2009 International Conference on Machine Learning and Computing
IPCSIT vol.3 (2011) © (2011) IACSIT Press, Singapore

Several-Tokens Distributed Mutual Exclusion Algorithm in a Logical Ring Network

Ousmane THIARE [+]

Department of Computer Science, UGB-UFR/SAT BP. 234 Saint-Louis SENEGAL

Abstract. A mutual distributed algorithm on a token ring is proposed in this paper. The proposed algorithm is based on the token ring approach and allows simultaneous existence of several tokens in the logical ring of the network. Each process generates a unique token and sends it as request to enter the critical section that travels along the ring. The process can only enter the critical section if it gets back its own token. The proposed algorithm also handles the coincident existence of multiple critical sections (if any) in the system.

Keywords: mutual exclusion, logical token ring, critical section

1. Introduction

In distributed systems, cooperating processes share both local and remote resources. Chance is very high that multiple processes make simultaneous requests to the same resource. If the resource requires mutually exclusive access (critical section-CS), then some regulation is needed to access it for ensuring synchronized access of the resource so that only one process could use the resource at a given time. This is the distributed mutual exclusion problem [1]. The problem of coordinating the execution of critical sections by each process is solved by providing mutually exclusive access to the CS. Mutual exclusion ensures that concurrent processes make a serialized access to shared resources.

In a distributed system, neither shared variables (semaphores) nor a local kernel can be used in order to implement mutual exclusion. Thus, it has to be based exclusively on message passing, in the context of unpredictable message delays and no complete knowledge of the state of the system. Basic requirements for a mutual exclusion mechanism are:

- **Safety:** at most one process may execute in the critical section (CS) at a time;
- **Liveness:** a process requesting entry to the CS is eventually granted it (so long as any process executing the CS eventually leaves it). Liveness implies freedom of *deadlock* and *starvation*.

Our proposed algorithm is based on the token ring and allows simultaneous existence of multiples tokens in the logical ring of the network. Each of the competing nodes generates a token for the permission to enter the CS. The token traverses the logical ring structure. The process can only enter the critical section if it gets back its own token. Each node maintains separate queues for storing different CS entry requests and thereby allows the existence and evolutions of multiple CSs at the same time. The proposed algorithm evenly distributes the burden of controlling the access to CS among all nodes in the network that make it more distributed and adaptable to growing network size.

In our algorithm, we try to increase the overall system throughput by decreasing both the average waiting time and the number of message passing per critical section entry. The second goal is to increase the CS access fairness and the system robustness.

[+] Corresponding author. Tel.: (+ 221) 77 556-84-77; fax: (+221) 33 961-53-38.
E-mail address: othiare@dept-info.u-cergy.fr

The rest of our paper is organized as follows: Section 2 describes the existing related works and the system model is introduced in section 3. The proposed algorithm and its correctness proof are presented in section 4. The conclusion is presented at the last section.

2. Related Work

All the algorithms presented in this section claim to satisfy the mutual exclusion requirement, no deadlock and starvation free. Their major characteristics and assumptions are discussed. The average waiting time per critical section entry, fault-tolerance and the number of message exchanged for an entry to the critical section to take affect are used as a complexity measure to compare them.

Two approaches have been used to implement a mutual exclusion mechanism in distributed computing systems: *centralized approach* and *distributed approach* [3]. This paper only considers the distributed approach. Distributed mutual exclusion algorithms are designed based on two basic principles: the existence of token in the system or the collection of permission from nodes in the system [4][5][6]. Once again, we will concentrate on token-based algorithms only.

In Suzuki-Kasami's broadcast algorithm [8], when a node wants to enter the critical section, it broadcasts a message to all other nodes. Whoever holds the token sends the token directly to the node that wants to enter CS. The algorithm requires *N* messages for handling each request. The simplest of token-based algorithms is the Agrawal-Elabbaei's token ring approach [7]. In this algorithm, the nodes in the system form a logical ring. A token always passes around the ring clockwise or anticlockwise. A node can enter the critical section if it holds the token. On an average *N/2* messages are required to handle one request in an *N* nodes system.

Nielsen and Mizuno extended by passing the token directly to the requesting node instead of through intermediate nodes [10]. Naimi-Trehel's algorithm [11] maintains a dynamic logical tree, such that the root of the tree is always the last node that will get the token among the current requesting ones. Chang, Singhal and Liu [12] improved this algorithm, aimed to reduce the number of messages to find the last requesting host in the logical tree. Mueller [13] also proposed an extension to Naimi-Trehel's algorithm, introducing the concept of priority and the algorithm first satisfies the request with higher priority.

In [14], Kawsar et al. introduced an enhanced token ring algorithm, they considered node as a competing unit and it requires *N* messages for handling a single request. But how to handle duplicate tokens within the ring is not also clarified. And the algorithm is described considering the conflict for a single resource only.

Jiang in [15], proposed a prioritized *h-out of-k mutual exclusion algorithm*. His algorithm provides mutual exclusion, deadlock and starvation freedom, concurrency and gives priority to real time applications of critical section entry. But, he assumed that the network would not be partitioned which is not realistic, *i.e.*, fault tolerance issues due to the lost token and partitioning problems have totally been avoided.

3. System Model

Network model

The proposed algorithm is based on the token ring algorithm. The following assumptions and conditions for the distributed environment are considered while designing the algorithm:

- all processes in the system are assigned unique identification number
- there may be more than one requesting processes from a node, mutual exclusion is implemented at the node level.
- nodes may compete for multiple resource type at the same time.
- process failure may be occurred.
- nodes or communication links may fail, which can result in partitioning of the network.

Another important characteristic of our network model is that messages are arrived at the destination node in the order in which they were sent from the source nodes. The network may be of any topology. In software, a logical is constructed in which each node is assigned a position in the ring as shown in Fig. 1. The logical ring topology is unrelated to the physical interconnections between the computers.

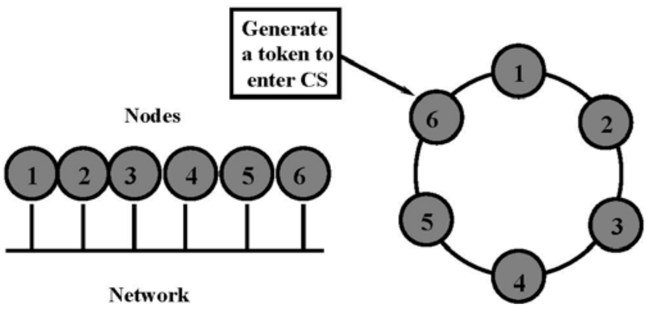

Fig. 1. Logical ring of unordered node of the network

Computational model

The token structure is as follow:

Token {

TokenID; // includes address of a node

ResourceID; // resource for which the request is made

Resend; // 0 for initial token, 1 for retransmitted tokens

Each token is uniquely identified by its request *TokenID* and *ResourceID*, where *TokenID* also works as a priority identifier and is defined as *TokenID= (SeqNum, PID).* The sequence number, *SeqNum,* is a locally assigned unique sequence number to the request token and *PID* is the process identifier that includes the node *ID* also. In the sequel, T_i^r represents the token generated from node N_i for resource *r*. *SeqNum* is determined as follow. Each node maintains the highest sequence number seen so far in a local variable *HSeqNumSeen.* When a node makes a request for the CS, it uses a sequence number which is one more than the value of *HSeqNumSeen.* When a token is received, *HSeqNumSeen* is updated us follows: *HSeqNumSeen=max (HSeqNumSeen, SN),* where, *SN* is the sequence number in the received token. Priorities of two requests, $TokenID_1$ and $TokenID_2$, where $TokenID_1= (SN_1, PID_1)$ and $TokenID_2= (SN_2, PID_2)$, are compared as follows.

Pri $(TokenID_1)$ is greater than Pri $(TokenID_2)$ *iff* $SN_1<SN_2$ *or* $(SN_1=SN_2$ *and* $PID_1<PID_2)$. All requests carried by tokens are thus totally ordered by priority. When these requests are satisfied in the order of decreasing priority, fairness is seen to be achieved.

Each node maintains priority queues for each resource type. A node N_i wants to enter CS(*r*), enqueues all tokens for resource *r*, T^r having lower priorities in queue, Q_i^r, and release them after completion of CS executions. The request queue structure is as follows:

request queue {// a priority queue

Token;

next-Token;}

};

4. Proposed algorithm

The algorithm works as follows: Every node maintains separate queues for each requested resource type in its local memory. The token generated by a node moves forward either clockwise and anticlockwise.

Rule-1. A node N_i wants to enter the critical section, CS(r), and generates o token, T_i^r, makes a copy of the token in its node and masses it to the next node.

Rule-2. Any node N_j receives a token for CS(r), and reacts to it in the following ways:

- if N_j has no intention to enter the CS(r), it replaces the sequence number by HSeqNumSeen and simply passes the token to the next node.
- if N_j is now in the CS(r), it puts the incoming token in Q_i^r. When N_j exits CS, it releases the tokens to the next node sequentially (if any) from Q_i^r.
- if N_j has already generated a token but not yet received that back, it compares the incoming token's priority with it's generated token's priority. If **Pri** (T_i^r) < **Pri** (T^r), it passes the token to the next node; otherwise it puts the token in Q_i^r.

Rule-3. if the node N_i receives T_i^r, that is all other nodes allowed N_i to enter CS(r), and then it enters the CS. On exiting from the CS(r) it sends the queued tokens to the next node sequentially (if any) and deletes the associated copy and original token.

Rule-4. If the node N_i does not receive its own generated token, T_i^r, within a certain timeout period (because, either token is lost or held by some other died node), N_i retransmits the token with the initial priority identifier and *Resend* field value of 1. No node will store duplicate tokens in their request queues. Upon reception of a token, if its *Resend* field is found to be 1, a simply query in the request queues is executed to find that token. If it is not found, the token is added in the request queue, otherwise it is simply discarded. Doing so handles the lost token detection and regeneration problem and ensures the *progress* property of critical section problem.

Rules-5. We assume in our algorithm that tokens are never lost and if a node dies, there is no way to detect it. To avoid this unexpected situation, acknowledgement of receiving o token from the next node may be used. Thus node failure can be easily be detected. At the point the dead node can be removed from the logical ring and the token holder can throw the token over the head of the dead one to the next node down the line. Of course, doing so requires that the network is fully connected. We assume that an up node never lies, faithfully executes the mutual exclusion algorithm and honestly interacts with other processes. So, the node failure case can easily be handled in the proposed algorithm rather than some exhaustive crash recovery procedures employed in [7][9][10][15] .

Correctness proof

Message complexity. As the token must visit all the nodes before returning back to its originator, the number of messages per CS access is deterministic and always *N-1*, where *N* is the number of nodes in the system. A mutual exclusion algorithm satisfies the safety specification of the mutual exclusion problem if it provides mutually exclusive access to the critical section. A (safe) mutual exclusion algorithm is said to provide fair mutual exclusion if the following property holds.

Property 4.1: An algorithm provides fair mutual exclusion *iff* Pri (T_i^r) > Pri (T_j^r) $\leftrightarrow$ N_j executes CS(r) after N_i finishes its execution.

Definition 4.1: Two tokens T_i^r and T_j^r are concurrent ($T_i^r \parallel T_j^r$) *iff* N_i's request is received by N_j after N_j has made its request and N_j's request is received by N_i after N_i has made its request.

Each token T_i^r sent by N_i has a concurrency set, denoted by *Cset*$_i$, which is the set of those requests T_j^r that are concurrent with T_i^r. *Cset*$_i$ also includes T_j^r.

Definition 4.2: Given T_i^r, $Cset_i$={ T_j^r | T_i^r is concurrent with T_j^r}U{ T_i^r}. The relation "is concurrent with" is defined to be symmetric, *i.e.*, $T_i^r \in Cset_j$ iff $T_j^r \in Cset_i$.

Theorem 4.1: (Safety and Fairness) The proposed algorithm provides safe and fair mutual exclusion as defined in property 4.1.

Proof. We prove it using *proof by contradiction*. Safety property is violated whenever more than one node is allowed to execute the same CS simultaneously. If it is true, one of the following conditions must be hold:

- N_i enters CS before getting back its generate token.
- N_i forwards other requests while it is in CS execution.
- Two requests tokens have the same sequence number.

First condition can never hold, because it contradicts with the restriction imposed by relations (1) and relation (2) below:

If own token is received back (1)

Enter the CS and execute it (2)

Second condition directly opposes the relation (3) below:

Enqueue the token into the priority queue (3)

which state that any T_i^r received by N_i, while it is in CS(*r*) execution, it enqueued at local queue and forwarded after completing CS execution. The probability that two tokens appear at a node having the same sequence number is zero, because of relation (4) and (5) below when new tokens are assigned highest sequence number so far seen plus one.

HSeqNumSeen= HSeqNumSeen+1 (4)

HSeqNumSeen=max (HSeqNumSeen, SN) (5)

Similarly, if $T_i^r \parallel T_j^r$ and Pri (T_i^r) > Pri (T_j^r), fairness property is infringed whenever node N_j gets access to CS first, depriving the higher priority request, T_i^r. This situation will stand if at least one of the following conditions holds:

- N_i enters CS before getting back its generate token.
- N_i forwards other requests while it is in CS execution.
- Only a subset of nodes of the system takes part in decision.

First condition can never hold, already proved above. The second condition contradicts with relations (6) and (7), where N_i forwards only higher priority tokens.

If (no token is generated by this node or (Pri (received Token) >= Pri (N_i's Token) and not executing CS now)) (6)

Forward the Token to the next node (7)

The third condition contradicts with the system computation model, which models a logical ring structure of nodes and a token must pass through all these nodes before getting back to its originator. Therefore, fairness property holds in our algorithm.

Theorem 4.2: (Liveness). The proposed algorithm achieves liveness.

Proof. Liveness property of a mutual exclusion algorithm includes two sub-properties: *starvation* free and *deadlock* free, *i.e*, each node N_i, requesting to enter its critical section, will obtain it in a finite time. Again, we prove this theorem using *proof by contradiction.* Starvation occurs when few nodes repeatedly execute their CS's while other nodes wait indefinitely for their turns to do so. This situation may only occur if new requests are coming with higher priority, which is impossible in the proposed algorithm.

Let T_i^r be the request that has the highest priority among all requests ever made and T_j^r be the request that has the lowest priority among all request ever made till now. Theorem 4.1 tells us that the request T_i^r is serviced first and then all other requests $T_k^r \in Cset_i$ are serviced in order. Relations (4) and (5) force a newly arrived token to receive lowest priority, therefore progress property is preserved. Moreover, the proposed algorithm is deadlock free as because there is no way that a token traverses the ring indefinitely or a node executes its CS for infinity period of time. Thus, the request T_j^r is serviced within finite time and the algorithm guarantees liveness.

5. Conclusion

In this paper, we have proposed an algorithm for solving mutual exclusion in a logical ring network. This algorithm is based on the token ring and allows simultaneous existence of several tokens in the logical ring of the network. This algorithm also achieves safety and liveness properties. Future works involves a more detailed study of the case of lost tokens.

6. References

[1] E. W. Dijkstra. Solution of a Problem in Concurrent Programming Control Recognition. *Communication ACM*, vol. 8, no. 9, Sept 95.

[2] S. Liu, S.Q. Lian, M. Chen and Z. Zhang. A Practical Distributed Mutual Exclusion Protocol in P2P Systems. *Microsoft Research, Technica Report*, MSR-TR-2004-13.

[3] M. Benchaiba, A. Bouabdallah, N. Baddache and M. Ben. Nacer. Distributed Mutual Exclusion Algorithms in mobile ad hoc networks: an overview. *ACM Operating Systems Review*, vol. 38, no. 3, July 2004, pp. 74-89.

[4] L. Lamport. Time, clock, and the ordering of events in a distributed system. *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[5] G. Ricart and A.K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM,* vol. 24, no. 1, pp. 9-17, 1981.

[6] M. Maekawa. A sqrt(n) algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems.* Vol. 3, no. 2, pp. 145-159, 1985.

[7] D. Agrawala, A. Elabbaei. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems.* Vol. 9, no. 1, pp. 1-20, 1991.

[8] I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. *ACM Transactions on Computer Systems. Vol. 3*, no. 4, pp. 344-349, 1985.

[9] M. Bertier, L. Arantes and P. Sens. Hierarchical token based mutual exclusion algorithms. *IEEE International Symposium on Cluster Computing and the Grid*, pp. 539-546, 2004.

[10] M. Naimi, M. Trehel and A. Arnold. A log(N) distributed mutual exclusion algorithm based on path reversal. *Jour. Of Parallel and Distributed Computing,* vol. 34, no. 1, pp. 1-13, 1996.

[11] I. Chang, M. Singhal and M.T. Liu. An improved log(N) mutual exclusion algorithm for distributed systems. *Proc. of the International Conf. on Parallal and Distributed Processing*, pp. 295-302, 1990.

[12] F. Mueller. Prioritized token-based mutual exclusion for distributed systems. Proc. of the 12th Intern. *Proc. of symposium & 9th symposium on Parallel and Distributed Processing*, pp. 791-795, 1998.

[13] F. Kawsar, H.S. Shariful, M.S. Saikat, M.A. Razzaque and M.A. Mottalib. An efficient token based algorithm for mutual exclusion in distributed system. *Jour. of Engineering and Technology,* vol. 2, no. 2, pp. 39-44, 2003.

[14] J.R. Jiang. A prioritized h-out of-k mutual exclusion algorithm with maximum degree of concurrency for mobile ad hoc networks and distributed systems. Proc. of the 4th Intern. Conference on Parallel and Distributed *Computing, Applications and technologies (PDCAT),* pp. 329-334, 2003.