Multi-Token Distributed Mutual Exclusion Algorithm

Md. Abdur Razzaque
Dept. of Computer Engineering,
Kyung Hee University, 449-701
m_a_razzaque@yahoo.com

Choong Seon Hong*
Dept. of Computer Engineering,
Kyung Hee University, 449-701
cshong@khu.ac.kr

Abstract

This paper is a contribution to the inception of multiple tokens in solving distributed mutual exclusion problem. The proposed algorithm is based on the token ring approach and allows simultaneous existence of multiple tokens in the logical ring of the network. Each competing 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 algorithm also handles the coincident existence of multiple critical sections (if any) in the system. The algorithm eliminates the idle time message passing, increases overall throughput and provides fault-tolerance. We discuss the impact of process failures and loss of tokens and propose corresponding recovery methods. The results of simulation show that the proposed algorithm overcomes the key limitations of the major token ring algorithms.

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 node 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 multiple 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 execution 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 makes it more distributed and adaptable to growing network size.

Our first goal is 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.

The remainder of the paper is organized as follows. Section 2 describes the existing related research works and the system model is introduced in section 3. The proposed algorithm is and its correctness proof are presented in Section 4, section 5 discusses on the failure cases and section 6 compares the performances of the algorithms. Finally, section 7 concludes the paper.

* Correspondent author
This work was supported by KISDI and MIC under the ITRC support program supervised by the IITA (IITA-2006-(C1090-0602-0002)

1550-445X/08 $25.00 © 2008 IEEE
DOI 10.1109/AINA.2008.38

2. Related Work

All the algorithms presented in this section claim to satisfy the mutual exclusion requirement, be deadlock free and starvation free. Their major characteristics and assumptions are discussed. The average waiting time per critical section entry, fault-tolerance and the number of messages exchanged for an entry to the critical section to take effect 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, 16]. 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 permissions 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 algorithm [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* node system.

In Raymond's tree-based algorithm [9], the token is always kept at the root node. When a node wants to enter the critical section, it sends a request to its parent. The parent sends a request to its parent, recursively, thus eventually the request reaches the root node. Upon receiving the request the root node sends the token down to the child that requested the token and is placed on top of the request queue. Once the node gets the token, it can enter the critical section. This logical tree structure remains unchanged, but the direction of its edges can change dynamically as the token propagates. In this algorithm, it requires an average of *2logN* messages for handling each request.

Nielsen and Mizuno extended [9] 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 requests with higher priority.

Although all these *O(log(N))* algorithms achieve better performance with respect to the average number of messages exchanged per critical section entry when compared to other mutual exclusion algorithms, their fault tolerance capability is very poor in the failures of nodes or communication links or in the cases of token loss. Availability of the system is not guaranteed during failures. Moreover, failure of root node causes loss of token *i.e.*, though the algorithm is designed for distributed mutual exclusion, the token management is centralized.

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 The System Model

3.1 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:

i. All processes in the system are assigned unique identification numbers.
ii. There may be more than one requesting processes from a node, mutual exclusion is implemented at the node level.
iii. Nodes may compete for multiple resource types at the same time.
iv. Process failures may be occurred.
v. 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. Message propagation delay is arbitrary but finite. The network may be of any topology. In software, a logical ring 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.

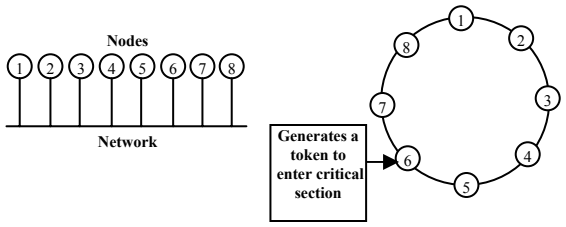

Fig. 1: Logical ring of unordered node on the network

3.2 Computation model

The token structure is as follows:

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 *Resource_ID*, 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 follows. Each node maintains the highest sequence number seen so far in a local variable *HighestSeqNumSeen.* When a node makes a request for CS, it uses a sequence number which is one more than the value of *HighestSeqNumSeen.* When a token requesting CS is received, *HighestSeqNumSeen* is updated as follows:

HighestSeqNumSeen=max(*HighestSeqNumSeen, 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**($TokenID_2$) *iif* $SN_1<SN_2$ 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 releases them after completion of CS execution. The request queue structure is as follows-

request queue { // a priority queue
token;
next_token; };

4 Proposed Algorithm

The proposed basic algorithm is presented first and then its enhancement and correctness proofs are described in the following subsections.

4.1 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 or anticlockwise.

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

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

i. If N_j has no intention to enter the CS(*r*), it replaces the sequence number by *HighestSeqNumSeen* and simply passes the token to the next node.
ii. If N_j is now in the CS(*r*), it puts the incoming token in Q_i^r. When N_j exits the CS, it releases the tokens to the next node sequentially (if any) from Q_i^r.
iii. 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.

Step-3. If the node N_i receives T_i^r, that is all other nodes allowed N_i to enter the 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.

Step-4. If 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 simple 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.

The proposed mutual exclusion algorithm for node N_i is presented below:

Algorithm 1: Multi-token Distributed Mutual Exclusion

```
HighestSeqNumSeen=0
1. loop (forever) {
2.  switch (event)
3.  event: CS(r) request received from
application process
4. HighestSeqNumSeen = HighestSeqNumSeen + 1
5.       create TOKEN = (TokenID, r, 0)
6.       if (timer currently not running)
7.            start timer
8.        send TOKEN to next node
9.  event: TOKEN received from other node
10. HighestSeqNumSeen=max(HighestSeqNumSeen,
SN)
11.  if (no TOKEN is generated by this
node OR (Pri(received TOKEN) >= Pri(N_1's
TOKEN) AND not executing CS now))
12. forward the TOKEN to next node
13. else if (own TOKEN is received back)
14.  enter the CS and execute it
15. else
16.  enqueue the TOKEN into the priority
queue
17. event: Completion of CS execution
18.      forward all tokens in Q(r)
19. event: timer timeout
20.      retransmit the token
21.      start timer
22. }  /* end of loop forever */
```

Fig. 2: Pseudo code of the proposed multi-token mutual exclusion algorithm for node N_i

4.2 Enhancement

The waiting time of lower priority tokens increases to higher values at node N_i whenever it's higher priority token, T_i^r, is lost repeatedly. Because, N_i does not forward any token until it successfully completes CS execution; rather on occurrence of timeout, it resends the token repeatedly. Therefore, we modify our algorithm presented in section 4.1 as follows. Each node has a local integer variable $RetryLimit^r$ for resource type r, and it is set to an initial value. If the number of retries crosses the $RetryLimit^r$, N_i releases all tokens queued in Q_i^r first and then send T_i^r with new sequence number. In our simulation, we get better response time with $RetryLimit^r$=2.

4.3 Analysis and 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.

Poperty 1. An algorithm provides fair mutual exclusion *iff* $\mathbf{Pri}(T_i^r) > \mathbf{Pri}(T_j^r) \Leftrightarrow N_j$ executes CS(*r*) after N_i finishes its execution.

Defiinition 1. Two token requests T_i^r and T_j^r are concurrent ($\| T_j^r$) *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 2. Given T_i^r, $CSet_i = \{T_j^r \mid T_i^r$ 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 1 (Safety and Fairness). The algorithm 1 in Fig. 2 provides safe and fair mutual exclusion as defined in property 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 generated token.
- N_i forwards other requests while it is in CS execution.
- Two request tokens have the same sequence number.

First condition can never be hold, because it contradicts with the restriction imposed in lines 13 and 14 of our algorithm. Second condition directly opposes the lines 15 and 16 of our algorithm which states that

any T^r received by N_i, while it is in CS(r) execution, is 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 line numbers 4 and 10, where new tokens are assigned highest sequence number so far seen plus one. Therefore, third condition also never holds.

Similarly, if $T_i^r \| T_j^r$ 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 requests, T_i^r. This situation will stand if at least one of the following conditions is hold:

- N_i enters CS before getting back its generated token.
- N_i forwards lower priority requests while it is intending to execute CS.
- Only a subset of nodes of the system takes part in decision.

First condition can never be hold, already proved above. The second condition contradicts with lines 11 and 12, where N_i forwards only higher priority tokens. 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 2 (Liveness). The algorithm 1 in Fig. 2 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 requests ever made till now. Theorem 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. Lines 4 and 10 in the algorithm force a newly arrived token to receive lowest priority, therefore progress property is preserved. Moreover, the 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.

Lemma 1. The algorithm 1 in Fig. 2 overcomes *idle time message passing*.

Proof. The most attractive feature of the proposed algorithm is that it does not pass any token around the ring when no node requires entering the CS; *i.e.,* idle period token passing is completely eliminated. It reduces communication overhead to a great extent.

Theorem 3 (Robustness). The loss of a token, T_i^r, generated by node N_i, does not deprive any other node from getting access to CS(r).

Proof. In the proposed algorithm, if two token requests T_i^r and T_j^r are concurrent ($T_i^r \| T_j^r$) and Pri(T_i^r)>Pri(T_j^r) then the service of the request T_i^r is not affected by T_j^r in no means; but the service of T_j^r is delayed by T_i^r until it finishes its CS execution. The loss of token T_i^r may deprive T_j^r from getting its service for indefinite period of time if and only if the algorithm is strict on the rule that even after repeated failure of T_i^r, the service of T_i^r must be completed before T_j^r. But the section 4.2 closes this unfairness and thereby it is proved that loss of a token does not deprive other tokens to get CS access.

5 Failure Handling

So far we have assumed that a node does not fail and communication media is reliable. But in real life, nodes and communication media are prone to failure and therefore tokens might be lost. In this section, we will describe the impact of above mentioned failures on the functioning of the proposed algorithm and suggest some ways to cope with these. We prove that despite of failures, the system continues to function properly without any disruptions.

5.1 Node Failure

If a node dies, there is no way to detect it so token could be lost. To avoid this unexpected situation, acknowledgement of receiving a token from the next node may be used. Thus node failure can easily be detected. At that 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 nodes. 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 and 15].

5.2 Loss of Token

Token sent by a node may be lost due to a number of causes – failure of a node having token(s) of other node(s), failure of communication media or the token packet is received in error. Loss of a token keeps its originator in *waiting for the token* state for a longer period of time. As described in section 4.1 (Step-4), after some timeout period the originator node resends the token with the initial timestamp value and thus ensures a live competitor token in the ring to get access of the requested resource. Assume that the token loss rate p stays constant during the controllable time frame, it can be shown that in a negative acknowledgement system, the probability of a successful delivery of a packet between two nodes that allows k retransmissions can be expressed recursively as

$(1-p) + p \times \Omega(k) \qquad (k >= 1)$

$\Omega(k) = \Phi(1) + \Phi(2) + \Phi(3) + \ldots \Phi(k)$

$\Phi(k) = (1-p)2 \times [\ 1 - p - \Phi(1) - \ldots - \Phi(k-1)\ ]\ (\Phi(0) = 0)$

Where $\Omega(k)$ is the probability of a successful recovery of a missing token within k retransmissions, $\Phi(k)$ is the probability of a successful recovery of the missing segment at k^{th} retransmission. Implementation of the algorithm and experimental results show that the loss of a token does not seriously affect the operation of our algorithm but might delay a bit the execution of the CS.

6 Performance Comparison

In consideration of all other comparative parameters, we have decided to compare our proposed protocol with Agrawal-Elabbaei's Token Ring algorithm [7] and Naimi-Trehel's algorithm [11]. We have implemented three algorithms (including proposed one) in a distributed computing system having 10 nodes. We have written five user programs, installed in node numbers 1 to 5, that need write access to a file TEST.TXT, located at node 8. The duration of CS execution is kept constant, 30ms. For the sake of simplicity, each of the nodes generated requests using a Poisson probability distribution with the same arrival rate. Further, all the algorithmic parameters, such as the request collection and request forwarding durations are the same at all the nodes. It was found that the algorithm worked efficiently *i.e.*, the targeted file TEST.TXT was updated by the running processes correctly in their allocated timeslots.

The algorithm is also verified for multiple critical sections *i.e.*, the processes are allowed to request for multiple resources simultaneously. Write requests to three files TEST1.TXT, TEST2.TXT and TEST3.TXT have been allowed by seven processes running at different nodes. The number of requests for the critical sections is kept 4 per 100ms. A single process is allowed for multiple critical sections entries, but not simultaneously.

Considering the packet sending, receiving and transmission times are constants, we have found that, in our algorithm, the average waiting time per CS entry increases at much slower rate as the increased number of requests, as shown in Fig. 3. Fig. 3 also tells that our algorithm performs better as CS requests increases. This is due to the use of unique token for individual CS access requests, which allows progress of token in ahead of time.

Fig. 4 illustrates that whenever there is no requests for CS, the proposed algorithm requires zero message passing while other algorithms experience highest (around 20) number of token message passing/unit time. This proves again that our algorithm reduces the idle time message passing to a great extent. On the otherhand, when load (no. of CS requests) increases, the average number of message passing is sharply dropped in existing algorithms while it is noticeably increased in our algorithm. In the first appearance, it may be assumed that it's a major drawback of the proposed algorithm, but that's not true. Since the tokens are able to travel a long way in ahead of time, this increased number of messages/unit time allows other processes to enter CS quickly, just a moment after the CS is released by some other process.

Fig. 5 states that the existing algorithms are greatly affected in the event of token loss, since their token-loss detection and recovery methods require much time. As described in section 4.1 (step-4) and section 5.2, our method is simple and efficient, does not require to execute any special/complex algorithm. Hence, the average waiting time is slightly increases on failure of tokens in our algorithm.

Fig. 6 states that as the number of failed nodes increases the rate of requests drops also increases for all the algorithms. The graph for our algorithm gives a little better performance, since in it the failure of a node does not cause the loss of other nodes' token.

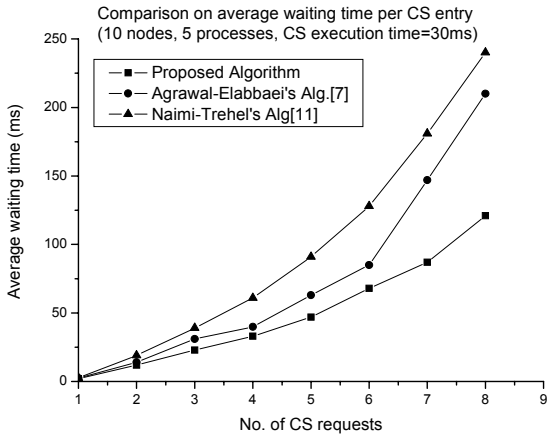

Fig. 3: Average waiting time for a process per CS entry

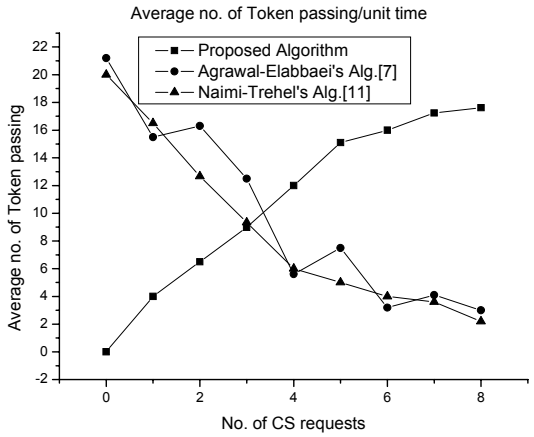

Fig. 4: Average no. of token passed per unit time

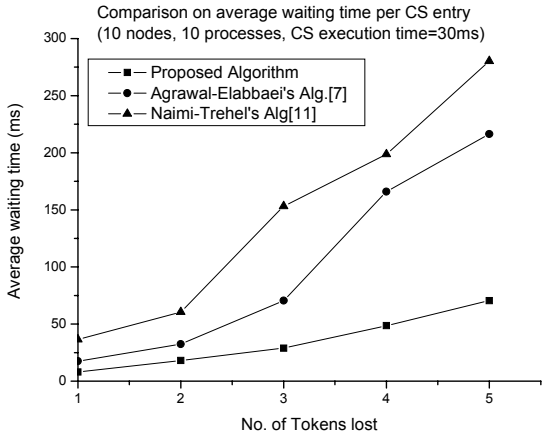

Fig. 5: Average waiting time per CS entry in presence of loss of Tokens

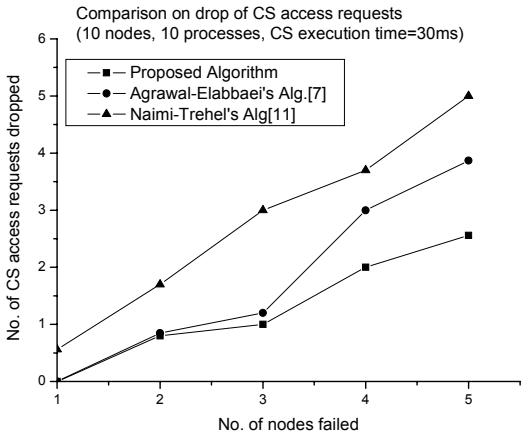

Fig. 6: Number of CS access requests dropped per node failure

7 Conclusions

The motivation towards the development of this algorithm is to present a method that guarantee mutual exclusion and works fairly and efficiently. The fault tolerance capability of this algorithm clearly makes it superior over the existing algorithms. Our algorithm tolerates network partitioning to a great extent. Considering the rapid growth of distributed systems, our presented method may provide a lucrative approach towards the solution of mutual exclusion problem with the cost of some computational overhead and increased number of message passing during highly loaded condition.

The distributed mutual exclusion algorithm presented in this paper is strictly distributed as defined by Lamport[4], Ricart-Agrawala[5] and Agrawal-Elabbaer[7]. However, in Maekawa[6], Suzuki-Kasami[8] and all tree-based algorithms[9,10,11,12 and 13], not every node participates in the decision to grant permission to access the critical section. They are partially distributed, the process of deciding the next token holder is not distributed but the task of decision making is distributed to each in turn. Finally, the algorithm is fair with respect to load balancing and scheduling of critical section, while it can support prioritized access.

References

[1] E.W. Dijkstra; Solution of a Problem in Concurrent Programming Control, Communication ACM, vol. 8, no. 9, Sept. 1965

[2] S. Lin, S. Q. Lian, M. Chen, and Z. Zhang, A Practical Distributed Mutual Exclusion Protocol in P2P Systems, Microsoft Research, Technical Report MSR-TR-2004-13
[3] M. Benchba, A. Bouabda Uah, N. Badache, and M. bed-Nacer: Distributed mutual exclusion algorithms in mobile ad hoc networks: an overview. ACM Operating Systems Review, vol. 38, no. 3. (July 2004) 74-89
[4] Lamport L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, vol. 21, no. 7. (July 1978) 558-565
[5] Ricart G. and Agrawala A.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, vol. 24, no. 1. (Jan. 1981) 9-17
[6] Maekawa M.: A sqrt(n) algorithm for mutual exclusion in decentralized systems. ACM Transactions on Computer Systems, vol. 3, no.2. (May 1985) 145-159
[7] Agrawal D., Elabbaei A.: An efficient and fault-tolerant solution for distributed mutual exclusion. ACM Transactions on Computer Systems, vol. 9, no. 1. (Feb. 1991) 1-20
[8] Suzuki I. and Kasami T.: A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems, vol.3, no.4. (Nov. 1985) 344-349
[9] Raymond, K.: A tree-based algorithm for distributed Mutual Exclusion. ACM Transactions on Computer Systems, vol. 7, no. 1. (Feb. 1989) 61-77
[10] Marin Bertier, Luciana Arantes, Pierre Sens: Hierarchical token based mutual exclusion algorithms. IEEE International Symposium on Cluster Computing and the Grid. (2004) 539-546
[11] M. Naimi, M. Trehel, and A Arnold: A log (N) distributed mutual exclusion algorithm based on path reversal. J. of Parallel and Distributed Computing. vol. 34, no.1. (Apr.1996) 1-13
[12] I. Chang, M. Singhal, and M.T.Liu: An improved log(N) mutual exclusion algorithm for distributed systems. Proceedings of the 1990 International Conference on Parallel Processing. (Aug. 1990) 295-302
[13] F. Mueller: Prioritized token-based mutual exclusion for distributed systems. Proceedings of 12th Intern. Parallel Proc. Symposium & 9th Symposium On Parallel and Distributed Processing. (March 1998) 791-795
[14] Kawsar F., H.S. Shariful, Saikat M. S., Razzaque M.A., Mottalib M.A.: An Efficient Token Based Algorithm for Mutual Exclusion in Distributed System. J. of Engineering and Technology, vol. 2, no. 2. (2003) 39-44
[15] 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 Fourth International Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT). (August 2003) 329-334