Computer & Robotics

An Improved Token-Based and Starvation Free Distributed Mutual Exclusion Algorithm

Om-Kolsoom Shahryari [a,*], Ali Broumandnia [b]

[a] Department of Computer Engineering, Sanandaj Branch, Islamic Azad University, Sanandaj, Iran
[b] Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

Received 31 January 2017; Revised 05 July 2018; Accepted 27 August 2018; Available online 18 September 2018

Abstract

Distributed mutual exclusion is a fundamental problem of distributed systems that coordinates the access to critical shared resources. It concerns with how the various distributed processes access to the shared resources in a mutually exclusive manner. This paper presents fully distributed improved token based mutual exclusion algorithm for distributed system. In this algorithm, a process which has owing token, could enter to its critical section. The processes communicate to each other in an asynchronous message passing manner. We assume the distributed processes are organized in a wraparound two dimensional array. Also, the communication graph of the network is supposed to be a complete graph. The proposed algorithm uses three types of messages, namely ReqMsg, InfoMsg and RelMsg. Beside token-holding node, there are some nodes, we call them informed-nodes, which can know token-holding node and transmit request message to it directly. The number of messages, which are exchanged per each critical section entrance, is a key parameter to avoid posing additional overhead to the distributed system. In this paper, we obtain to $3\sqrt{N} - 1$ messages per critical section access where N is the number of nodes in the system. The proposed algorithm outperforms other token based algorithms whilst fairness is kept and the proposed algorithm is starvation free.

Keywords: *Critical Section, Concurrency, Distributed System, Mutual Exclusion, Message Passing, Token-based Algorithm.*

1. Introduction

A distributed system consists of many independent process that communicate using message passing and collaborate to execute some task. In a distributed system, any given node has only a partial or incomplete view of the total system and a system-wide common clock does not exist [1]. In distributed systems, the resources are allowed to be shared. The mutual exclusion problem states that only a single process can be allowed to access a protected resource, also termed as a critical section, at any time. The mutual exclusion algorithms should have two main properties: (1): safety, by which at most one process can get the resource at each time, and (2): liveness, by which all process that want a resource can get it in a finite time [2], [3]. Mutual exclusion is a form of synchronization and one of the most fundamental paradigms in computing systems. The algorithms designed to ensure mutual exclusion in distributed systems are termed Distributed Mutual Exclusion (DME) algorithm. Distributed mutual exclusion algorithms is divided to two families, token-based and permission-based algorithms. In the token-based model, only the process that holds a privileged message called token can access the shared resource [4], [5], [6], [7]. The token is moved among the processes following a logical order. In

* Corresponding author. Email: shahryari.kolsoom@gmail.com

the permission- based model, each process must ask for permission to use the resource and will be allowed to access the resource only after getting permission from all other processes [3], [8].

Token-based algorithms exploit different solutions for the forwarding of critical section requests of processes and token transmission. Each solution is usually expressed by a logical topology that defines the paths followed by critical section request messages which might be completely different from the physical network topology. In proposed token-based algorithm, distributed process are organized in a wraparound two dimensional array. An entry request message, into the critical section (CS), is sent to the nodes horizontally in a row. On the other hand, the token vertically circulates in the array. The CS entry request are sent vertically down or vertically up with probability 1/2. In half of cases, the correct path was selected. Token continue to vertical travel until arrive at one of the nodes that know the token holding node, which named informed-nodes. Then the request sent directly to token-holding node.

The reminder of paper is organized as follows: In section 2, we discuss about related works and describe the problem and preliminary definitions. The proposed algorithm is presented in section 3. Section 4 proofs correctness properties and section 5 presents performance analysis. Finally, section 6 conclude the paper and points out directions of future work.

2. Related Work

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.

Each process has a code segment, called a critical section, in which the process can access the shared resource. 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. A process is said to execute repeatedly a sequence of non-critical section code and critical section code segments, each of finite execution time. 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 provide mutually exclusive access to a resource, ensure deadlock freedom, ensure starvation freedom, and must provide some fairness in the order that requests are granted.

Two approaches can be 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. Processes ask only the coordinator for permission to enter their critical section. Only when a requesting process receives permission from the coordinator can it proceed to enter its critical section.

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 and the solution to the mutual exclusion problem is far more complicated because of the difficulty to obtain a complete knowledge of the total system. This is due to the lack of a common shared memory, a common physical clock and because of unpredictable message delay.

A distributed mutual exclusion algorithm for multiple access channel was proposed in [10] that its time complexity is O ($\log_2 N$) where N is the number independent processes. In [11] intersection traffic control was modelled as a distributed mutual exclusion algorithm.

Distributed mutual exclusion algorithms can be classified into two groups by a basic principle in their design. These two groups are token-based algorithms and

permission-based algorithms. The basic principle for the design of a distributed mutual exclusion algorithm is the way in which the right to enter the critical section is formalized in the system.

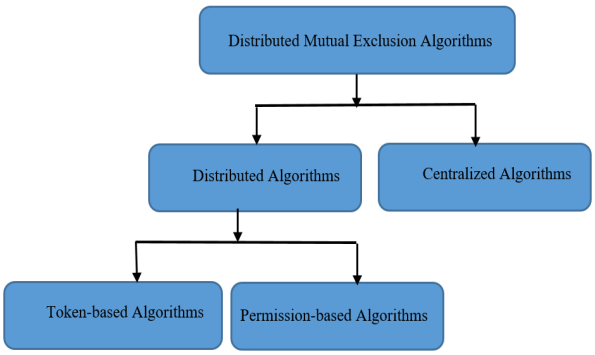

Fig. 1. Distributed Mutual Exclusion Algorithms.

2.1. The Token-based Approach

In the token-based group the right to enter a critical section is materialized by a special object, namely a token. The token is unique in the whole system. Processes requesting to enter their critical section are allowed to do so when they possess the token. The token gives to a process the privilege of entering the critical section. A token is a special type of message. The singular existence of the token implies the enforcement of mutual exclusion. Only one process, the one holding the token, is allowed to enter to its critical section.

At any given time, the token must be possessed by one process at most. Granting the privilege to enter the critical section is performed by a single process, which is the current owner of the token. This process chooses the next token owner and sends it the token. A distinction has to be made between the mechanisms used to move the token among the processes in the system. If processes are logically organized in a direct ring structure, the token can travel around the ring from process to process to give them the right to enter the critical section. If a process receives the token and it is interested in the critical section (CS), it can proceed to its execution. After the process exits its CS the token is released to circulate again. On the other hand, if the process is not interested in its CS it just passes the token to the next node in the logical ring. If the ring is unidirectional, starvation freedom is ensured. Under light load this algorithm has a high cost since the token message circulates even if no process wants to enter the CS, but it is very effective under high load.

Another method to move the token in the system is by asking for it when a process wants to enter its CS. A requesting process sends a request message to the token holder and waits for the token arrival. After completing the execution of its CS, the process holding the token chooses a requesting process and sends it the token. If no process wants to use the token; the token holder does not need to send the token away. Using this method, a major concern is how to locate the token holder in order to minimize message exchanges originated by a requesting process.

The token-based approach is highly susceptible to the loss of the token, since this can induce a deadlock situation. Also, problems can occur with the existence of duplicated tokens. Complex token regeneration must be executed to ensure the uniqueness of the token.

In [12] an effective token-based starvation-free priority-based mutual exclusion algorithm was proposed. For avoiding starvation, the priority of requests was increased dynamically. By postponing priority increment, the number of priority violations can be strongly reduced but at the expense of message overhead.

Kakugawa et al. [13] proposed a token-based distributed algorithm for the group mutual exclusion problem in asynchronous message passing distributed systems. The message complexity of proposed algorithm is O(|Q|) in the worst case, where |Q| is a quorum size that the algorithm adopts.

Tamhane and Kumar in [14] presented a token-based distributed mutual exclusion algorithm for accessing to a shared resource in opportunistic networks. They proved algorithm's correctness and its communication efficiency.

Bagchi in [15] showed that traditional distributed mutual exclusion algorithms which were used for accessing to shared resource, were not suitable for heterogeneous large scale mobile distribute systems and in some cases would lead to violate safety property of distributed critical section. He proposed a failure analysis model identifies such condition in analytical forms.

2.2. The Permission-based Approach

In the permission-based group the right to enter a critical section is formalized by receiving permission from a set of nodes in the system. A process wishing to enter its critical section asks the others to give it their permission to proceed; and then it waits until these permissions have arrived. A process enters its CS only after receiving permission from all nodes in a set. Non-requesting processes send their permission to requesting ones.

In [16] an autonomic distributed permission-based algorithm for k-mutual exclusion was presented. The proposed algorithm adapts itself dynamically to changes in the system composition. The proposed algorithm employs a hierarchical broadcast algorithm to propagate messages reliably and efficiently based on spanning trees which constructed in a fully distributed way.

In [17] an efficient permission-cum-cluster based distributed mutual exclusion algorithm was proposed for MANETs. The average number of messages/CS entry in proposed algorithm significantly is less than similar algorithms which it will lead to reduced communication delay in order to retrieve the required permissions.

Table. 1. Each node of system has these data structures

Data Structure	Definition
SN_i	A counter that process Pi increases by one whenever it attempts to invoke its CS, to indicate that there is a request from this process which is not responded.
$Waiting_i$	A FIFO queue of process P_i which is composed of not responded ReqMsgs.
CL_token_i	A variable to keep the node identifier of the current token-holding node. Also, when Pi is the token-holding process, if CL_tokeni equals i , it can enter its CS
Sr	A Boolean variable to determine that P_i is informed-node (Sr=1) or no (Sr=0).
Right and Up and Down	Every node knows its right, Up and down neighbours which are represented by constant identifiers placed in Right, Up and Down variables, respectively.

3. System Model

3.1. Assumptions

We assume an asynchronous message passing distributed system with a finite set Π of N > 1 nodes. Set Π is known by all processes. One process is executed per node, thus we use the terms node and process interchangeably. Processes are organized on a logical topology that is based on wraparound two-dimensional array with N nodes that each row has $d = \sqrt{N}$ nodes and each column has $d = \sqrt{N}$ nodes too.

A priority or an order of events has to be established between competing requesting processes so only one of them receives permission from all other nodes in the set.

Only one process, the one that has received permission from all members of a given set of nodes, is allowed to enter the critical section. This enforces the requirement for mutual exclusion. Granting the privilege to enter the critical section is performed by the set of nodes that send their permission to requesting processes. Conflicts are solved by a priority or an order of events mechanism.

The problem of finding a minimal number of nodes from which a process has to obtain permission to enter its CS has to be considered. This can be translated as to how many rights a process has to collect in order to proceed to the execution of the critical section. Many protocols have been developed to find a majority or quorum of processes from which rights have to be collected. The solution to this problem has a direct impact in the cost of messages exchanged per mutual exclusion invocation [9].

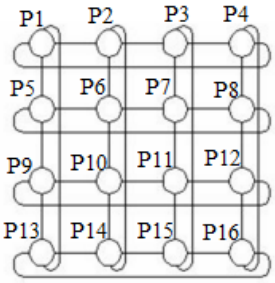

Fig. 2. Two dimensional wrap-around array network.

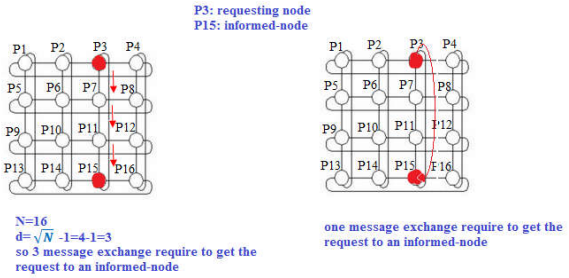

Fig. 3. Compare proposed algorithm by similar former algorithm [12].

Initially, a unique identification number between 1 to N is randomly assigned to each process. There were some assumptions that we considered in proposed algorithm:

- It supposed that the communication graph of the network is complete graph. On the other hand, there is a channel between each arbitrary pairs nodes P_i and P_j.
- The links are assumed to be reliable. It means that each transmitted message get to destination node surely.
- For any two processes P_i and P_j, the messages sent from process P_i to process P_j are received in the same order as they are sent.
- There is only one process in each node.
- Each process cannot create new CS entering request until the prior are granted.

3.2. Proposed Algorithm

We call the proposed algorithm as info-based algorithm; because of some nodes that we name them informed-nodes have information about the location of token. Informed-node is a node in the same row of token-holding node. As informed nodes know the location of token, when entry request message arrives them, they send the request to token-holding node directly. In the proposed algorithm each node can create an entry request message. This request moves along column upward and downward by probability of 0.5 and at last arrives to an informed-node. By this way the maximum steps required to get message to an informed-node is $\sqrt{N}/2$. When Request message reach to an informed-node, it sends the request to token-holding node along the row directly.

In another work [18] the request sends downward along the column that requires in the worst case d-1 or $\sqrt{N}-1$ message passing until get to an informed-node. In our improved inform-based algorithm, in the worst case $\sqrt{N}-1$ message were exchanged and in the average case $\frac{\sqrt{N}-1}{2}$ messages were exchanged. So in average case, the number of message exchange in the later is behalf of former work.

In the proposed algorithm, token is a data structure [5] that defines as Fig.4.

Table. 2. Description of message types

Message Type	Description
ReqMsg (i, $Waiting_i$)	A message which is sent by process Pi to invoke its CS. It is composed of the identification number of the process i, and its P_i's local view of not responded messages (Waiting i)
InfoMsg (i)	This message informs the receiving node that process Pi is the explicit token-holding node.
RelMsg (i)	This message requests from its receiver node to inform all nodes in its row that it is not the token-holding node, any more.

Table 1 shows the data structures that we use in this algorithm. Furthermore, in Table 2 we descript some message types used by proposed algorithm: ReqMsg, InfoMsg, RelMsg. Type of messages in the proposed algorithm is fewer that similar algorithm proposed in [12]; this leads to reduce message complexity. In the following of the paper, we assume that process Pk is a token-holding node and process Pi is a node that has request to execute its CS.

Now we descript the algorithm in more detail. Process Pi can execute its CS whenever it receives the token. Now P_i sets CL_tokeni to i.

Now we consider a situation that process P_i requests its CS. Pi must increase SNi a unit. Then it set $Waiting_i[i]$ by SN_i and creates ReqMsg (i, $Waiting_i$). If process P_i is an informed node, sends ReqMsg (i, $Waiting_i$) to token-holding node directly. Otherwise, P_i generate a random number been 0 and 1. If generated number less than 0.5, it sends ReqMsg (i, $Waiting_i$) to its Up and if it greater than 0.5, sends ReqMsg (i, $Waiting_i$) to its Down. Thus ReqMsg (i, $Waiting_i$) starts its vertical movement until it arrives to one of the informed-nodes. Fig. 5 illustrates creating request to enter critical section. When a process P_j receives a ReqMsg from a P_i, several cases may be occurred. Fig. 6. Shows all of these situations.

Let us consider another scenario, when P_i receives a message. This message can have seven types:

ReqMsg, RelMsg and Token. Table 3 shows the various situations of receive a message by P_i.

Also, P_k by getting token message, be allowed to enter its CS. At this time, P_k should notify all node of its row to owing the token. Fig. 7. Shows steps of P_k's CS entering.

Furthermore, P_k after executing CS should release CS. It means that P_k sets CL_token_k to zero and sends RelMsg to informed-nodes in its row to show that it no longer hold the token. As P_k gets permission to enter its CS by ReqMsg (k, $waiting_k$) at time stamp SN_k, at the time of leaving CS, copies SN_k to Q[k] which means that the last granted P_k's request message is occurred at SN_k. Leaving the CS by P_k has been shown at Fig. 8. In addition, P_k should inspect Q and $Waiting_k$ to find next node for sending token. As depicted in Fig. 9, P_k investigates Q circularly to find P_i such that Pending_request[i]> Q[i]. Based on this scheme all of processes enter their CS in turn. On the other hand, each process waits limited time to enter its CS. It is obvious that proposed algorithm is starvation-free.

When a token-holding node receives a request message (i, $waiting_i$), it updates pending-request by comparing $waiting_i[i]$ and pending-request[i] one by one (Fig. 10).

Table. 3. Various situations of receive a message by Pi.

Message Type	Description
ReqMsg (j, waitingj)	If (Pi is a token-holding node) then update pending_requests.
	Send the ReqMsg to token-holding if (Pi is an informed-node) then node.
	Else Until ReqMsg arrived at an informed-node, ReqMsg continues to move upward or downward.
Token	Pi become an explicit token-holding node and inform all nodes in its row.
infoMsg(j)	if (process Pi does not token-hold node) then first it sets CL_tokeni to j
	So, process Pi become as an informed-node and set Sr=1
	Send InfoMsg(j) to its Right.
	If (InfoMsg(j) has arrived back to j then Pj stops sending infoMsg(j).
RelMsg(j)	Pi first set Sr=0 and then set CL_tokeni=0
	Send RelMsg(j) to its Right

```
Struct token
{
    Queue Q [N]; // An array with N elements that holds the time
    stamp of last request of each process that had been realized.
    Queue pending_requests [N]; // An array with N elements that
    each element i of it holds the time stamp of last responded
    requests of process Pi
    int idexp; // Shows identification of the token-holding node.
    int rowexp; // Shows the index of token-holding node' row.
    int Current_SN;// Shows the time stamp of current request
    message that be serviced;
}
```

Fig. 4. Token data structure.

4. Correctness Properties

Correctness of distributed mutual exclusion algorithm can be proved by proofing safety and liveness.

4.1. Safety

Safety is guarantying mutual exclusion. On the other hand, safety is being assured of preventing concurrent access to critical section by several process.

On the one hand there is only a token in the whole of distributed system and on the other hand, a process can access to its CS if and only if gets token. So, only one

process can access to its CS in each time. Thus it's impossible simultaneous access to CS by several processes. Therefore, safety is guaranteed.

4.2. Liveness

Liveness is guaranteed if none of processes meet starvation and deadlock does not occur in the system.

As illustrated in Fig. 11, when process Pj leaves its CS, it should inspect Q and Pending_requests arrays in order to find next node which the token must be sent to it. Proposed algorithm uses circular search from j to find next node such that it has at least one not responded request. Additionally, at the proposed algorithm each process can access its CS consecutively only once. So, all processes endure limited waiting time to catch token. Thus, processes access to the CS one after another. Therefore, the proposed algorithm is starvation-free and deadlock does not happen in the system. So, liveness is assured.

5. Performance Evaluation

The processes can communicate with each other by message passing in the distributed systems. So, one of the significant parameter for evaluation distributed algorithm performance, is message complexity. Message complexity shows the number of messages which should be exchanged to complete the algorithm. So, we focus on number of messages which will be exchanged per each request to CS entrance.

The performance of distributed mutual exclusion often studied consideration to two scenarios: light load and heavy load.

5.1. Light Load

In this scenario, there is only a process that wants to enter its CS. Consider P_i as an applicant node, is neither token-holding node nor informed-node. In the worst case, P_i situated at the farthest distance to an informed node. So, the request of P_i (ReqMsg(i, $Waiting_i$)) achieves to informed node by $\sqrt{N}-1$ exchanged messages. Then the informed-node sends the ReqMsg (i, $Waiting_i$) directly to token-holding node. When Pj leaves its CS, sends RelMsg(j) to all of nodes in its row to announce them to that P_j isn't token-holding node anymore. So $\sqrt{N}-1$ messages are required. Since, there is no request message except request of P_i in pendin_requests, P_j (token-holding node) sends the token to P_i by a message. As soon as Pi receives the token, informs all nodes in its row as informed-nodes by sends infoMsg(i) to them. So, $\sqrt{N}-1$ messages are needed to inform all nodes of a row except token-holding node itself. Then Pi can enter its CS. Therefore,

$$\left(\sqrt{N}-1\right)+1+\left(\sqrt{N}-1\right)+1+\left(\sqrt{N}-1\right) = 3\sqrt{N}-1 \quad (1)$$

Messages must be exchanged per each CS entrance in light load situation which are fewer than similar prior algorithms.

5.2. Heavy Load

Suppose P_j is a token-holding node and situated at row r. In the heavy load scenario, all nodes want to grab token for entering their CSs simultaneously. Even, when P_j leaves its CS, it creates reqMsg again for entering its CS. Each process Pi create its request message by ReqMsg (i, Waitingi) and transmits it vertically (upward or downward) if Pi isn't token-holding node and informed-node too. Since each process has request for invoking token itself, when receives a message from other nodes from next or prior row, it appends the received request messages to its Waiting and sends only one message instead of two or more messages. Since there are $\sqrt{N}$ columns in the given system, and $\sqrt{N}$ messages are exchanged in each column, therefore $\sqrt{N}\times\sqrt{N}=N$ messages are exchanged. All nodes of row r except Pj send received messages and their request messages to token-holding node directly. So, $\sqrt{N}-1$ messages are passed in this step. Due to starvation-free property of proposed algorithm, Pj after leaving its CS should send the token to another process. Therefore, P_j must send RelMsg(j) to all nodes in row r which means that P_j isn't token-holding node more. In this step, $\sqrt{N}$ messages are passed. Further P_j finds the next applicant node (suppose P_k) and sends the token to it by a message. After the receiving token by P_k, it sends infoMsg(k) to all nodes in its row to inform them based on P_k is token-holding node now. The number of total messages which are exchanged

in heavy load case is $N + 3\sqrt{N} - 2$. The number of message exchange per each CS request is:

$$\frac{N + 3\sqrt{N} - 2}{N} = 1 + \frac{3}{\sqrt{N}} - \frac{2}{N} \quad (2)$$

The number of messages which should be exchanged in the proposed algorithm are compared with similar algorithm which presented at [12] and are summarized in Table 4, under these two kinds of loads.

As mention above, the number of message exchanges in proposed algorithm is lower than similar info-based algorithm [12] whilst the number of message types which used by proposed algorithm is fewer than [12]. Also, proposed algorithm uses simple data structures and messages.

```
CS_REQUEST
SNi=SNi+1;
Waitingi[i]= SNi
Create new request Message ReqMsg (i, Waitingi)
if CL_tokeni==i then
        Pi runs Enters_CS(i)
else
        if Sr==1
                Send ReqMsg directly to Token-Holding node
        else
                P=Select a random number in (0,1);
                if P < 0.5 then
                        Send ReqMsg downward
                else
                        Send ReqMsg Upward
```

Fig. 5. A pseudo code for Process Pi.

```
When a Process Pj receive a ReqMsg from Pi
There are three possible state:
        if Pj isn't a token holding node
                if Pj isn't an informed-node then
                        if Pj has request Message to enter CS too
                                run CS_REQUEST and Update Waitingj[i] to Waitingi[i] and then send ReqMsg (j, Waitingj)
                                vertically; // Send only one request that leads to reduce the number of message exchanges.
                        if Pj hasn't request Message to enter CS
                                transmit message vertically to next node
                if Pj is an informed-node then
                        if Pj has request Message to enter CS too
                                run CS_REQUEST and Update Waitingj[i] to Waitingi[i] and then send ReqMsg (j, Waitingj)
                                directly to token_holding node; // Send only one request that leads to reduce the number of
                                message exchanges.
                        if Pj hasn't request Message to enter CS
                                transmit message directly to token_holding node
        if Pj is a token holding node
                run RCV_request ();
                if Pj is leaving the CS
                        run Pj leaves CS
                else
                wait until CL_tokenj=0
```

Fig. 6. Process Pj receives ReqMsg from Pi

```
Enters_CS (int k)        // Pk Enters its CS
Token.idexp=k;
Token · rowexp = [k/√N] + 1;
Send InfoMsg(k) to all of nodes in Token · rowexp except Pk
CL_tokenk=k;
Pending_requests[k]=Pending_request[k]-1;
Current_SN= ReqMsg.SNk ;
Pk run its CS;
```

Fig. 7. P_k Enters to its CS

```
Pk leaves CS
Token.Q[k]=Current_SN;
CL_tokenk=0;
Token.idexp=0;
Pk sends RelMsg(k) to all of nodes in Token · rowexp
except Pk
nextNode= find_Node_To_Send_Token (k);
send Token to nextNode;
```

Fig. 8. P_k leaves its CS

```
find_Node_To_Send_Token (int j)
for i=j+1 to N
        if pending_requests [i] > Q [i]
                return i;
for i=1 to j-1
        if pending_requests [i] > Q [i]
                return i;
```

Fig. 9. Token-holding node run find_Node_To_Send_Token to find next node.

```
RCV_request
//when a token_holding node receive a request message from Pj
For i=1 to N
        if Waitingj[i] >pending_requests[i]
                pending_request[i]=Waitingj[i];
endfor
```

Fig. 10. Token-holding node updates pending-requests when receives a new ReqMsg.

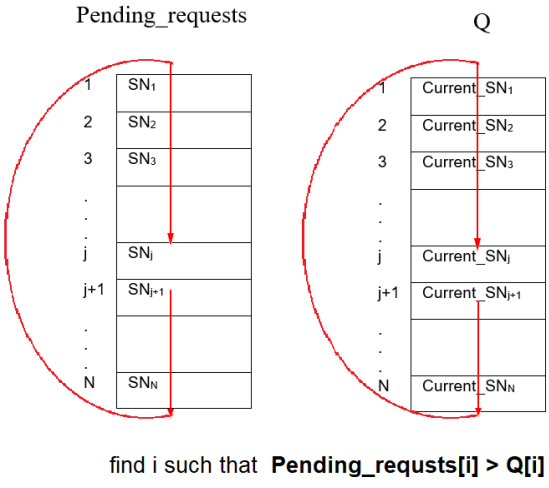

Fig. 11. Circular search Pending_requests array to fine next node to send token.

Table. 4. A comparison based on message complexity.

Algorithms	**Message Complexity (per each CS invocation)**	
	Light Load	**Heavy Load**
Info-based algorithm presented at [12]	$4\sqrt{N}+1$	$2+\frac{4}{\sqrt{N}}-\frac{1}{N}$
Proposed Algorithm	$3\sqrt{N}-1$	$1+\frac{3}{\sqrt{N}}-\frac{2}{N}$

6. Conclusion and Future Work

A token-based algorithm for solving distributed mutual exclusion problems are proposed in this paper. The process which owing the token can run CS. Also, the proposed algorithm called info-based because of some nodes know the token-holding node. We assumed a

logical topology in the form of wraparound two-dimensional array, which token is getting around between all processes. To guarantee mutual exclusion to access to shred resource, some message types are used. Request messages move vertically until arrive to informed-nodes. As informed-nodes know the token, they send request to token-holding node directly. By this way, we can reduce the number of message exchanges in the system. Since there is only a token in the whole of system and only a process can grab the token at any given time: So, the safety is assured. In the proposed algorithm, a process can not run CS more than once successively. So, liveness is assured too and the proposed algorithm is starvation-free.

The Performance analysis done shows that, the proposed algorithm needs $3\sqrt{N} - 1$ message exchange in light load situation, which outperforms other algorithms. When the number of process goes to infinity and all of them want to run CS, the number of message exchange per each CS entry will be close to 1 at heavy load situation.

In future research, it will be necessary to propose a distributed mutual exclusion algorithm for other network topology such as tree, mesh, ring and hyper-cube.

References

[1] Velazquez, M., "A survey of distributed mutual exclusion algorithms", Technical Report CS-93-116, Colorado State University, (1993).

[2] Kshemkalyani, A. D.; Singhal, M., "Distributed mutual exclusion algorithms, in Distributed Computing: Principles, Algorithms, and Systems", 1st ed. Cambridge University Press (2008).

[3] Raynal, M.; Beeson, D., "Algorithms for mutual exclusion Cambridge, MA, USA: MIT Press (1986).

[4] Le Lann, G., "Distributed systems - towards a formal approach", in IFIP Congress, pp. 155–160 (1977).

[5] Suzuki, I.; Kasami, T., "A distributed mutual exclusion algorithm, ACM Trans. Comput. Syst., vol. 3, pp. 344–349 (1985).

[6] Raymond, K., "A tree-based algorithm for distributed mutual exclusion", ACM Trans. Comput. Syst., vol. 7, pp. 61–77 (1989).

[7] Naimi, M.; Trehel, M.; Arnold, A., "A log (n) distributed mutual exclusion algorithm based on path reversal", J. Parallel Distrib. Comput, vol. 34, pp. 1–13 (1996).

[8] Sanders, B. A., "The information structure of distributed mutual exclusion algorithms", ACM Trans. Comput. Syst., vol. 5, pp. 284–299 (1987).

[9] Velazquez, M. G., "A Survey of Distributed Mutual Exclusion Algorithms", Velazquez Technical Report CS-93-116 September 6, (1993).

[10] Czyzowicz, J.; Gasieniec, L.; Kowalski, D. R.; Pec, A., "Consensus and Mutual Exclusion in a Multiple Access Channel", IEEE Transactions on Parallel and Distributed Systems, vol. 22, pp. 1092-1104 (2011).

[11] Wu, W.; Zhang, J.; Luo, A.; Cao, J., "Distributed Mutual Exclusion Algorithms for Intersection Traffic Control", IEEE Transactions on Parallel and Distributed Systems, vol. 26, pp. 65-74 (2015).

[12] Lejeune, J.; Arantes, L.; Sopena, J.; Sens, P., "A fair starvation-free prioritized mutual exclusion algorithm for distributed systems", J. Parallel Distrib. Comput., vol. 83, pp. 13–29 (2015).

[13] Kakugawa, H.; Kamei, S.; Masuzawa, T., "A Token-Based Distributed Group Mutual Exclusion Algorithm with Quorums", IEEE Trans. Parallel Distrib. Syst., vol. 19, no. 9, pp. 1153–1166 (2008).

[14] Tamhane, S. A.; Kumar, M., "A token based distributed algorithm for supporting mutual exclusion in opportunistic networks", Pervasive Mob. Comput, vol. 8, no. 5, pp. 795–809 (2012).

[15] Bagchi, S., "Design and topological analysis of probabilistic distributed mutual exclusion algorithm with unbiased refined ordering", Future Generation Computer System, vol. 95, pp. 175–186 (2019).

[16] Rodrigues, L. A.; Duarte, E. P.; Arantes, L., "A distributed k-mutual exclusion algorithm based on autonomic spanning trees", J. Parallel Distrib. Comput., vol. 115, pp. 41–55 (2018).

[17] Gupta, A.; Saini, P.; Krishna, C. R., "An efficient permission-cum-cluster based distributed mutual exclusion algorithm for mobile adhoc networks", in 2014 International Conference on Parallel, Distributed and Grid Computing, pp. 141–146 (2014).

[18] Taheri, H.; Neamatollahi, P.; Naghibzadeh, M., "Info-based approach in distributed mutual exclusion algorithms", Journal of Parallel Distribution and Computation vol.25, no. 2 , pp. 650–665 (2012).