G_k \) and process \( P_i \)
- \( \text{tpodur}_i \): Process time associated to the group in a stage
- \( \text{role}_i \): Process role associated to the group in a stage
- \( \text{gtpo}_{i,k} \): Group time in a stage

Considering the defined variables, the acceptance test (2) is done. Where \( \text{tpodur}_i \) means the time associated to the process, \( \text{remaindertime}_i = \text{gtpo}_{i,k} - \text{tpouse}_k \), \( \text{Req}_{j,E} \in \text{LP} \) means all the accepted requests with exclusive role to use the resource in this stage.

The process, in each stage, sends two messages to the group and receives one message from the group.

- **Req-Process** \( (G_k, P_i, \text{tpodur}_i, \text{role}_i) \): This message is received from a process. If the group is INACTIVE then it changes its state to ACTIVE and adds the request to the request list and adjusts the group time (\( \text{gtpo}_{i,k} \)) to the process time (\( \text{tpodur}_i \)).

If the group is ACTIVE it requests the lock if available. If the lock is not available the request is delayed. (a) If the priority of the received message is less than the priority of the message the lock has been given to. In this case the request is delayed. (b) If the priority is higher it requests the lock from the group and grants it the highest priority.

- **Rec-Group** \( (G_i, \text{priority}) \) this message comes from group \( G_i \) with the priority \( \text{priority} \).

The messages received from the other groups are:

- **Req-Process** \( (G_k, P_i, \text{tpodur}_i, \text{role}_i) \): This message is received from a process. If the group is INACTIVE then it changes its state to ACTIVE and adds the request to the request list and adjusts the group time (\( \text{gtpo}_{i,k} \)) to the process time (\( \text{tpodur}_i \)).

If the group is ACTIVE it requests the lock if available. If the lock is not available the request is delayed. (a) If the priority of the received message is less than the priority of the message the lock has been given to. In this case the request is delayed. (b) If the priority is higher it requests the lock from the group and grants it the highest priority.

- **Rel-Group** \( (G_i, \text{priority}) \) this message comes from group \( G_i \) with the priority \( \text{priority} \).

The messages received from the other groups are:

- **Rel-Process** \( (G_k, P_i) \): This message comes from group \( G_k \) with the priority \( \text{priority} \).

The variables of group \( G_k \) are the following: **state** (INACTIVE, ACTIVE, CS, EXIT), \( \text{LP} \): keeps information of all linked processes, \( \text{LW} \): keeps information of all linked processes, \( \text{LG} \): keeps information of all waiting requests of lock, \( \text{gtpo}_{i,k} \): keeps the time to use the resource. The group communicates with the processes and with other groups. The messages received from the group are:
concurrently in CS. At x+2, a new request from process P_6 with exclusive role (rolE) and tpodur=4 arrives. Since the sum of the time of all exclusive requests is higher than (gtpo_k - tpouse_k - t_{6,k}) and the group is in the CS the process P_6 has to wait for the next stage.

The messages among the groups correspond to the competition to gain access to the resource. The algorithm uses messages to obtain permission from the other groups. Each group has an associated quorum (set of groups) to request permission of access. To select the quorum, we use the Maekawa method [12]. The size of the quorum is \( \sqrt{m} \), where \( m \) is the number of groups. When the group obtains all the permissions the resource can be used and this is informed to its associated processes.

6.1 Correctness and Complexity

In this section, we show the correctness of the proposed algorithm. The algorithm satisfies the properties of mutual exclusion, progress, concurrent entering and role mutual exclusion.

**Theorem 6.1** Role Mutual Exclusion satisfied the proposed algorithm.

For role mutual exclusion to be achieved, at most one process with exclusive role should enter the critical section at any time. If group G_k is in CS and allows process P_i with exclusive role to access, then it waits until P_i finishes to allow another process (P_j) with exclusive role to access.

**Theorem 6.2** The maximum number of processes associated to a group is \( n \).

When each process makes a request for the same group simultaneously, all the requests are added to the active queue. When the group grants the locks of its quorum, it can access all the processes with shared role and one process with exclusive role concurrently. If there is only one request with exclusive role then it can access the \( n \) processes concurrently.

**Theorem 6.3** The proposed algorithm ensures bounded delay.

Suppose a process P_i makes a request to group G_k and is waiting indefinitely. This means that:

(a) Another group (G_l) stays indefinitely in the critical section. This situation would happen if the arrival of all new processes are accepted. This is not possible because, when a new request arrives and G_l is in critical section, the group performs the acceptance test. If the test is fine, the group accepts the request for this stage. Otherwise, it waits for the following stage.

(b) G_k waits indefinitely to access the critical section. Each group request has an associated priority (G_k, priori). This priority will eventually be the highest and grants the access to the critical section.

The complexity of an algorithm can be measured using different topics, like the number of access to shared memory, the delay time between entries in the critical section and the number of exchanged messages. The election of the measure depends on the type of the algorithm. For this algorithm the complexity is measured in function of the number of messages required. Let \( q = |S_k| \). In the best case: If the group has one associated process, it will require \( 3+3(q-1) \) messages; if it has \( l \) associated processes, it will require \( 3l+3(q-1) \) messages. With the maximum number of processes associated, \( n \) requests for the same group simultaneously, the algorithm requires \( 3n + 3(q-1) \) messages. If in average, each group has to yield once, the number of messages required is \( 5(q-1) \). If each group has to yield the permission at most \( p \) times, then it requires \( 3l+3(q-1)+ 2p(q-1) \) messages with \( l \) associated processes.

7 Algorithm Memory GME-RP

The shared memory algorithm is based on the adaptive bakery algorithm [1]. An adaptive algorithm, the worse-case
\[
\left\{
\begin{array}{ll}
\text{(a) Shared Role} & tpodur_i \leq \text{remaindertime}_k - tc_{i,k} \\
\text{(b) Exclusive Role} & \sum_{R \in G_p} tpodur_j + tpodur_i \leq \text{remaindertime}_k - tc_{i,k}
\end{array}
\right.
\]

**MESSAGE ACCEPTANCE TEST**

\[w = \text{chosen weight} \]
\[gt p o u s e (i) = \text{findlidlpo}(\text{list}, i) \]
\[\text{state}(i) = \text{ACTIVE} \]
\[\text{join}(i) \]
\[\text{choosing}(i) = \text{true} \]
\[S = \text{getSet()}; \text{number}(i) = 1 + \max_{j \in G} \text{number}(j) + w \]
\[\text{choosing}(i) = \text{false} \]
\[S = \text{getSet()} \]
\[\text{for each } j \in S, j \neq i \text{ do} \]
\[\text{waitfor} (\text{choosing}(j) = \text{false}) \]
\[\text{waitfor} (\text{number}(j) = 0) \lor ((\text{number}(j)) < (\text{number}(j))) \]
\[\text{gt p o u s e}(i) = \text{assigntime}(\text{list}, i) \]
\[\text{state}(i) = \text{CS} \]

\[\text{... Critical Section} \]
\[\text{waitfor}(\forall j : 1..n, \text{list}(j) \neq \langle ..., \text{encs}, ..., \rangle) \lor \]
\[\langle \text{list}(i) \rangle \neq \langle ..., \text{wait}, ..., \rangle \]
\[\text{state}(i) = \text{INACTIVE} \]
\[\text{number}(i) = 0 \]
\[\text{leave()} \]

**Figure 8. Behaviour of Player Group**

... and can join the group in this stage. Otherwise, it has to wait for the next stage.

In the same group, Processes with role access one at a time. Functions add and remove also require exclusion access.

The behaviour of the player group is shown in figure 8.

The player group waits until a process activates. When this happens, it assigns a time to use the resource. This time is that of the first process and changes the state to ACTIVE. Then, it begins the competition for the use of the resource. The idea of the competition is simple, it is based on the scheduling of a bakery or bookshop.

The adaptive Bakery solution uses the concept of active set. An active set object supports the following operations [1]:

- **join()**: it is used to join the set.
- **leave()**: it is used to leave the set.
- **getset()**: it returns the current set of active processes. More formally, getset() must return all the processes that have finished join() before getset() has started and did not start leave() during getset(). It must not return any process that has finished leave() before getset() started and has not started join() during getset() either.

An adaptive solution involves the set of processes that...
are interested in the use of the resource (active). After the initialization of the variables, the player group joins the active set. Then, it prepares to obtain the turn, changes the value of choosing to true and gets the active set. Afterwards, it gets the number, this is one plus the maximum number of the groups in the active set and plus a weight. After the player group obtains the number, it changes the value of choosing to false. While it was obtaining the turn, the active set could change. The player group gets again the active set and waits for its turn.

7.1 Correctness and Complexity

In this section, we show the correctness of the proposed algorithm. The algorithm satisfies the properties of mutual exclusion, progress, concurrent entering and role mutual exclusion.

Theorem 7.1 Mutual Exclusion satisfied the proposed algorithm.

Suppose that player process \( P_j \) associated to player group \( G_i \) and player process \( P_l \) associated to player group \( G_k \) access the critical section at the same time. The state of \( G_i \) and \( G_k \) is INACTIVE.

1. \( g_j = G_i \) group selected by \( P_j \), \( g_l = G_k \) group selected by \( P_l \)
2. \( \text{list}(i,j) = \langle 2, \text{wait}, \text{role}_j, \text{tpodur}_j, \rangle \)
3. \( \text{list}(k,l) = \langle 2, \text{wait}, \text{role}_l, \text{tpodur}_l, \rangle \)
4. \( w_i = 0, w_k = 0 \)
5. \( \text{state}(i) = \text{ACTIVE}, \text{state}(k) = \text{ACTIVE} \)
6. \( \text{join}(i), \text{join}(k) \)
7. \( \text{choosing}(i) = \text{true}, \text{choosing}(k) = \text{true} \)
8. \( S_i = \text{getset}(i, k), S_k = \text{getset}(i, k) \)
9. \( \text{number}(i) = 1, \text{number}(k) = 1 \)
10. \( \text{choosing}(i) = \text{false}, \text{choosing}(k) = \text{false} \)
11. \( \text{waitfor}(\text{choosing}(i) = \text{false}), \text{in } G_i. \text{ It is ok.} \)
12. \( \text{waitfor}(\text{choosing}(i) = \text{false}), \text{in } G_k. \text{ It is ok.} \)
13. \( \text{waitfor}(\text{number}(k) = 0) \lor (\text{number}(i), i < (\text{number}(k), k)) \)
14. \( \text{waitfor}(\text{number}(i) = 0) \lor (\text{number}(k), k < (\text{number}(i), i)) \)

In step 13, \( G_k \) is in the second waitfor, and the first condition is false. The second condition checks if the value of \( \text{number}(i), i \) is less than the value of \( \text{number}(k), k \). If it is true, it continues. Otherwise, it waits. In this case, \( \text{number}(i) \) and \( \text{number}(k) \) are equal, so the check continues with \( i \) and \( k \). If \( k \) is less than \( i \), then \( k \) advances. Otherwise, it waits. In step 14, \( G_k \) is in the second waitfor, and the first condition is false. The second condition checks if the value of \( \text{number}(k), k \) is less than the value of \( \text{number}(i), i \). If it is true, it continues. Otherwise, it waits. In this case, \( \text{number}(i) \) and \( \text{number}(k) \) are equal, so the check continues.

The algorithm guarantees bounded delay. In the worst case, a process \( P_j \) has to wait \( m-l \) stages to access the critical section. The property of progress is satisfied because the solution is based on the active set, and the only competitors are the members of this set. The maximum number of concurrency is \( n \).

The algorithm guarantees role mutual exclusion. The player processes with exclusive role are added to a queue. The first process in the queue is allowed to access the critical section, and when it finishes another process with exclusive role can use the resource.

The step complexity for mutual exclusion is considered as follows: each spin-lock on a variable while this does not change is counted as one operation. The steps can require access to local or remote variables. For the complexity only the remote steps are considered. The complexity of the active set functions depends of the implementation. The first solution [1] requires \( O(k^4) \) steps, where \( k' \) is the point-contention of the operation. A more efficient implementation exists which has only \( O(k^2) \) steps [4, 9]. If there is only one member, then the complexity is \( O(1) \). If group \( G_i \) is active and there are \( s \) members in the active set, the order complexity of the entry section is \( O(s^2) \). If only one process is associated with shared role, it requires \( (4+1)l \) steps. If there are \( l \) processes associated with shared role, they require, in the worst case \( (4+n)l \) steps. If only one process is associated with exclusive role, it requires \( (6+1)l \) steps. If there are \( l-l \) processes associated with shared role and one with exclusive role, they require, in the worst case \( (4+n)(l-1)+(6+n) \) steps.

8 Comparisons

For the comparison of the algorithms we consider that a message is equivalent to a step in complexity.

Table 1 shows several cases of comparison. Where \( s \)
represents the point of contention, \( q \) means the number of members of the quorum and \( p \) the number of times a group has to yield the permission.

The memory GME-RP presents higher performance in the case of low contention. This happens because the algorithm is adaptive and the only groups that compete are those actually active. A special case takes place when only one group is active, and the order complexity is \( O(1) \). In this case, the group does not require to read remote variables, all the accesses are to local variables.

Consider Swaroop[18] GME. In the best case, no message needs to exchange. In the worst case, \( n \) messages require the captain and \( n+l \) the follower. The complexity is \( O(n) \). In the case of \( l \) concurrency processes this variable requires \( n + (l-1)(n+1) \) messages.

The message GME-RP presents a better performance with high contention and high concurrency. The number of exchanged messages depends on the quorum and the number of times a group has to yield the permission. The complexity is \( O(3(q-1)) \), and with \( l \) concurrency processes it requires \( 3l + 3(q-1) \) messages.

9 Conclusion

Distributed applications share and compete for resources. Mutual exclusion was the first problem, several variations appeared, like \( k \)-mutual exclusion, \( (h \text{-out-of-} k) \) mutual exclusion, group mutual exclusion and the most recent extension, called group mutual exclusion role process (GME-RP). The model for this problem considers that processes have a role and an associated time to share the group. This should be the time they will work cooperatively in the group in each stage. Designing a solution to this problem requires satisfying the properties of GME and the role mutual exclusion property.

A first solution to GME-RP was proposed using messages for the communication. The groups have been assigned a quorum, that is used in the competition to get permission to access the resource. The algorithm guarantees the properties and in the best case, with \( l \) processes linked, it requires \( 3l + 3(q-1) \) messages. A new solution to GME-RP is presented using shared memory for the communication among processes and groups. The algorithm is based on an adaptive Bakery solution to the problem of mutual exclusion. The solution uses the concept of active set. If there exists only one member in the active set, the step complexity order is \( O(1) \). In general, the step complexity order is \( O(s^2) \). This concept of this solution is very simple. Moreover, if several groups begin concurrently their competition, they can obtain its position considering the maximum number of the active set (in this case all the concurrent groups obtain the same number), plus its own weight. This allows different level of weight or priority at the moment of choosing the position.

References