International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 5 (2017) pp. 738-743

An Efficient Fault-Tolerant Algorithm for Solving Mutual Exclusion Problem over Peer-to-Peer Networks

Khalid A Altarawneh* and Dr. Murat Akkaya **

**Department of Management Information Systems, Mutah University; Jordan.*
***Associate Professor, Department of Management Information Systems, Girne American University; North Cyprus, Turkey.*

Abstract
Distributed system is a collection of distinct processes that do not share memory or clock. These processes interconnected by a communication network in which each process has its own local memory and other peripherals. The communication between any two processes of the system takes place by message passing over the communication network. These processes in the distributed system run concurrently. The purpose of the distributed system is to provide an efficient and convenient environment for sharing of resources.In this paper, we present an efficient fault-tolerance algorithm for Mutual Exclusion (ME) in distributed systems. This algorithm is considered as an enhancement to Reddy algorithm. The algorithm is based in token-based technique and requires a lot number of messages to enter the critical section (CS) and to detect the system failures. In Reddy algorithm the process must request to enter the CS from all process all the times, we used a technique wherein it can enter the CS by sending less number of messages. Also our algorithm reduces the number of messages for the detection of the token lost. We compare our algorithm with Reddy and we find that our algorithm is more efficient.

Keywords: Mutual Exclusion, critical section

INTRODUCTION

Mutual Exclusion (ME) problem is a classical problem in distributed systems [Swaroop]. It occurs in an environment where several system processes are working at the same time and share the same recourse [6], the two processes modify a code for the same shared resource, and this code is called critical section (CS). Their must not be two processes execute their critical sections concurrently [2]. It is forbidden to a process to enter the CS while another process is in the corresponding section of code [7]. Numerous algorithms have been proposed to solve mutual exclusion in distributed systems. They can mainly be divided into two groups: permission-based algorithms [Manivannan] [Ricart] and token-based algorithms [1,2,3]. In permission-based algorithms each process can enter its critical section just after getting permission from all other processes [9]. While in token-based algorithms, the mutual exclusion is achieved by using a single token that is circulated between the processes in the system. The process that has the token has the permission to enter the CS.

The presented algorithms also can be seen from another point of view, the fault tolerant; we said that the algorithm is fault-tolerant if it treats the failure and still function without any problems. There are many algorithms that take into account several types of failures and treat them [8]. By contrast, we can not consider that the system is reliable and it is error free so to build an algorithm over this suggestion is not practical and they can not be classified with the others that take in their accounts the error and any failures in the system. The fault tolerant algorithms could be different in their efficiency and complexity. For example, in token-based algorithm the efficiency depends on the number of messages needed for entering the CS, for detecting loss of token, and for generate new token.

We present in this paper an efficient fault-tolerant token-based algorithm that reduces the number of messaged needed to enter the CS, the number of messages to detect the loss of token and to generate a new token in case of failures. We compare our algorithm with the algorithm presented in 2006[2].

Related Work

Many fault-tolerant token-based algorithms have been proposed to handle site failures and message loss[13, 11,12, 14,4,10,2]. Here is a brief review of some of these algorithms. In Chang algorithm [13], using election algorithm to generate a new token when token is lost. Singhal's [11] method for handling process and link failures maintains and uses state information about other processes for acquiring token. In Nishio algorithm [14], when a process detects token loss, it requires a positive acknowledgment from every other process to create a new token. In this algorithm, each site I has an integer variable called site-agei and also the token has an integer variable associated with it, called token-age. initially, token-age 0 and site-agei =0 for all I. if a site I making request for CS does not receive a valid token within a timeout period, it tries to generate a new token by setting site-agei:= propose-agei and sending a TOKENMISSING(propose-agei) message to every other site, where propose-agei is the smallest positive integer of the form K*N+I greater than the current value of site-agei, K being an integer. This choice of propose-age prevents two processes from sending TOKENMISSING message with the same propose-age. When the site J receives the TOKENMISSING (propose-agei) message from site I, it sends NACK reply to site I if it has the token or if site-agej is greater than propose-agei. But this method requires large number of message to generate a new token. Chaudhuri and Edward in [4] proposed algorithm

to reduce the number of messages exchange between processes to execute critical section. They partition the N processes of the network into √n sets of √n processes each. Each set is called a local group (LG). Processes in a local group can communicate directly with each other for the purposes of entering the critical section. That is, all processes in a local group are fully connected and select one process from each group To configure global group (GG) advantage the global group connect local groups with each other. This algorithm is based on queue migration and achieves a message complexity of O(√n) per mutual exclusion invocation. But this method depending on one process to organize CS between processes in the system. When any process want to execute critical section it must send CR message either to local request collector (LRC) when this process is a member of local group or to global request collector (GRC) when this process is a member of global group. Manivannan [10] token-based fault-tolerant mutual exclusion algorithm, can recover from loss of token, provided the process which failed after getting token is recover within a finite period of time. In Reddy et al's algorithm [2], proposed simple modification to Manivannan algorithm to give better performance in terms Message complexity and Synchronization delay. But this method required large number of messages when the token is lost or when more than one process want to execute the critical section. In case more than request each process did not receive the token in the timeout period send "is-token-lost" message to all other processes in the system although the token is not lost.

The rest of the paper is organized as follows. Section 2 presents the system model. Our proposed algorithm is presented in section 3 and Section 4 gives the correctness proof of the algorithm. Section 5 gives simulation results and performance comparison with Reddy et al's algorithm and finally section 6 concludes the paper.

SYSTEM MODEL

The system has *N* nodes numbered [*0…. N-1*] and there is only one critical section in the system. The nodes communicate only through messages. The system is fully logically connected that is the underlying network supports every node in the system to send message to every other nodes.

PROPOSED FAULT-TOLERANT ALGORITHM

We modified the Reddy et al's algorithm. Reddy algorithm is based in a fixed predefined timeout period to request and detect the system failures. Our alg. Use a variable length of timeout. The timeout depends on the number of process already have been requesting and waiting to enter the CS, the requesting process receives an acknowledgement from the process that is currently having the token. These processes are kept in a local queue in each process. The queue is generated when the process request to enter the CS and it enqueue itself. The acknowledgment message has a new structure which contains the processes already requested entering the CS. The acknowledgment prevents from multiple sending of request and reduces the number of messages to detect message loss or process failure. The time that the process must wait to check the system failures depends on the number of the process in the received acknowledgment message. While the process is in a CS, any request coming is entered to the queue

In our algorithm also, when the process finishes executing the CS, it dequeues it self and send the token and the queue to the next process in the local queue depending in the FIFO strategy. The token is passed from process to other in the order specified by the Queue. The site which executes the critical section is the current site at the head of the Queue. Each site in the system maintains a local queue.

Basic Idea

Assume *Pi* want to execute critical section, it sends CS request message to all other processes. When *Pj* receives this message, it checks 2 conditions (1) it has idle token (2) process i's request is of highest priority. If both conditions are satisfied, token with the queue is sent to *Pi*. Otherwise; it queues up the request in the queue. If Pj receives more than one request during work in the critical section, it enqueues process number in the queue and send the acknowledgement to the requesting process . After it finishes its work in the critical section the token and the queue are sent together to the next process.

When the pi receives the token message and queue, it performs the following points before entering the CS:

- Dequeues itself
- Put its process-id to the global queue.
- ??? Send ACK message the queue to all processes there exist in the T-queue.

Data Structure

A site can execute CS only when it possesses the token, a privilege message, which is a data structure of type token-type as defined below.

- Token: a Dynamically-Altered-Record which consists of three fields:
  1. Cs_excuted: an integer array [0…N-1] (N is the number of distributed sites), which its index represents the IDs of the distributed sites, and each cell contains the number of times that site has executed a CS.
  2. Total: an integer which contains the number of all CS executions by all sites.
  3. T_queue: an integer array [0…N-1] (N is the number of distributed sites), represent the order of the sites that want to execute the critical section.
- RNi : a Site_Local array [0…,N-1] (N is the number of distributed sites), which its index represent the ids of the distributed sites, and each cell contains the HIGHEST-REQUEST-SEQUENCE-NUMBER of the request for the CS received by the local site I from every corresponding site (which is identified by the cell index) .

- Local queue: a dynamically generated FIFO-Received-Requests queue, in case that more then one request is received before exiting the CS, this queue will contain the received the CS requests ordered by requests timestamps.
- Timestamp: a Dynamically-Generated-Value which is equal to time of its generation, every site generates timestamps as needed and uses them in manipulating the received CS requests.
- Global queue: a Constant_Integer_Array [0…3], represent the latest three sites posses copy of the token.
- Token_copyi: a Site-Local-Variable of token–type which contains a copy of the token when site I exited the CS last time.
- Seq_ni: a Site-Local-Variable of integer type which contains the sequence number assigned to the last request for CS sent by site I.
- Token_herei: a Site-Local-Variable which indicates whether token is in site I or not. Initially, token-herei = false at all sites except at site 0.
- Token_generated_fori: a Site-Local-Variable of integer type which contains the ID of site to whom site I has sent the token recently.
- Timeout: a Global_Value which contains the time quantum that a site waits after sending a CS request before it sends a probe message (is_token_lost?).
- Site_id : a Global-Identification-Number for every site.

The Algorithm

The following algorithm at process i is given in a Pascal like notation.

- Algorithm at site i:
- RNi: array [0..N-1] of integer; /* initialized with 0s*/
- Token_copyi: token type;
- Seq_noi: integer; /* initialized with 0 */
- Token_herei: Boolean; /* initialized as mentioned in 3.3.4*/
- Requested-for-CS: Boolean; /* initialized with false */
- Token-generated-fori : integer; /* initialized with -1

Procedure Request_CS ( )

begin
seq_noi:= seq_noi+1; RNi[i]:= seq_noi;
If (token_herei) then
{
put its id in the global queue
after put its id in the global queue check the T_Queue.
If the (T_Queue>0)
{
broadcast the global queue to all the processes in the T_Queue.
Remove its process_id from the front T_Queue and enter to the critical section.
execute CS; token.total:= token.total+1;
token.cs_exuted[i]:= RNi[i]; token_copyi:= token;
Else
Remove its process_id from the front T_Queue and enter to the critical section.
execute CS; token.total:= token.total+1;
token.cs_exuted[i]:= RNi[i]; token_copyi:= token;
}
else
send Bcast (requesti, RNi [i]) to every process;
}
end;

Procedure Handle_CS_Request ( )

receive Bcast (request j , RN j [j]) form process j generate a timestamp of Bcast receiving time ;
if(RN i [j] < RN j [j]) then
RN i [j] := RN j [j];
if (token_here i) and (token.cs _ exuted [j] < RN I [j]) and (process have IDLE token) then
send (token with T_Queue and global queue) to (process j) ; token _ here i := false ;
else
if (token _ herei) and (token.cs_exuted[j] < RN i [j]) and (process have BUSY token) then
push j into local queue ;
if (more than one request were received with the same timestamp) then
push these requests into local queue in ascending order according to process_id;
when exiting the CS , copy the processes id from the local queue and push these id at the end of the T_Queue, send the token with the new T_Queue to the process in the front T_Oueue;
end;

Procedure On_receive_token ( )

Begin

if (token_copyi.total < token.total) then put its process_id in the global queue; check the T_Queue;
if(T_Queue>0) then
{
broadcast the global queue to all other process in the T_Queue;

Remove its process_id from the front T_queue and enter to the critical section;
token_here:= true; execute CS;
token.total:= token.total+1; token.cs_exuted[i]:= RNi[i];
token_copyi:= token;
}
else
{
Remove its process_id from the front T_queue and enter to its critical section;
token_here:= true; execute CS;
token.total:= token.total+1; token.cs_exuted[i]:= RNi[i];
token_copyi:= token;
}
After finish its work in the critical section call procedure Handle_CS_Request ( )and send the token and T_Queue to the process at the front T_Queue ;
end;

Procedure Handle_nodefaliure_messagelost ( )

Begin
After receive the global queue message;
if (token not received in timeout period) or (not receive another global queue message in time out period) then
send Bcast(is-token_lost?, i, RNi [i]) to all process that exist in the global queue;
if (receive Bcast(is_token_lost?, j, RNj[j]) from process j)
if (RNi[j] < RNj[j]) then
RNi [j]:= RNj[j];
if (token_herei) and (token.cs_exuted[j] < RNi [j])
then
send ACK message to indicted that the token is not lost but Busy ;
else
send reply(i, token_copyi);
if (process i receive ACK message) then
waiting another timeout period ;
else
collect reply messages from " m where m <= N-1;
Find process k such that
token_copyk.total is max m (token_copym.total)
inform process k to generate_token;
if (receive generate-token from process j) then
if (token not generated within timeout period) then
token: = token_copyj; token-generated-fori: = j;
send (token) to process j;
else
send token-generated (l, RNi [l]) to j where
l is the node to whom i last sent the token
end;

PERFORMANCE ANALYSIS *(*results and testing*)*
We compare our algorithm with the ready algorithm by implemented the two in C# using multi-threading technique.

First Reddy et al's algorithm
When we applied Reddy et al's algorithm in normal situation and in another case if there is one request to enter the critical section. We found that the number of messages transmitted between processes in these algorithms as follows.
(N-1) request message + one token message = (N Message).
Whereas: (N) number of processes in the system.

But when is more than one request to enter the CS we found the number of message in this algorithm as follow.
M (N-1) Request Message + Token Message + (M-1) (N-1) Is-Token-Lost Message + R Reply Message.

Whereas:
N: Number of processes in the system.
M: Number of processes wants to execute the critical section.
R: Reply message.

As shown in the following table.

Number of processes wants to execute the CS	Number of messages exchange between processes
1	9
2	29
3	47
4	65
5	83
6	101
7	119
8	138

Our Suggestion
When we applied our suggestion in normal situation and in another case if there is one request to enter the critical section. We found that the number of messages transmitted between processes in this algorithm as follows.
(N-1) request message + one token message = (N Message).
Whereas: (N) number of processes in the system.

But when is more than one request to enter the CS we found the number of message in our suggestion as follow.
M (N-1) Request Message + Token Message + M (ACK Message with Global Queue Message).

Whereas:
N: Number of processes in the system.
M: Number of processes wants to execute the critical section.

As shown in the following table:

Number of processes wants to execute the critical section.	Number of messages exchange between processes in our suggestion.
1	9
2	18
3	27
4	36
5	45
6	54
7	63
8	72

We cannot account the number of transmission messages between the processes if there any failure case in the system because the creator for this algorithm didn't take in the consideration the treatments operation in failure case.

When we tested both algorithms in failure situation and in case if there is one or more than process posses copy of the token after failure situation is done. We found that the number of messages transmitted between processes in both two algorithms as follows.
First: Reddy et al's Algorithm

(M (N-1)) Request Message + (M (N-1)) Is-Token-Lost Message + (F M) Reply Message + (M) Generating Message + (F) Copy of the Token Message.

Whereas:
N: Number of processes in the system.
M: Number of processes wants to execute the critical section.
F : Number of processes posses' copy of the token.

Second: Our Suggestion
(N-1) Request Message + X "is-token-lost" message + X Reply Message + one generating token message + token message.
Whereas:
N: Number of processes in the system.
X: Number of process in the global queue at most 4 processes.

We will take one example: when one process wants to execute the critical section and more than one process posses copy of the token. As shown in the following table and figure 1.

Number of processes posses copy of the token	Number of messages exchange between processes in Reddy algorithm.	Number of messages exchange between processes in Our suggestion.
1	19	13
2	20	15
3	21	17
4	22	17
5	23	17
6	24	17
7	25	17
8	26	17

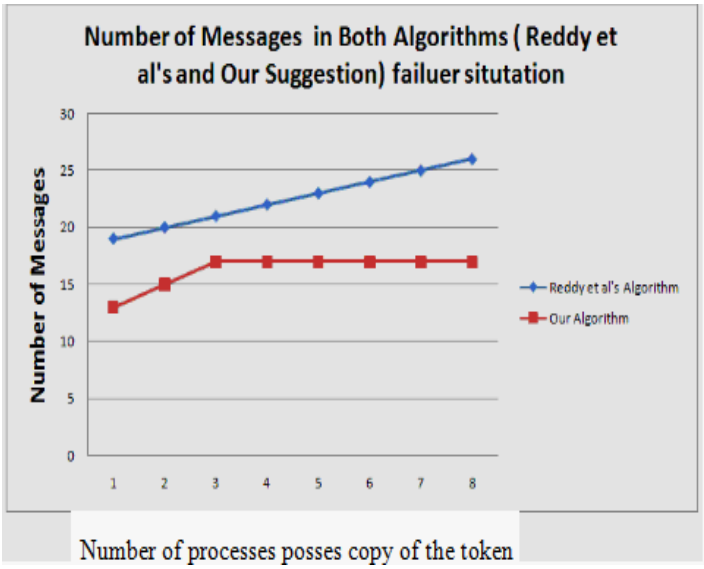

Figure 1: Number of Messages in Both Algorithm in failure situation.

We got through what the previous figures illustrated that the number of messages in our suggestion in the normal case and failure case is less than number of messages in Reddy's algorithm.

CONCLUSION

In this paper we compared between two mutual exclusion algorithms from different aspects and norms such as number of messages per CS execution, Scheduling of processes to

enter critical section and dealing with some failure cases. Also we presented an efficient fault-tolerant token-based mutual exclusion algorithm for distributed system which is an extension of Reddy et al's algorithm and made main adjustments that we have got new algorithm to improve performance by reducing number of messages that necessary to detect token loss and to generate token. Similarly, by reducing number of rounds of communications. As well as we obtained reducing synchronization delay as compared to, which in turn leads to lower response time.

We present an efficient algorithm to solve the problem of mutual exclusion in fully connected network. Our algorithm enhances the algorithm of Reddy et al's performance. Our algorithm can operate in reliable system and it can tolerate the node failures.

REFERENCES

[1] Vaibhav Saini, "Fault-Tolerant Token Based Algorithm for Mutual Exclusion in Distributed Environment" Computer Science & Engineering Department Thapar Institute Of Engineering & Technology, May 2006.

[2] P. Sukendar Reddy, Nityananda Sarma, Rajib Kumar Das," Enhancing Fault-Tolerance in a Distributed Mutual Exclusion Algorithm" 9th International Conference on Information Technology (ICIT'06) 0-7695-2635-7/06 $20.00 © 2006, IEEE.

[3] Sandeep Lodha and Ajay Kshemkalyani, Senior Member, IEEE, "A Fair Distributed Mutual Exclusion Algorithm", IEEE Transactions on Parallel and Distributed Systems, VOL. 11, NO. 6, JUNE 2000.

[4] Pranay Chaudhuri and Thomas Edward, "An O (√n) Distributed Mutual Exclusion Algorithm Using Queue Migration1" Journal of Universal Computer Science, vol. 12, no. 2 (2006), 140-159 submitted: 18/1/05, accepted: 20/10/05, appeared: 28/2/06 © J.UCS.

[5] Swaroop, A., Singh, and A.K.:" A Token-Based Fair Algorithm for Group Mutual Exclusion in. Distributed systems". J. of Computer Science 3(10), 829–835 (2007)

[6] T. H. Bredt, "The Mutual Exclusion Problem", August 1970, Technical Report No. 9.

[7] Lisa Higham, Member, IEEE, and Jalal Kawash," Tight Bounds for Critical Sections in Processor Consistent Platforms" IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 10, October 2006.

[8] P.C. Saxenaa, J. Raib,*," A survey of permission-based distributed mutual exclusion algorithms" Computer Standards & Interfaces 25 (2003) 159–181.

[9] Paul Krzyzanowski," Process Synchronization and Election Algorithms" Rutgers University – CS 417: Distributed Systems ©1998-2009 Paul Krzyzanowski.

[10] D. Manivannan, M.Singhal, A Efficient Fault-tolerant Mutual Exclusion algorithm for distributed system, In Proc. of the International conference on parallel and distributed computing Systems, October 1994, pp 525-530.

[11] M.Singhal, "A heuristically-aided algorithm for mutual exclusion in distributed systems, IEEE Transactions on Computers", vol. 38, No.5, May 1989, pp. 651-662.

[12] Abhijeet Uday Gole, "Master Token Resource Management Algorithm for Distributed System Priority Queue Strategy", International Journal of Recent Trends in Engineering, Vol 2, No. 2, November 2009.

[13] Chang, M. Singhal, and M. Liu, "A fault tolerant algorithm for distributed mutual exclusion" in Proc. of 9th IEEE Syrup. On Reliable Dist. Systems, 1990 pp. 146-154.

[14] S.Nishio, K.F. Li and E.G. Manning, "A resilient mutual exclusion algorithm for computer networks", IEEE Transactions on Parallel and Distributed Systems, vol. 1, No.3, July 1990, pp.344-355.