

Distributed Mutual Exclusion Algorithms on a Ring of Clusters

Kayhan Erciyes

California State University San Marcos,
Computer Science Dept., 333 S.Twin Oaks Valley Rd.,
San Marcos CA 92096, U.S.A.
kerciyes@csusm.edu

Abstract. We propose an architecture that consists of a ring of clusters for distributed mutual exclusion algorithms. Each node on the ring represents a cluster of nodes and implements various distributed mutual exclusion algorithms on behalf of any member in the cluster it represents. We show the implementation of Ricart-Agrawala and a Token-based algorithm on this architecture. The message complexities for both algorithms are reduced substantially with this architecture as well as obtaining better response times due to parallel processing in the clusters

...

1 Introduction

Mutual exclusion in distributed systems is a fundamental property required to synchronize access to shared resources in order to maintain their consistency and integrity. Comprehensive surveys about mutual exclusion are given in [8] [1]. For a system with N processes, competitive algorithms have message complexities between $\log(N)$ and $3(N - 1)$ messages per access to a critical section (CS). The distributed mutual exclusion algorithms may be broadly classified as permission based or token based. In the first case, a node would enter a critical section after receiving permission from all of the nodes in its set for the critical section. In the second case, the possession of a system-wide unique token would provide the right to enter a CS. For token based algorithms, a logical ring of processes may be constructed and a token is passed around the ring. The token holder gets the permission to access the critical section. These types of algorithms are considered fair, and have bounded waiting. The token based approach is highly susceptible to the loss of a token as this would result in a deadlock. Susuki-Kasami's algorithm [10] (N messages), Singhal's heuristic algorithm [9] ($N/2$, N messages) and Raymond's tree based algorithm [5] ($\log(N)$ messages) are examples of token based mutual exclusion algorithms. Examples of nontoken-based distributed mutual exclusion algorithms are Lamport's algorithm [3] ($3(N-1)$ messages), Ricart-Agrawala (RA) algorithm ($2(N-1)$ messages) [6] and Maekawa's algorithm (\sqrt{N} messages) [4].

The requirements for any distributed mutual exclusion algorithm are 1. At most one process should be executing in the critical section (safety), 2. request to

enter or exit the critical section will eventually succeed (liveness) and 3. If one request is issued before another, then the requests will be served in the same order (fairness). Lamport’s algorithm [3] and RA algorithm [6] are considered as one of the only fair distributed mutual exclusion algorithms in literature. Lamport’s algorithm requires $3(N-1)$ messages and RA algorithm optimizes Lamport’s algorithm and requires $2(N-1)$ messages per critical section access.

In this study, we propose an architecture where coordinators for clusters of nodes are placed on a ring. These coordinators perform the required critical section entry and exit procedures for the nodes they represent. This model is semi-distributed as we have central components. However, these components are homogenous and communicate with each other asynchronously. Using this architecture, we show that RA and Token-based algorithms may achieve an order of magnitude reduction in the number of messages required to execute a critical section at the expense of increased response times and synchronization delays. The rest of the paper is organized as follows. Section 2 is a review on the background of performance metrics for fundamental mutual exclusion algorithms. The RA algorithm on the proposed model, *RA_Ring* is described in Section 3 with the analysis of the achieved performance metrics. The second algorithm implemented on the model uses Token Passing and is called *Ring_TP* as described in Section 4 with performance considerations. Finally, discussions and conclusions are outlined in Section 5.

2 Background

2.1 Performance Metrics

Performance of a distributed mutual exclusion algorithm depends on whether the system is *lightly* or *heavily* loaded. If no other process is in the critical section when a process makes a request to enter it, the system is lightly loaded. Otherwise, when there is a high demand for the critical section which results in queueing up of the requests, the system is said to be heavily loaded. The important metrics to evaluate the performance of a mutual exclusion algorithm are the number of messages per request, response time and the synchronization delay as described below :

- *Number of Messages per Request (M)* : The total number of messages required to enter a critical section is an important and useful parameter to determine the required network bandwidth for that particular algorithm. M can be specified for high load or light load in the system as M_{heavy} and M_{light} .
- *Response Time (R)* : The Response Time R is measured as the interval between the request of a node to enter critical section and the time it finishes executing the critical section. When the system is lightly loaded, two message transfer times and the execution time of the critical section suffices resulting in $R_{light} = 2T + E$ units. Under heavy load conditions, assuming at least one message is needed to transfer the access right from one node to another, $R_{heavy} = w(T + E)$ where w is the number of waiting requests.

- *Synchronization Delay (S)* : The synchronization delay S is the time required for a node to enter a critical section after another node finishes executing it. The minimum value of S is one message transfer time T since one message suffices to transfer the access rights to another node.

The lower bounds for M , R and S are shown in Table 1.

Table 1. Lower Bounds for Performance Metrics [7]

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
3	3	$2T + E$	$w(T + E)$	T

2.2 Ricart-Agrawala Algorithm

The Ricart-Agrawala (RA) Algorithm represents a class of decentralized, permission based mutual exclusion algorithms. In RA, when a node wants to enter a critical section, it sends a timestamped broadcast *Request* message to all of its peers in that critical section request set. When a node receives a *Request* message, it returns a *Reply* message if it is not in the critical section or requesting it. If the receiving node is in the critical section, it does not reply and queues the request. However, if the receiver has already made a request, it compares the timestamp of its request with the incoming one and replies the sender if the incoming request has a lower timestamp. Otherwise, it queues the request and enters the critical section. When a node leaves its critical section, it sends a reply to all the deferred requests on its queue which means the process with the next earliest request will now receive its last reply message and enter the critical section. The total number of messages per critical section is $2(N-1)$ as $(N-1)$ requests and $(N-1)$ replies are needed. One of the problems with this algorithm is that if a process crashes, it fails to reply which is interpreted as a denial of permission to enter the critical section, so all other processes that want to enter are blocked. Also, the system should provide some method of clock synchronization between processes. The performance metrics for the RA Algorithm are shown in Table 2. When a node finishes execution of a critical section, one message is adequate for a waiting node to enter, resulting in $S = T$.

Table 2. Performance Metrics for Ricart-Agrawala Algorithm

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
$2(N - 1)$	$2(N - 1)$	$2T + E$	$w(T + E)$	T

2.3 Token-Based Algorithms

The general Token Passing (TP) Algorithm for mutual exclusion is characterized by the existence of a single token where the possession of it denotes permission to enter a critical section. The token circulation can be performed in a logical ring structure or by broadcasting [10]. In a ring based TP Algorithm, any process that requires its critical section will block the token and issue it when it finishes executing. Fairness is ensured in this algorithm as each process waits at most $N-1$ entries to enter the critical section. There is no starvation since passing is in strict order. The main difficulties with TP Algorithm are as follows. There would be the idle case of no processes entering CS would incur overhead of constantly passing the token. There could be lost tokens which would require diagnosis and creating a new token by a central node or distributed control is needed and to prevent duplicate tokens, central coordinator should ensure generation of only one token. Crashes should also be dealt with as these would require detection of the dead destinations in the form of acknowledgements. One important design issue with TP Algorithm is the determination of the holding time for unneeded token. If this time is too short, there will be high overhead. However, keeping this time too long would result in high CS latency. The performance metrics for a general Token-Based Algorithm is shown in Table 3. We assume a general case here where $N-1$ messages to solicit for the token and 1 reply message from the holder are needed.

Table 3. Performance Metrics for General Token-based Algorithms

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
N	N	$2T + E$	$w(T + E)$	T

3 Ricart-Agrawala Algorithm on the Ring

We propose the architecture shown in Fig. 1 where nodes form clusters and each cluster is represented by a *coordinator* in the ring. Coordinators are the interface points for the nodes to the ring and election of a new coordinator is provided as in [2] if it crashes. The relation between the cluster coordinator and an ordinary node is similar to a central coordinator based mutual exclusion algorithm. The types of messages exchanged are *Request*, *Reply* and *Release* where a node first requests a critical section and upon the reply from the coordinator, it enters its critical section and then releases the critical section. However, the coordinator in this case has to provide more sophisticated functionality as it has to communicate with the other coordinators.

The finite state diagram of the coordinator is depicted in Fig. 2. When a node makes a request for a critical section (*Node_Req*), the coordinator sends a

critical section request (*Coord_Req*) to the ring and sets its state to WAITRP. If there are any other requests from its cluster in this state, it sends a request message (*Coord_Req*) to the ring for each one and store these pending requests in its cluster. When it receives external requests (*Coord_Req*) in this state, it performs the operation of a normal RA node by checking the timestamps of the incoming requests by the pending requests in its cluster and sends a reply (*Coord_Rep*) only if all of the pending requests have greater timestamps than the incoming request. It goes back to the IDLE state only if there are not any other pending request in its cluster. For any pending request, it goes back to the WAITRP state to wait for coordinator replies.

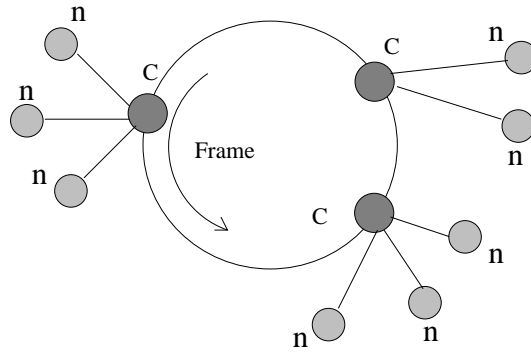


Fig. 1. The Architecture, n: Node, C: Coordinator

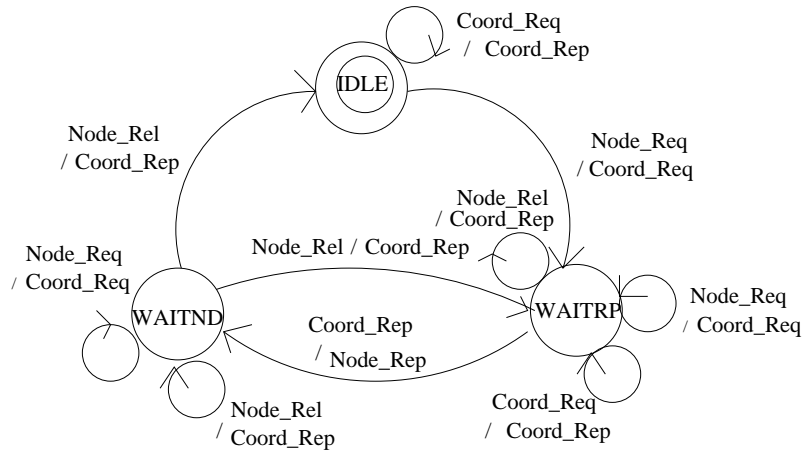


Fig. 2. The Coordinator for the Ring_RA Algorithm

Fig. 3 shows an example scenario for the Ring_RA Algorithm. The following describes the events that occur :

1. Nodes n_{13} , n_{33} and n_{12} in clusters 1 and 3 make critical section requests with messages $req_1(13, 1)$, $req_2(33, 2)$ and $req_3(12, 3)$ respectively where the first parameter is the identity and the second is the timestamp of the request.
2. The coordinator for Cluster 1, C_1 , forms two request messages R_{12} and R_{13} for each request and sends these to the next coordinator on the ring, C_2
3. C_2 passes these messages immediately to its successor C_3 as it has no pending requests in its cluster.
4. C_3 however has a pending request from n_{33} and checks the timestamps of the incoming requests with the timestamp of the request in its cluster. C_3 replies to R_{13} as it has a lower timestamp than n_{33} request by simply filling the acknowledgement field in the message but defers the reply to R_{12} as it has a greater timestamp.
5. C_1 receives the reply message for R_{13} and therefore sends a *Reply* message to n_{13} which enters its critical section.
6. C_3 has a similar request (R_{33}) in the ring which is blocked by C_1 because n_{13} has a lower timestamp.
7. When n_{13} finishes execution of its critical section, it sends a *Release* (rel_1) message to C_1 which in turn passes the blocked *Reply* message for n_{33} to C_2
8. C_3 now has the *Reply* message and n_{33} executes similar to Step 5.
9. C_3 now releases the *Reply* message for n_{12} to execute its critical section.

The order of execution in this example is $n_{13} \rightarrow n_{33} \rightarrow n_{23}$ in the order of the timestamps of the requests.

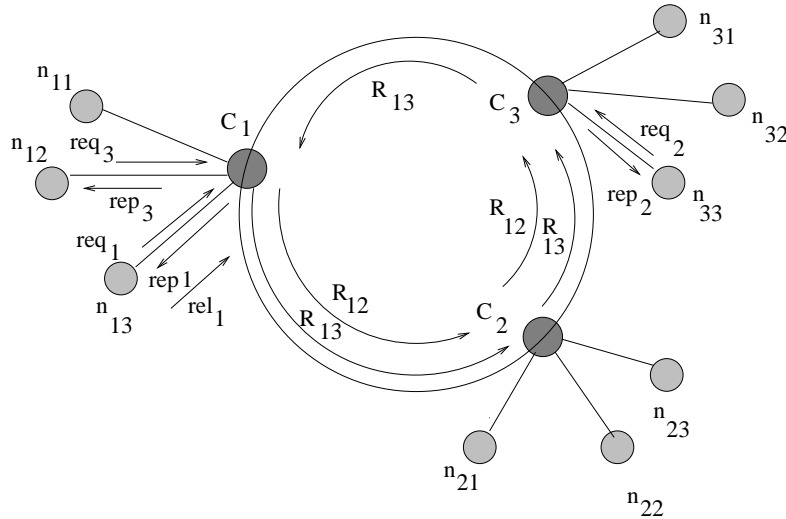


Fig. 3. Operation of the Ring_RA Algorithm

Theorem 1. *The total number of messages per critical section using the Ring_RA Algorithm is $k + 3$ where k is the number of coordinators (clusters).*

Proof. An ordinary node in a cluster requires three messages (*Request*, *Reply* and *Release*) per critical section to communicate with the coordinator. The full circulation of the coordinator request (*Coord_Req*) requires k messages resulting in $k + 3$ messages in total.

Corollary 1. *The Synchronization Delay (S) in the Ring_RA Algorithm varies from $2T$ to $(k + 1)T$ where k is the number of clusters.*

Proof. When the waiting and the executing nodes are in the same cluster, the required messages between the node leaving its critical section and the node entering are the *Release* from the leaving node and *Reply* from the coordinator resulting in two message times ($2T$) for S_{min} . However, if the nodes are in different clusters, the *Release* message has to reach the local coordinator, circulate the ring through $k - 1$ nodes to reach the originating cluster coordinator in the worst case and a *Reply* message from the coordinator is sent to the waiting nodes resulting in $S_{max} = (k + 1)T$.

Corollary 2. *In the Ring_RA Algorithm, the response times are $R_{light} = (k + 3)T + E$ and R_{heavy} varies from $w(2T + E)$ to $w((k + 1)T + E)$ where k is the number of clusters and w is the number of pending requests.*

Proof. According to Theorem 3, the total number of messages required to enter a critical section is $k + 3$. If there are no other requests, the response time for a node will be $R_{light} = (k + 3)T + E$ including the execution time (E) of the critical section. If there are w pending requests at the time of the request, the minimum value R_{heavy_min} is $w(2T + E)$. In the case of S_{max} described in Corollary 1, R_{heavy_max} becomes $w((k + 1)T + E)$ since in general $R_{heavy} = w(S + E)$.

Since the sending and receiving ends of the algorithm are the same as of RA algorithm, the safety, prevention of starvation and the fairness attributes are the same. The performance metrics for the Ring_RA Algorithm are given in Table 4.

Table 4. Performance of the Ring_RA Algorithm

M_{light}	M_{heavy}	R_{light}	R_{heavy_min}	R_{heavy_max}	S_{min}	S_{max}
$k + 1$	$k + 1$	$(k + 3)T + E$	$w(2T + E)$	$w((k + 1)T + E)$	$2T$	$(k + 1)T$

4 The Token Passing Mutual Exclusion Algorithm on the Ring

We propose a practical implementation of TP Algorithm to be executed using the architecture described in Section 3. In this algorithm, the token is circulated

in the ring only. The coordinator for each cluster determines whether to consume the token or not, based on its state. The FSM diagram of the TP Coordinator is depicted in Fig.4. When a node wants to enter a critical section, it sends a request message to the coordinator which records this event and changes its state to WAITTK to wait for the token. Once it has the token from the ring, it sends this token to the node by *Coord_Tok* to grant the request of the node and changes its state to WAITND to wait for the node to finish its critical section. A finished node will then send the token *Node_Tok* to the coordinator which will then release the token for circulation in the ring or send it to another waiting node in its cluster as shown in Fig. 4.

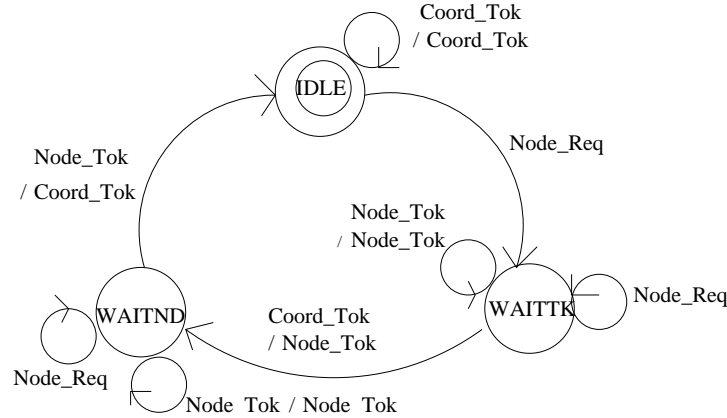


Fig. 4. The Coordinator for the Ring-Token Algorithm

Theorem 2. *The Ring-TP Algorithm has a messages complexity of $O(k + 1)$ per critical section.*

Proof. The proof is similar to the proof of Theorem 1 except that $k + 3$ is an upper bound on the number of messages depending on the current location of the token when a request is made.

The Synchronization Delay (S) and the Response Time values are similar to the Ring-RA Algorithm as shown in Table 5.

Table 5. Performance of the Ring-TP Algorithm

M_{light}	M_{heavy}	R_{light}	$R_{heavy-min}$	$R_{heavy-max}$	S_{min}	S_{max}
$O(k + 3)$	$O(k + 3)$	$(k + 3)T + E$	$w(2T + E)$	$w((k + 1)T + E)$	$2T$	$(k + 1)T$

5 Discussions and Conclusions

We proposed an architecture to implement distributed mutual exclusion algorithms. We showed that this architecture provides improvements over message complexities of Ricart and Agrawala and Token-based algorithms and also the time required to execute a critical section.

A comparison of the two algorithms with their regular counterparts in terms of their message complexities is shown in Table 6. Here we see that it is possible to obtain an order of magnitude of improvement over the classical RA and the Token-Based algorithms using our model at the expense of large response times and increased synchronization delays. For large k values, the gains with respect to normal algorithms are shown in the last column of the table as $O(2m)$ and $O(m)$. This may be interpreted as the more nodes a cluster has, the less number of messages required to enter a critical section with respect to the regular algorithms. However, large k and m values would result in a coordinator becoming a bottleneck as in the central coordinator case and the response time would be large. The coordinators have an important role and they may fail. New coordinators may be elected and any failed node member can be excluded from the cluster which is an improvement over both classical algorithms as they do not provide recovery for a crashed node in general. The recovery procedures can be implemented using algorithms as in [2] [11] which is not discussed here. One other advantage of the proposed model is that the pre-processing of the requests of the nodes by the coordinators are performed independently resulting in improved performance.

Table 6. Comparison of the Ring Mutual Exclusion Algorithms with others

	Regular	Ring Algorithms	Ring (k=m)	Gain(large k,m)
Ricart-Agrawala Alg.	$2(N - 1)$	$k + 3$	$O(\sqrt{N})$	$O(2m)$
Token Passing Alg.	N	$O(k + 3)$	$O(\sqrt{N})$	$O(m)$

Our work is ongoing and currently we are investigating the implementation of Susuki and Kasami algorithm [10] and Maekawa's algorithm [5] on this architecture. We are also looking into implementing k-way distributed mutual exclusion where there may be k nodes executing a critical section at one time. One other direction of study we are pursuing is the implementation of this model in mobile ad hoc networks. The mobile network can be represented as a graph which can be partitioned into a number of clusters periodically or upon change of configuration, using suitable heuristics. Once the partitioning is completed, each cluster can be represented by a coordinator and the model proposed will be valid to provide distributed mutual exclusion in mobile networks.

References

1. Chang, Y.I.: A Simulation Study on Distributed Mutual Exclusion. *Journal of Parallel and Distributed Computing*, Vol. 33(2). (1996) 107-121
2. Erciyes, K.: Implementation of A Scalable Ring Protocol for Fault Tolerance in Distributed Real-Time Systems. *Proc. of Computer Networks Symposium BAS 2001*. (2001) 188-197
3. Lamport, L.: Time, Clocks and the Ordering of Events in a Distributed System. *CACM*, Vol. 21. (1978) 558-565
4. Maekawa, M.: A \sqrt{n} Algorithm for Mutual exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, Vol. 3(2). (1985) 145-159
5. Raymond, K.: A tree-based Algorithm for Distributed Mutual Exclusion. *ACM Trans. Comput. Systems*, Vol. 7(1). (1989) 61-77
6. Ricart, G., Agrawala, A.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. *CACM*, Vol. 24(1). (1981) 9-17
7. Shu, Wu, An Efficient Distributed Token-based Mutual Exclusion Algorithm with a Central Coordinator, *Journal of Parallel and Distributed Processing*. Vol. 62-10. (2002) 1602-1613
8. Singhal, M.: A Taxonomy of Distributed Mutual Exclusion. *Journal of Parallel and Distributed Computing*, Vol. 18(1). (1993) 94-101
9. Singhal, M.: A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed System. *IEEE Trans. Parallel and Distributed Systems*, Vol. 3(1). (1993) 94-101
10. Susuki, I., Kasami, T.: A Distributed Mutual Exclusion Algorithm. *ACM Trans. Computer Systems*, Vol. 3(4). (1985) 344-349
11. Tunali, T, Erciyes, K., Soysert, Z.: A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System. *Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems*, Vol. 2(1). (2001) 47-62