An Efficient Token Based Algorithm for Mutual Exclusion In Distributed System

Fahim Kawsar, Md. Shahariar Saikat, Dr. M. A. Mottalib.
Department of Computer Science and Information Technology
Islamic University of Technology

ABSTARCT

Many distributed computations involving the sharing of resources among various processes require that a resource be allocated to a single process at a time. Therefore, mutual exclusion is a fundamental problem in any distributed computing system. This problem must be solved to synchronize the access to shared resources in order to maintain their consistency and integrity. The major goal of this paper is to get the reader acquainted with a new approach towards the ring based technique for mutual exclusion in a distributed system. An algorithm is proposed based on the idea of generating token by the competing processes to enter the critical section and thus eliminating idle time message passing and reducing communication overhead.

Keywords: Mutual exclusion, critical section, token, ring structure.

1. INTRODUCTION

A distributed computing system is a collection of autonomous computing sites that do not share a global or common memory and communicate solely by exchanging messages over a communication facility. In a distributed computing system any given site (also referred to as "node") has only a partial or incomplete view of the total system and a system-wide common clock does not exist. Processes must share common hardware or software resources, cooperating in such a way that they can work in parallel and independently of each other. The access to a shared resource must be synchronized to ensure that only one process is making use of the resource at a given time. The problem of coordinating the execution of critical sections by each process is solved by providing mutually exclusive access in time to the critical section (CS). Each process must request permission to enter its critical section and must release it after it has completed its execution. A mutual exclusion algorithm must satisfy the following requirements [1, 2]:

i. At most one process can execute its critical section at a given time.
ii. If no process is in its critical section, any process requesting to enter its critical section must be allowed to do so at finite time.
iii. When competing processes concurrently request to enter their respective critical sections, the selection cannot be postponed indefinitely.
iv. A requesting process cannot be prevented by another one to enter its critical section within a finite delay.

To simplify, an algorithm must provide mutually exclusive access to the source, ensure deadlock freedom, ensure starvation freedom, and must provide some fairness in the order that requests are granted.

The algorithm presented in this paper is based on the token ring approach and satisfies the mentioned requirements in a way that minimize the communication overhead and ensure deadlock freedom, ensure starvation freedom. The competing nodes generate a token for the

permission to enter CS. The token traverses the logical ring structure. A node can enter CS if and only if it receives back its generated token.

The performance of the algorithm presented here will be evaluated using the total number of messages required for a node to enter the critical section as a criterion. Message traffic should be minimized in order to decrease the overhead in the communications network.

The remainder of the paper is organized as follows. Section 2 compares our work with existing research in distributed scheduling. The algorithm is presented in Section 3. Finally, Section 4 concludes the paper.

2. Previous Works

Two approaches have been used to implement a mutual exclusion mechanism in a distributed computing system. In a *centralized* approach, one of the nodes functions as a central coordinator. The central coordinator is fully responsible for having all the information of the system and for granting permission to make use of a shared resource. In a *distributed* approach, the decision-making is distributed across the entire system. 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.

2.1 Permission-based Algorithm

All the permission-based algorithms are introduced so far basically work in the same way. The node that wants to enter the critical section sends messages to other processors. Also, associated with each request there is a timestamp. When there are competitions for the critical section, the one with the lowest timestamp should enter first.

In **Lamport's event ordering mutual exclusion algorithm** [3], a node that wants to enter the critical section, broadcasts a message to all nodes in the system. The node that made the request enters the critical section if it received responses from all other nodes. After the node finished with the critical section, it again broadcasts a message to all other nodes. For a N node system a total of 3(N-1) messages are required to handle one request.

Ricart-Agarwala's algorithm [4] is very similar to Lamport's algorithm. The difference is that in Ricart-Agrawala's algorithm, the response message is deferred. Similar to Lamport's algorithm, Ricart-Agrawala's algorithm requires totally ordered events and all nodes being alive. About the number of messages, it does not need the release message, so it requires 2(N-1) messages for handling one request.

Maekawa's algorithm [5] associates a set of nodes is with each node, and this set has a nonempty intersection with every set associated with each other node. A node i must obtain permission from all other nodes in its home set Si before it can enter its critical section (CS). The number of messages required to handle a request is 3 times the size of the request set. For a system with N nodes, the size of each request set is roughly square root of N. So total $3\sqrt{N}$ messages are required to handle a request.

2.2 Token-based Algorithm

The simplest of token-based algorithm is the **Token Ring algorithm** [6]**.** In this algorithm, the nodes in the system form a logical ring. A token is passed around the ring. A node can enter the critical section if it holds the token. In average N/ 2 messages are required to handle one request in a N node system.

In **Suzuki-Kasami's broadcast algorithm** [7]. 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 the token. The algorithm requires N messages for handling each request.

In **Raymond's tree-based algorithm** [8] 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, this request will reach the root node. The root node, upon receiving the request sends the token down to the child that requested the token and is on top of the request queue. Once the node gets the token, it can enter the critical section. In this algorithm, it requires an average of 2logN messages for handling each request.

3. The Proposed Algorithm

Our 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 nodes in the system are assigned unique identification numbers from 1 to N.

ii. There is only one requesting process executing at each node. Mutual exclusion is implemented at the node level.

iii. Processes are competing for a single resource.

iv. At any time, each process initiates at most one outstanding request for mutual exclusion.

v. All the nodes in the system are fully connected.

The following aspects about the reliability of the underlying communications network should be considered.

- Message delivery guaranteed.
- Message-order preservation.
- Message transfer delays are finite, but unpredictable.
- The topology of the network is known.

The network may be of any topology with no inherent ordering of the processes. In software a logical ring is constructed in which each process is assigned a position in the ring. The ring positions may be allocated in numerical order of network addresses. All the matter is that each process knows who is next in line after itself.

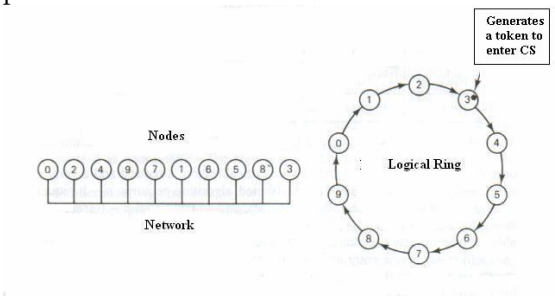

Fig 1: Logical ring of unordered node on the network.

The basic idea is that whenever a process needs to enter a critical section it generates a token and puts it in the network. On receiving a token a node reacts in the following manner

i. A process P wants to enter the critical section and generates a token, makes a copy of the token in it's node & passes it to the next node. The token structure is

```
token
{
   node no;
   timestamp;
};
```

ii. A process Q receives a token. It reacts to the token in the following ways:

- If Q has no intention to enter into the critical section it simply passes the token to the next node.

- If Q is in the critical section it puts the token in its request list. The request list structure is

request list
{
 token
 next list;
};

When Q exits the critical section it sends the tokens to the next node sequentially (if any) from its request list.

- If Q has already generated a token but not yet received that back it compares the incoming token's timestamp with it's generated token's timestamp. If it's token's timestamp is higher then it passes the token to the next node otherwise it puts the token in it's request list.
- If the timestamps in both the tokens are equal then Q looks for the node_no. If Q's node_no is lower (lowest number highest priority) then it adds the token in its request list otherwise passes it to the next node.

iii. If the process P receives a token, it compares the token with the stored token copy. If it matches then (that is no node wants to enter into the critical section) it enters into the critical section. When P exits the critical section it sends the tokens to the next node sequentially (if any) from its request list and deletes the associated copy and original token.

iv. If process P does not receive it's own generated token within a certain timeout period (token is lost), P resends the token with the initial timestamp.(The timestamp that is used while generating the first token, this ensures no starvation)

Since each node must receive its token eventually and at a time at most one node can have its token mutual exclusion is guaranteed.

The algorithm is free from deadlock as there is no way that a token traverse the ring indefinitely. Since time stamping is used so it ensures starvation freedom.

The timeout period for regenerating token must be designed in such a manner that unnecessarily no token should be resend. If multiple copies of the same token arrive at a originator node after it performs its operation in the CS or it is in the CS, it absorbs the token. (Since there is no copy of any token to be matched)

Another problem is that if a node dies there is no way to detect it so token can be lost. To avoid this acknowledgement of receiving a token may be used. Thus node failure can easily be detected. At that point the dead node can be removed from the group and the token holder can throw the token over the head of the dead process to the next node down the line. Of course, doing so requires that everyone maintains the current ring configuration.

From the working principle of this algorithm it is visible that for a N node system we need N messages to be transferred for handling one request of entering critical section.

3.1 PERFORMANCE GAIN

The most attractive feature of this newly proposed method is that there is no need for passing the token around the ring when no node requires it that is idle period token passing is

eliminated. It reduces communication overhead to a great extent.

So what is the performance gain of this proposed method? As a comparison let's see the following table:

Algorithm	Messages/Request
1.Lamport's Algorithm	3(N-1)
2.Ricart-Agarwals Algorithm	2(N-1)
3.Maekawas Algorithm	3√N or 5√N
4.Token Ring Algorithm	Avg N/2
5.Suzuki-Kasamis Broadcast Algorithm	N
6.Raymonds Tree-based Algorithm	Avg 2LogN
7.Ours Algorithm	N

N = Number of Node
Avg = Average

Fig 2: Comparison Table

So from the above table we find that this newly proposed algorithm will provide similar performance as some the existing techniques provide. But the attractive potentiality of this algorithm lies in error handling techniques. As in all token ring algorithm if the token is lost then detection and regeneration of the token is a big problem. Who will be responsible for regenerating the token and how to detect the loss of the token?

Our proposed method handles the loss of token in an efficient manner and as each node is responsible for its token so the responsibility of resending tokens completely lies on each node. In addition due to the elimination of idle time message passing communication overhead is reduced.

4. CONCLUSION

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. Considering the rapid growth of distributed system, our presented method may provide a lucrative approach towards the solution of mutual exclusion problem.

5. REFERENCES

[1] MAEKAWA, M.; OLDEHOEFT, A.E.; and OLDEHOEFT, R.R., "Operating Systems, Advanced Concepts," Benjamin-Cummings, 1987.

[2] SILBERSCHATZ, A. and PETERSON, J.L., "Operating System Concepts," Addison-Wesley, Alternate edition, 1988.

[3] LAMPORT, L., "Time, clocks, and the ordering of events in a distributed system," Communications of the ACM, vol. 21, no. 7, July 1978, pp. 558-565.

[4] RICART, G. and AGRAWALA, A., "An optimal algorithm for mutual exclusion in computer networks," Communications of the ACM, vol. 24, no. 1, Jan. 1981, pp. 9-17.

[5] MAEKAWA, M., "A sqrt(n) algorithm for mutual exclusion in decentralized systems," ACM Transactions on Computer Systems, vol 3, no. 2, May 1985, pp. 145-159.

[6] AGRAWAL, D., EL ABBADI, A., "An efficient and fault-tolerant solution for distributed mutual exclusion," ACM Transactions on Computer Systems, vol. 9, no. 1, Feb. 1991, pp. 1-20.

[7] SUZUKI, I. and KASAMI, T., "A distributed mutual exclusion algorithm," ACM Transactions

on Computer Systems, vol. 3, no. 4, Nov. 1985, pp. 344-349.

[8] RAYMOND, K., "A tree-based algorithm for distributed Mutual Exclusion," ACM Transactions on Computer Systems, vol. 7, no. 1, Feb. 1989, pp. 61-77.

[9] ANDREW S. TANENBAUM, "Distributed Operating Systems", Prentice Hall, 1995

[10] PRADEEP K SINHA", Distributed Operating Systems, Concepts and Design", IEEE Press, 1998