Journal of Theoretical and Applied Information Technology

© 2005 - 2009 JATIT. All rights reserved.

www.jatit.org

DISTRIBUTED GROUPS MUTUAL EXCLUSION BASED ON DYNAMICAL DATA STRUCTURES

[1]OUSMANE THIARE, [2]MOHAMED NAIMI, [3]MOURAD GUEROUI

[1]Asstt Prof., Department of Computer Science, UGB-UFR SAT, Saint-Louis, Senegal

[2]Prof., Department of Computer Science, University of Cergy-Pontoise, France

[3]Prof., PRiSM Lab, University of Versailles Saint-Quentin, France

E-mail: othiare@dept-info.u-cergy.fr, naimi@dept-info.u-cergy.fr, mogue@prism.uvsq.fr

ABSTRACT

The group mutual exclusion (GME) problem is an interesting generalization of the mutual exclusion problem. Several solutions of the GME problem have been proposed for message passing distributed systems. In this paper we present a new Distributed Group Mutual Exclusion (DGME) based on Clients/Servers model, and uses a dynamic data structure. Several processes (Clients) can access simultaneously to a same opened session (Server). The algorithm ensures that, at any time, at most one session is opened, and any requested session will be opened in a finite time. The number of messages is between 0 and m, where m is the number of session in the network. In the average case, O(Log(m)) messages are necessary to open a session. The maximum concurrency is n, where n is the number of processes in the network.

Keywords: *Group Mutual Exclusion (GME), Client/Server*

1. INTRODUCTION

The mutual exclusion problem states that only a single process can be allowed to access in its critical section (CS). Hence, the mutual exclusion problems plays an important role in the design of computer systems. Several distributed systems are based on asynchronous messages passing, and without global clock.

In the first class Permission-Based Algorithm (PBA) [3][7][10], where all involved processes vote to select one which receives the permission to access the CS. Lamport [9] was the first to design a fully distributed permission-based mutual exclusion algorithm using logical timestamps. In his algorithm each request set is the entire distributed system. Then, if n is the number of processes in the distributed system, the algorithm requires (n-1) request, (n-1) reply, and (n-1) releases. The algorithm requires 3(n-1) messages per critical section execution. Ricart and Agrawala [12] have reduced the number of messages in Lamport's algorithm to 2(n-1). Carvalho and Roucairol's algorithm [5] has further improved the number of messages in Ricart and Agrawala's algorithm by avoiding some unnecessary request and reply messages. They have shown that the number of messages exchanged in their algorithm is between 0 and 2(n-1). Maekawa uses the number of message from O(n) to $\sqrt{n}$.

In the second class, Token-Based Algorithms (TBA) [8][12], in which only one process, holding a special message called the token, may enter the critical section. The dynamical spanning tree is used in [13] to ensure the mutual exclusion. The reversal path permits to reduce the number of messages to Log(n) where n is the number of processes in the network. The performance metrics of the mutual exclusion algorithms are: the average number of messages necessary per critical section invocation, the response time, the fault tolerance. The mutual exclusion algorithm should be starvation-free, and fairness. The reversal path principle is used in [13] to solve the mutual exclusion problem in distributed system without logical time. The average number of messages needed per request is O(Log(n)), where n is the number of processes in networks.

The rest of this paper is organized as follow: the principle of DGME is presented in the section 2. Section 3 describes the computational model assumed and we then present the algorithm. In section 4, we present an example. The section 5

© 2005 - 2009 JATIT. All rights reserved.

gives a proof of liveness and fairness properties. The performance and message complexity are discussed in the section 6. The last section concludes this work.

2. RELATED WORK

The design issues for mutual exclusion between groups of processes have recently been modeled by Joung [11] as the Congenial Talking Philosophers (CTP). In [11], another type of mutual exclusion called Group Mutual Exclusion (GME for short) is presented. In the GME problem, every critical section to the same group can be executed simultaneously. However CS belonging to different groups must executed in a sequential mode with mutual exclusion principle.

An example of GME, described by Joung in [8], is a CD juke box shared by several processes. Any number of processes can simultaneously access the currently loaded CD, but processes wishing to access different CD other than the currently loaded one must wait. In this case, the sessions are the CDs in juke box. Joung introduced the concept of m-group quorum system. Several solutions for GME problem are proposed without shared memory and global clock, where the processes communicate by exchanging message. J. Beauquier and al. [4] presented three new distributed solutions for GME: two solutions based on static spanning tree, and the third solution uses a dynamical spanning tree. In [1], the notion of surrogate-quorum is used to solve the GME, and requires a low message complexity, low minimum synchronization delays is two message hops. The maximum concurrency is n, which implies that it is possible for all processes to the same group. In this paper, we propose a new distributed solution for GME problem based on reversal path. The problem is to design a solution for DGME satisfying the following requirements:

- **Mutual exclusion**: if some process participates an opened session, then no other process participates to a different session simultaneously.
- **Concurrent access**: if some processes are interested to participate to a session, and no process is interested in a different solution, then the process can attend the session concurrently.
- **Fairness**: a process attempting to participate to a session will eventually succeed.

We use the same principle to solve the DGME problem in a distributed system. A process can enter several times in the critical section, while any other session does not is requested. The leader, and root manage one or more sessions, the token is managed by the root.

3. DISTRIBUTED MUTUAL EXCLUSION

3.1 DISTRIBUTED SYSTEM MODEL

The distributed system consists of n distinct failure-free nodes (processors), which communicate with each other by message passing over a connected network. The messages take finite but arbitrary time to reach at the receiving nodes. The order of message through the reliable communication links is FIFO. The algorithm presented in this paper share in their design to following assumptions and conditions for the distributed system environment. All processes in the distributed system are assigned unique identification integers numbers. There is only one requesting process executing at each node. Processes are competing for a single resource. At any time, each process initiates at most one outstanding request for mutual exclusion.

3.2 MUTUAL EXCLUSION PROBLEM

A mutual exclusion algorithm must satisfy the following requirements:

- at most one process can execute its critical section at a given time if no process is in its critical section.
- any process requesting to enter its critical section must be allowed to do so in finite time.

3.3 PRINCIPLE OF THE ALGORITHM

From at any connected graph, we construct a spanning tree. Every node y belongs to the spanning tree, y knows its neighbors in the network, successors and predecessors in the spanning tree. The spanning tree is used to minimize the number of messages exchanged, and to eliminate the duplicate message. Initially, one token is assigned to one node.
When a node x request to enter in its critical section, two cases are possible:

- node x holds the token, in this case, it enters immediately in its critical section without sending a request message.

© 2005 - 2009 JATIT. All rights reserved.

- node x does not hold the token, it sends a request to its successor in its spanning tree, and waits for the token.

When a node y receives the request message sent by the node x, several cases are possible:

- the node y holds the token, and does not use it, then y sends it immediately to the node x.
- the node y is its critical section, then, y records this request in its waiting queue.
- in the other case, y broadcasts this request to successors in the spanning tree excepted the node from which it has received the request.

3.4 MESSAGES OF THE ALGORITHM

Initially, all sessions in the network are connected logically to a tree rooted at one session (Leader). We give the token to the root session. When a session X receives a request to be opened from a process P_i, several cases are possible: the session holds the token, and no other session is waiting for the token, in his case, the session X sends immediately an authorization to P_i. Else, if it exists another session in waiting queue, the request's of P_i is added to a waiting set. Now, we examine the situation where a session X receives a request from a process P_i and does not hold the token, and it does not exist process in its waiting set. In this case X sends immediately a request message to the leader, to obtain the token from it. The session X waits for the token.

When a session X receives a request for the token from another session Y. The session X broadcasts a message to all processes in its waiting set, and waits for all release messages. Once, the all release messages received by session X, it sends the token to the next session. When a session X receives the token from another session Y, it sends an authorization messages to all processes in its waiting set. Each process P_i behaves like a customer. Indeed, when P_i wish to take part in session X, it sends an OPEN request to him, then attend its agreement. Once its participation with the session is finished, it sends a message REL to the session.

3.5 LOCAL VARIABLES AT SESSION X

We consider two kinds of messages: messages exchanged between sessions, and messages exchanged between sessions and processes.

- **Messages exchanged between session**:

REQ(x): to obtain the token, this message is sent to leader session.

TOKEN(x): message to denote the permission to open a session x.

- **Messages exchanged between sessions and processes:**

OPEN(x): request sent by a process to open the session x.

OK(x): authorization to participate to the session x by process P_i.

REL(x): message sent by process P_i to session x, to signify that process P_i has closed the session x.

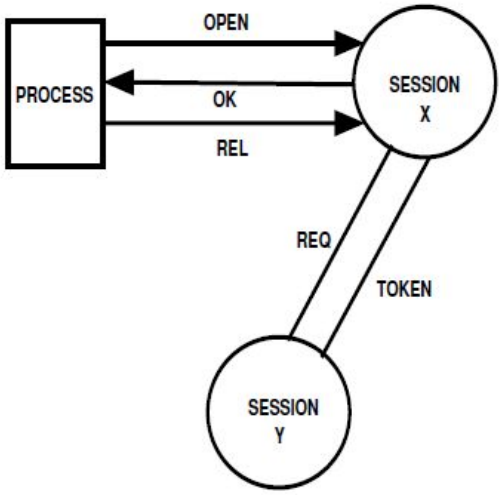

Fig. 1. Messages exchanged between processes and sessions

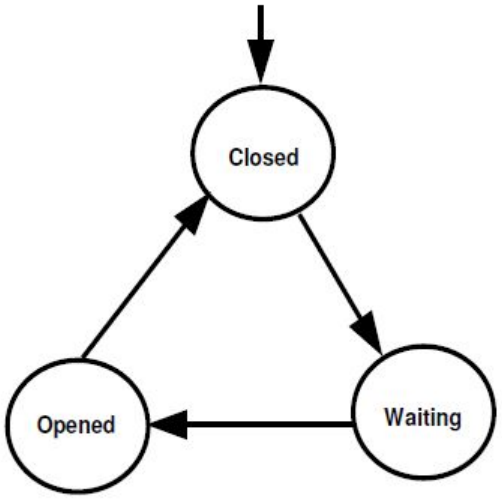

Fig. 1. State session

© 2005 - 2009 JATIT. All rights reserved.

3.6 LOCAL VARIABLES AT SESSION X

$Leader_x$: a pointer which denotes a path from a session to the session at root of the logical rooted tree. Initially, $Leader_x$=Nil if x is the root, and $Leader_x \neq$Nil otherwise.
$Next_x$: pointer which indicates the next session for which the token will be transmitted. Initially, $Next_x$=Nil.
HT_x: Boolean, true if the session x holds the token, false otherwise. Initially, HT_x=true if x is at the root, otherwise, HT_x=false.
RS_x: group of processes waiting to participate to the session x. Initially, $RS_x=\emptyset$ for all session x.
$Nrel_x$: denotes the number of release messages that the session x waits from its group. Initially, $Nrel_x$=0 for all session x.

3.7 LOCAL VARIABLES FOR EVERY PROCESS P_i

X: where X={x, y, z,…} is a dynamic set of m sessions in the network.
Open_Session: Boolean set to true if a session requested by P_i is opened, false otherwise. Initially, Open_Session=false for all processes.
CS_i: denotes the current session requested by process P_i. Initially, CS_i=Nil.

3.8 ALGORITHM DESCRIPTION

The distributed algorithm is based on the following rules: rules of processes and rules of sessions.

Rules of processes

Rule 1::

When a process P_i wants to open a session x

Do

$CS_i \leftarrow$ x

SEND OPEN(P_i) **to** x

Od

Rule 2::

When a process receives OK(x)

Do

Open_Session← true

Od

Rule 3::

When a process P_i releases session x

Do

SEND REL(P_i) **to** x

$CS_i \leftarrow$ Nil

Open_Session← false

Od

To open a session x, every process P_i must execute the following steps:

<Rule 1> **Wait**(Open_Session) <Rule 3>

Rules of sessions

Rule 4::

When session x receives OPEN(P)

Do

If ((HT_x) □ ($Next_x$=Nil)) **then**

SEND OK() **to** P_i

$Nrel_x = Nrel_x + 1$

Else

If (($Next_x$=Nil) □ ($RS_x=\emptyset$)) **then**

SEND REQ() **to** $Leader_x$

$Leader_x \leftarrow$ Nil

EndIf

$RS_x \leftarrow RS_x \cup \{P\}$

EndIf

Od

Rule 5::

When a session x receives REQ(y)

Do

If ($Leader_x$=Nil) **then**

If((HT_x) □ ($Nrel_x$=0)) **then**

SEND TOKEN() **to** y

HT_x= false

Else

$Next_x \leftarrow$ y

EndIf

© 2005 - 2009 JATIT. All rights reserved.

Else SEND REQ(y) **to** $Leader_x$

EndIf

$Leader_x \leftarrow y$

Od

Rule 6::

When a session x receives REL(P)

Do

$Nrel_x \leftarrow Nrel_x - 1$

If $((Nrel_x=0) \square (Next_x \neq Nil))$ **then**

SEND TOKEN() **to** $Next_x$

$HT_x \leftarrow$ false

$Next_x \leftarrow$ Nil

If $(RS_x \neq \emptyset)$ **then**

SEND REQ(x) to $Leader_x$

$Leader_x \leftarrow$ Nil

EndIf

EndIf

Od

Rule 7::

When a session x receives TOKEN()

Do

$HT_x \leftarrow$ true

$\forall P_i \in RS_x$ SEND OK() **to** P_i

$Nrel_x \leftarrow |RS_x|$

$RS_x \leftarrow \emptyset$

Od

Initially, we construct a rooted tree from the network, where the root is a session holding the token and called Leader.

A process sends directly its request to a session, and waits for authorization. Every session manages a group of processes requesting it and a process opens only one session at time if it is the root in a rooted spanning of a given network, and manages all the sessions. When a process x wants to open a session k. Several cases are possible:

- the session k is opened, in this case x can access immediately to CS.
- the session k is closed:
  - x is a root, if all sessions are closed, x opens the session k.
  - x is not root, it sends a request to the root, and waits.

4. EXAMPLE

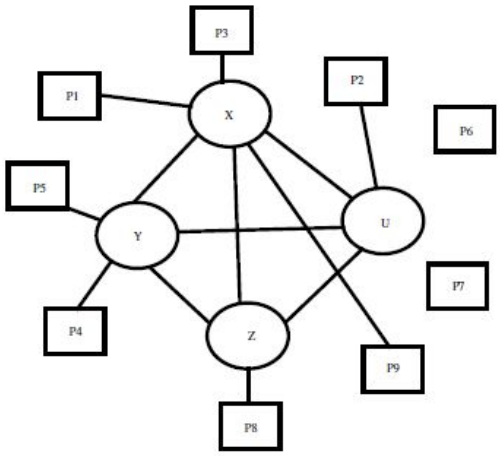

Fig. 3. Graph of sessions and processes.

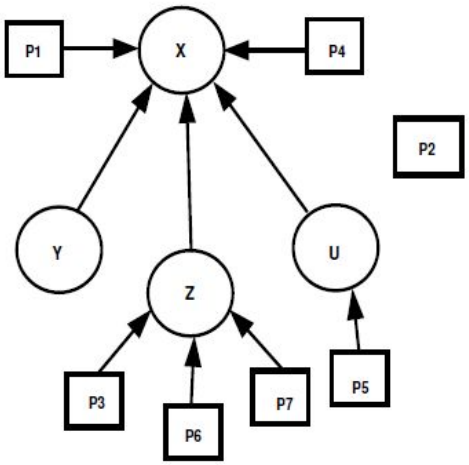

Fig. 4. Initial logical rooted tree.

	U	x	y	z
Leader	X	Nil	x	x
Next	Nil	Nil	Nil	Nil
HT	False	True	False	False
RS	Ø	Ø	Ø	Ø

© 2005 - 2009 JATIT. All rights reserved.

Nrel	0	0	0	0

Fig. 5. Initial global state.

	u	x	y	z
Leader	Nil	u	x	u
Next	Nil	z	Nil	u
HT	False	True	False	False
RS	$\{P_5\}$	$\{P_1\}$	Ø	$\{P_3, P_6, P_7\}$
Nrel	0	1	0	0

Fig. 6. Global state after execution of the algorithm.

Initially, the global state of distributed system is given by Fig. 5. And Fig. 4 represents the initial logical rooted tree.

Now, we illustrate the algorithm by the following scenario:

T_1: Processes P_1 and P_4 want to a session x, and send the message OPEN to session x.

T_2: Process P_5 wants to open the session u, and sends a message OPEN to session u.

T_3: Processes P_3, P_6, and P_7, want to open the session z, they send a message OPEN to z.

T_4: The session u receives the message OPEN for P_5, t is not leader, it sends a message REQ to session x.

T_5: The session x receives a message OPEN from P_4, x sends a message OK to P_1.

T_6: the session z receives a message OPEN from P_6, it sends a message REQ to session x.

T_7: The session x receives a message REQ from the session z. The session z becomes the next session for which, x must send the token.

T_8: The session x receives a message REQ from u, it transmits it to the new leader z.

T_9: The session z receives the message OPEN from the processes P_3 and P_7. The processes P_3, P_6, and P_7 are in the waiting set RS_z.

T_{10}: the session x receives a message OPEN from P_1, the waiting set RS_x contains now, the process P_1.

T_{11}: The session z receives the request of u from session x. The new leader becomes u.

Looking at the overall state after running the algorithm (Fig. 6.), the new logical rooted tree is given by Fig. 7.

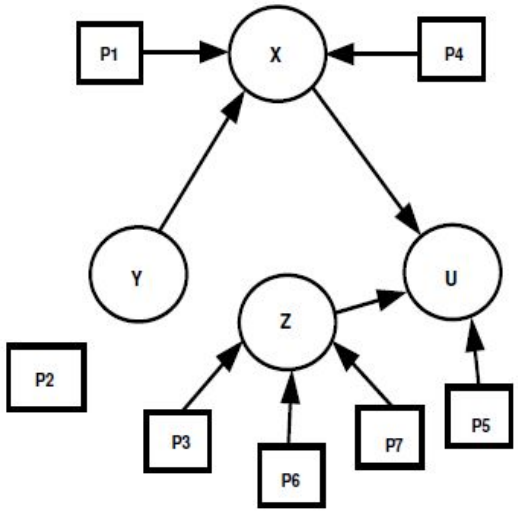

Fig. 5. New logical rooted tree.

5. PROOF

Let x, y and z the sessions in the network.

Lemma 1: *For all session x, y, we have* (HT_x □ HT_y)*=false is invariant.*

Proof

Initially true. When a session x transmits the token to another requesting session y, it sets its local variable HT_x to false (from rules 5 and -) in the algorithm. When the token is in transit to a requesting session, we have HT_x=false for all session x in the network. Upon receiving the token, the session x sets HT_y to true.

Theorem 2: *The algorithm ensures mutual exclusion (at most one session is opened).*

Proof

At any time during execution of the algorithm, only one single token is either located at a session

© 2005 - 2009 JATIT. All rights reserved.

or in transit between two sessions in the network (Lemma 1). Therefore, mutual exclusion is always guaranteed by the protocol.

Lemma 3: *At any time, if we start from any session x and traverse along the chain of pointers $Leader_x$ variables, then we will reach a session y which is the root of the tree.*

Theorem 4: *The algorithm is starvation free.*

Proof

Starvation occurs when one session must wait indefinitely to be opened even though sessions are opened. Consider a requesting session x, and let us examine the reception of a message OPEN from a process P. If x is the leader ($Leader_x$=Nil), it waits for the token. Otherwise, the request of session x is transmitted by arcs corresponding to leader, to a session y for which $Leader_y$=Nil. If y has invoked the critical section, x will be the successor (Next) of session y; otherwise, y sends immediately the token to session x.

We have checked that:

- The request of session x is transmitted to a session y for which $Leader_y$=Nil, within a finite delay. This will by Lemma 3 in which we use the fact that there are no circuits (Lemma 1).
- If y has requested the critical section, x becomes the successors of y, and that fact will allow x to hold the token within a finite delay. That will be proved in Lemma 9 and we shall use the file data structures (Lemma 6 and Lemma 7).

Lemma 5: *The following properties are satisfied.*

- *The mapping leader constitutes a set of rooted trees (a forest)*
- *The set of rooted trees is reduced to a one rooted tree if no request message is in transit between two sessions.*

Proof

Initially, the two points of the Lemma are satisfied.

Let us suppose they are true at some instant.

We assume that x invokes the critical section and let us consider the use of the Rule 1. If x is leader, there is no change made to the set of rooted trees; otherwise a rooted tree is disconnected from the rest. The number of rooted trees is increased by one. Let us examine the receiving of the request message. When a request message is received by a leader session y, the new value of pointer becomes x, y is cut off from the rooted tree in which it was and is attached to the rooted tree of the requesting session. We have a new forest. The number of rooted trees is unchanged. When a leader session y receives a request message, y is connected to session x. The number of rooted trees is decreased by one.

Lemma 6: *A request message is transmitted to a session for which Leader=Nil which a finite delay.*

Proof

Let us that session x requests the critical section without having the token. A request is therefore sent from session x toward the leader of a tree. Consider an instant during the transmission of this request, when it is in transit between session y and z. the arc from y to z has been deleted and the forest is partitioned into two parts: part A which the message comes from, and part B which the request message. No other request message can pass between A and B because no path can be crated before the request has arrived at z.

When the request message has arrived at z, if z is not the leader, the request message is sent from z to v. Part A is increased and part B is decreased and there is always a cut between A and B. Therefore, the request message can never again reach a session of A. We have proved a request message can never received twice times by the same session; i.e., the number of sessions which the request message passes by is less than n.

Otherwise, the delays of transmission are finite. We have proved that the request message will reach a leader within a finite delay.

Lemma 7: *($Leader_x$=Nil)→ ($Next_x$=Nil) is an invariant.*

Proof

True initially, and remains true for all actions of the algorithm.

Lemma 8: *($Leader_x$=Nil) □ (RS_x=Ø) → (HT_x=true).*

Proof

Initially true, only the root session holds the token, and for which the $Leader_x$=Nil. A session lost a token, when it receives a request from another session in this case $Leader_x$≠Nil.

Lemma 9: *($Next_x$≠Nil) ↔ ($Leader_x$≠Nil) □ (RS_x≠Ø).*

© 2005 - 2009 JATIT. All rights reserved.

Proof

Initially true and remains true for all actions of the algorithm.

6. PERFORMANCE

The performance of a distributed mutual exclusion algorithm can be evaluated in terms of a number of metrics. Messages complexity and synchronization delay are two parameters, which can be used to compare the performance of various distributed mutual exclusion algorithm. The message complexity of a distributed mutual exclusion algorithm is the number of messages exchanged by a process per critical section access.

6.1 SYNCHRONIZATION DELAY

One of the performance measurement criteria of mutual exclusion algorithms is synchronization delay. In mutual exclusion, the synchronization delay is the time required after a process exists to the CS and before the next process enters to the CS. In group mutual exclusion it is defined as the time between two consecutive sessions. A synchronization delay is measured in terms of maximum message delay, t. Figure 8. depicts the synchronization delay both in mutual exclusion.

Theorem 10: *The synchronization delays of our algorithm environment Is at most 2t where t is the maximum message delay.*

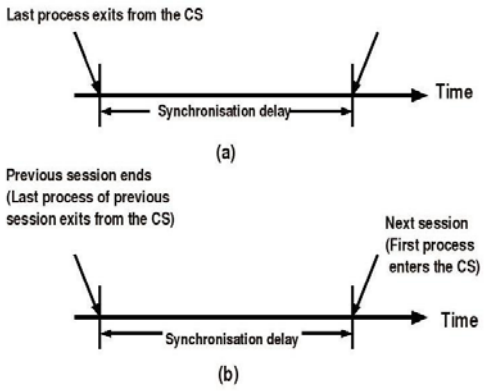

Fig. 6. Synchronisation delay in (a) mutual exclusion (b) group mutual exclusion

6.2 CONCURRENCY

The (maximum) degree of concurrency is defined by the maximum number of session that can still be established while a session is going and some process is waiting for a different session. Obviously, higher degree concurrency implies better resource utilization. According to our algorithm, if all the processes are interested to join the same session simultaneously, one of the processes (the current token holder) will start the session and declares the session to other processes.

Theorem 11: *the maximum concurrency of our algorithm is n.*

The fault tolerance of a distributed mutual exclusion algorithm is the maximal number of nodes that can fail before it become impossible for any node to access its critical session.

The availability is the probability that the critical section can be entered in the presence of failure. In fact, availability of a distributed mutual exclusion algorithm is a measure of its fault tolerance.

Lemma 12: *the number of request message sent by requesting session is bounded by (m-1) where m is the number of sessions in a given network.*

Proof

When a session x holds the token, it does not send any request messages; otherwise, the session x sends the request message to the current root, and waits for the token. From Lemma 4, never session receives the same request twice. Let h be the hight of session x in the rooted tree. We have $0 \leq \sqrt{h} \leq (m-1)$.

Lemma 13: *the number of message necessary to transmit the token from a session x to another requesting session is 1.*

Proof

The token is transmitted directly from a session to another requesting session.

Theorem 14: *the algorithm requires m messages by access to a critical section in the worst case.*

Proof

By the lemmas 7 and 9, a complete topology where every node x has (n-1) neighbors and the radius is equal to 1. The number of request messages sent is (n-1). In a spanning tree every request is sent exactly one time at every node, and the number of message is equal to (n-1).

7. CONCLUSION

In this paper, we have presented a Distributed Group Mutual exclusion algorithm based on clients/servers model. The number of message necessary to satisfy each request is between 0 and m messages in the best case and worst case respectively, where m is the number of sessions in

© 2005 - 2009 JATIT. All rights reserved.

distributed system. If some processes are interested to participate to an opened session, and no process is interested in a different session, then the process can attend the session concurrently. Future works involves a more detailed study of the performance in the face of failure, as well as comparisons with additional algorithms.

8. REFRENCES

[1] R. Atreya, N. Mittal, "A Distributed Group Mutual Exclusion Algorithm using Surrogate-Quorums", *IEEE International Conference on Distributed Computing Systems* (ICDCS), 2005, pp. 251-260.

[2] R. Baldoni, A. Virgilito, "A token-based mutual exclusion algorithm Distributed mutual exclusion for mobile ad hoc networks", *Tech. Report 28-01, Dipartimento di Iinformatica*, Univ. di Roma, 2000,.

[3] S. Banerjee, P.K. Chrysanthis, "A new token passing distributed mutual exclusion algorithm", *16th International Conference on Distributed Computing Systems*, 1996, pp. 717-724.

[4] J. Beauquier. S. Cantarell, A.K. Datta, "Group mutual exclusion in tree networks", *Journal of Information Science and engineering*, Vol. 19, 200, pp. 415-432.

[5] O. Carvalho, C. Roucairol, "On mutual exclusion in computer networks", *CACM*, Vol. 26, No. 2, 1984, pp. 146-147.

[6] K.M. Chandy, J. Misra, "The drinking philosophers problem", *ACM TOPLAS*, Vol. 6(4), 1984, pp. 632-646.

[7] Y.I. Chang, M. Singhal, and Liu T, "A dynamic token-based distributed mutual exclusion algorithm", *10th International Conference on Computers and Communications,* Scottsdale, Arizona USA, 1991.

[8] Y.J. Joung, "Asynchronous group mutual exclusion", *Distributed Computing*, Vol. 13, No. 4, 2000, pp. 189-206.

[9] L. Lamport, "Time, clocks , and the ordering of events in distributed system", *Communications of the ACM*, Vol. 21, No. 7, July 1978, pp. 558-565.

[10] M. Maekawa, "A O($\sqrt{n}$) algorithm for mutual exclusion in decentralized systems", *ACM Trans. Computer systems*, May 1985, pp. 145-159.

[11] Y.J. Joung, "The congenial talking philosophers problem in computer networks", *Distributed Computing*, 2002, pp.155-175.

[12] G. Ricart, "An optimal algorithm for mutual exclusion in computer networks", *CACM*, Vol. 24, No. 1, 1981, pp. 9-17.

[13] M. Naimi, "Une structure arborescente pour une classe d'algorithmes distribues d'exclusion mutuelle", *PhD Thesis*, University of Franche-Comté, 1987.