International Journal of Advanced Research in Engineering and Applied Sciences

ISSN: 2278-6252
Impact Factor: 7.687

A PERMISSION BASED DISTRIBUTED GROUP MUTUAL EXCLUSION ALGORITHM HAVING QUALITY OF SERVICE

DR. PAWAN K THAKUR[1] AND VIVEK CHAUDHARY[2]

1. **Associate Professor, Department of Computer science and Engineering, Govt. College Dharamshala, H.P. (India) pawansarkaghat@gmail.com**
2. **Research Scholar , Career Point University, Kota , Rajasthan; viveksalil@gmail.com**

ABSTRACT

In large distributed systems which are based on cloud computing, the resources are shared to the clients. In these systems , there must be some service level agreement between the clients and the provider of the service. In Quality of Service, some constraints such as priority, response time and fault tolerance must be taken into consideration.

In this paper , we present a permission based group mutual distributed algorithm. The group mutual distributed algorithm is a generalization of mutual exclusion problem. In group mutual exclusion algorithm, processes in the same group can enter the critical section simultaneously.

Keywords: Cloud computing, Quality of Service, Group mutual exclusion.

1.0 Introduction

Cloud computing model provide on-demand network access to shared resources[4]. The services of cloud computing are hosted on a series of virtual machines running over the physical machines. The property of cloud elasticity must be fulfilled. The cloud elasticity is the ability to provide the cloud resources to different processes dynamically. In large distributed systems which are based on cloud computing, the resources are shared to the clients. There must be some Service Level Agreement between the clients and service provider. In SLA, the QoS[10] is the main issue. The changing needs of the cloud computing must be taken into consideration. The new properties of cloud computing such as scalability, fault tolerance and QoS must be considered while developing the new algorithm. Since applications in cloud computing share resources, concurrent access to them might be critical in order to avoid inconsistencies. These resources are considered as critical section and

concurrent access to these resources must be done carefully and some mutual exclusion algorithm must be there.

The three basic approaches for implementing distributed mutual exclusion are used. These are: Token based approach[2][8][16], Non token based approach[6][7][14][20] and Quorum based approach[5][17][18].

In token based approach, a logical token representing the access right to the shared resource is passed in a regulated manner among the sites. The site who is having the token is allowed to enter the critical section. Mutual exclusion is ensured because the token is unique. The algorithms based on this approach have to search the token .These types of algorithms provide better message complexity and easy to extend but the loss of token is the bottleneck .

In non token based approach , each site freely and equally competes for the right to use the shared resource. The message are used among the sites to determine which site will enter the critical section. A site enters the critical section when an assertion, defined on its local variables become true. The assertion becomes true only at one site at a given time and it ensures the mutual exclusion. These type of Algorithms are fault tolerant but at the cost of increased message complexity.

In quorum based approach, each site request permission to execute the critical section from a subset of sites. This set of sites is called quorum. Any two quorums contains a common site. This common site is responsible to make sure that only one request executes the critical section at any time. These type of Algorithm have lesser message complexity because they have to take permission from the subset and not from all the processes in the system but the problem of creating and initialization of quorum is there.

The most of the current mutual exclusion algorithms are not suitable for cloud computing because they do not consider cloud characteristics such as Quality of Service and fault tolerance. There are some algorithms which provide QoS [4][9][11][12]. In this paper we will present a group mutual algorithm which consider QoS based on Service Level agreement. The QoS requirement is important for any application because they are recorded between customers and the system designers. Any violation from QoS can lead to customer dissatisfaction. The QoS defined in our algorithm is priority , waiting time and response time.

2.0 Related Work:

The problem of GME was firstly given by Yuh-Jeer Joung[25]. Joung proposed two different algorithm for GME. These are Joung's broadcast based algorithm[23] and Joung's quorum based algorithm[24]. Joung's broadcast based algorithm was an extension of Ricart and Agarwala distributed mutual exclusion algorithm[20]. Joung proposed two algorithms RA1 and RA2. In RA1, the process which wants to enter the critical section , sends a request message to all the processes and upon receiving reply message from all the processes, it enters the critical section. There are some concurrency related issues in RA1, which was later solved by using RA2. In Joung's quorum based algorithm , the concept of quorum is used. A process has to obtain permission from all the processes in the quorum to enter critical section. For concurrency, Joung proposed two algorithms , the first one is Maekawa_M, which sends message in parallel and second one is serial version called Maekawa_S, which obtains sequential permission from each process in quorum. These two algorithms avoids deadlock .

In comparison to classical distributed systems , the working in cloud computing is different because it deals with different characteristics . The different characteristics in cloud computing includes fault-tolerance, QoS, scalability and priority. There are different priority based algorithms which are used for real time systems. These can be categorized as :

(i)Static priority algorithms (ii) Dynamic priority algorithms.

The priority in static priority algorithms remains the same. There is no priority inversions but it can lead to starvation as low priority processes cannot be able to enter the critical section. Housni and trehel[8] proposed an algorithm where sites with same priority forms the group. It uses router for external communication and the processes within the group communicate with each other by passing messages. When any process wants to enter the critical section, it sends the request and that request is forwarded to the root. The root sends the token request to the routers. In each group , the Raymond algorithm[19] is used.

In dynamic priority algorithms, the priority of algorithm is increased with the passage of time. For increasing the priorities , different factors such as request time, level and distance are used in different algorithms. In Kanrar-Chaki[10] token based algorithm , which is based

on Raymond algorithm[19], the low priorities of pending requests are increased dynamically. In avoids starvation but increases priority inversions. Jonathan Lejeunl et al[12] proposed a token based algorithm where new concepts have been added in Kanrar-Chaki[10] token based algorithm. These are level heuristics and level distance heuristics. Level heuristics postpones the priority increment of pending requests. In level distance heuristics , the processes are incremented according to the level of the tree. These two heuristics removes the drawbacks of the Kanrar-Chaki[10] token based algorithm where the low priority processes frequently access the critical section which is priority inversion. In priority inversions, a low priority process has been granted the access to critical section before the high priority process which is violation of Service Level Agreement. Jonathan Lejeunl et al[13] proposed a new algorithm where the attempt is balance the priority inversions and response time of low priority processes. It uses the awareness concept which aims at reducing maximum response time whereas the number of priority inversions remains low. For this global view of pending requests is necessary.

3.0 System Model:

The distributed system consists of set of n processes and a set of communication channels. The distributed system is asynchronous and does not have global clock. Information is exchanged between different processes by passing message asynchronously. We have assumed that message delay is finite and processes are non faulty and channels are reliable.

3.1 Group mutual exclusion problem:

The problem of GME was firstly given by Yuh-Jeer Joung[18]. In GME problem the processes which are competing for the critical section , must be placed in the request queue. From the request queue the group will be assigned to the process after considering the different factors such as waiting time, execution time , priority , age and group size. If a process has been granted the privilege message of a particular type, then this process will grant the requests of different processes of the same type. If there are n processes of the same type, then n processes can enter the critical section.

Group mutual exclusion satisfies the following properties:

Safety: This property states that there will be only one privileged message in the system. At any time , the number of privileged message should not exceed more

than one which further states that if the two processes are of different group, then they cannot enter the critical section simultaneously.

Liveliness: This property ensures that every process gets a chance to enter the critical section and it avoids unnecessary blocking and starvation.

Concurrent entry: If the processes belongs to the same type, then they can enter the critical section concurrently.

3.2 Performance metrices:

Message Complexity: It is the number of messages required to enter the critical section by any process.

Concurrency: it is the number of processes which are in the critical section at a given time.

Synchronization delay: It is the time when process from the current session exits from the critical session and next process from the different session enters the critical section.

4.0 A permission based distributed group mutual exclusion algorithm having Quality of Service:

In this section , we will present a permission based distributed group mutual exclusion algorithm having QoS considering the different factors such as waiting time, priority and execution time. Our algorithm solves the critical section problem for distributed group mutual exclusion and also provide QoS.

4.1 Outline:

It is a permission based distributed group mutual exclusion algorithm in which different processes communicate by message passing. The root node has all the authority and it maintain information regarding all the processes. The root node maintains the global view of all the processes. When child node requests for the critical section , its value Vi(t) is calculated . This value is based on waiting time, priority and execution time of the process. This request is forwarded to the parent node along with Vi(t) value and from parent node to the root node. The parent node maintains all the information about the child nodes in its local queue. Now at parent node, it is checked whether it is having the privilege message or not. If the parent node is having the privilege message , it means that this parent node is

having the authority to grant the requests of a particular type of its child nodes. If the parent node is not having privilege message or the type of the resource is different , the request is forwarded to root node. The root node maintains all such information in its global queue.

The root node maintains a pool of different requests. At the root node , the requests for critical section of different processes is counted type wise. Then the group will be created based on number of counts of same types. Now the priority of the different groups are calculated and these groups of different types are sorted in ascending order. It is explained in the section “Sorting the global queue”.

The first process in the first group now holds the privilege message. It will act as parent node and all other nodes will be its child node. The parent node and its child nodes can now enter the critical section i.e. they can be allowed to use the sharable resources concurrently.

If some high priority process requests for the critical section of different type, the root node calculates the total priority of that group and checks whether the priority exceeds the priority of currently executing group. If it is the case then the interrupt message is sent to the node which is having the privilege message. On receiving the Interrupt message, the parent node which is having the privilege message will send the Interrupt_ack message after its child node finishes critical section execution. The root in response to the Interrupt_ack message will send the privilege message to the first process of different type.

4.2 Sorting the global queue:

In various algorithms , queues are sorted by using FIFO policy. But FIFO policy does not consider different factors such as waiting time, priority and execution time. These parameters are important for the applications. We will use the formula which allow us to insert a new process in request queue. Suppose if Si is requesting the critical section , then its request will be forwarded to the Sp. If Sp is not having the privilege message, then Si request will be forwarded to Sr through Sp including its Vi(t) value. Sr will maintain information about all requests. Here the number of requests of different type will be counted. Based on largest to smallest count, the groups will be formed. Now the total priority of groups will be calculated. Suppose if Si is having priority Pi, then its value Vi(t) is calculated as:

Vi(t)= $\frac{t-ED}{RWT-t}$*Pi*$\frac{1}{ET}$

Where RWT= LT-(ET+Il+e)

ED= hi-1

pi=Priority of initiator site of request

hi=Initial Lamport stamp

ED= Emission date of request

RWT=Rest of request waiting time

e= treatment time of request at a site.

Il= Intermediate links

Accordingly ,all the values of processes which belongs to the same group will be calculated and finally the total value is calculated. Now the different groups will be sorted based on this value.

4.3 Description of the Algorithm:

In this algorithm , five types of messages are used:

Request: When a process send a request to enter into critical section.

Reply: When some process sends reply message to another process.

Exit: When the process releases the critical section.

Privilege: The node which have the authority to grant access to critical section.

Interrupt: When high priority process sends the request and total value of requests exceeds the current value.

Different cases of the algorithm is discussed.

Case1: When a site Si(Child node) wants to enter the critical section , it defines its expiration time and execution time in critical section. It then calculates the value Vi(t). Then the request of Si along with its Vi(t) value is send to the parent node(Sp). At Sp it is checked whether waiting time has reached or not. If waiting time is reached then this message will be deleted from the local queue. From the parent node , it is forwarded to root node Sr. The root node maintains a pool of different requests. At the root node , the requests for critical section of different processes is counted type wise. Then the group will be created based on number of counts of different types. Now the value Vi(t) of the different groups are calculated and these groups of different types are sorted in descending order. If Sp

belongs to the first group, which is having the highest value and if it is the first process in that group, then a privilege message is send to the Sp. Sp which is the parent node and all other processes become the child node of Sp. All these processes can enter the critical section concurrently. The tree is rearranged dynamically according to these changing values.

Case 2: If some other process Sj of Sp sends a request of same type to enter the critical section ,then its request is forwarded to the Sp. Since Sp is having the privilege message, it has the authority to grant the access to Sj directly.

Case 3: If some process Sk sends the request with higher priority and its type is different , then its request will be forwarded to the root node. Root node now calculates the total value of all the processes waiting in the global queue . If this value exceeds the current value , then interrupt message will be send to the Sp. Sp now send the Interrupt_ack message to Sr once the processes which have entered the critical section, finished their execution. Now the reply will be send to Sk, and all the processes in the group including Sk will enter the critical section.

Case 4: If all the requests of local queue Sp are fulfilled , the Sp will send a exit message and delete the messages in the local queue. It also returns the privilege message to Sr. On encountering the exit message , the Sr will remove all the information of that group in the global queue. Now if second group in the global queue will have the maximum value, then all the processes of that group can enter into the critical section.

4.4 Pseudo code of the algorithm:

Variables:

pi=Priority of initiator site of request

hi=Initial Lamport stamp

ED= Emission date of request

RWT=Rest of request waiting time

e= treatment time of request at a site.

Il= Intermediate links

S= Maximum number of processes of particular type in Sr

Msg=Message of particular type.

Type= Request of particular type of resource

ISSN: 2278-6252
Impact Factor: 7.687

Si=Request site

Sc= Current site

Sp=Parent site of current site

Sr= Root site

Lt= Latest timestamp of CS exit

Et= CS execution time

Local_queue= Local queue at parent node

Global queue= Global queue at the root node

Message having the following attributes:

Identification

Vi(Value of Si at a particular site)

Initialization

Procedure init()

Flag=0

Granted=0

Msg=nil

Local_queue=Ø

Global_queue=Ø

Procedure Request()

for(i=0;i<=n;i++)

Count the number of processes of different types.

Form the groups of similar types.

Calculate priority of groups.

Sort the groups in ascending order.

If(Si==p1) and (Msg_i==request)

Send privilege message to p1

```
Else If(Si==g1)and (Si!=root)
        Send message to Sp
        Insert message in local_queue
Else if (typei!=x)
        Send message to Sr
        Insert message in global_queue
Else if (msgi==Interrupt)
        Call Interrupt()
Else
        Reject message
```

Procedure exit()

```
for(i=0;i<=n;i++)
        flag=o
        granted=0
        msgi=exit
        do
                (IdSi==Idmessagei)and(Typei==x)
                        Remove message from local_queue
        While(IdSi==IdMessagei)
        If(Sc==root)
                Send message to Sr
        Else
                Send message to Sc
```

Procedure Process_message()

```
for(i=0;i<=n;i++)
while(flag==0)
        receive message from Si
        if(msgi==request) and(typei==x) and(wt>t)
                if(Sc==root)
```

ISSN: 2278-6252
Impact Factor: 7.687

```
                insert message in global_queue
        else if (type_i!=x)
                send message to Sr
        else if(msg==Interrupt)
                call Interrupt()
        else
                delete message
if(msg_i==reply) and (Type_i==x) and (Sp==p1)
        If(Sc==Si)
                Flag=1
                Granted=1
                Enter in CS
If(msg_i==Interrupt) and(Type_i!=x) and(Sp==p1)
        If(Sc==Sk)
                Flag=1
                Granted=1
                Enter in CS
If(msg_i==exit)
        Granted=0
        Do
         If(t>RWT) or (IdS_i==IdMessage_k)
                Remove message from local_queuep
        While(IdS_i==IdMessage_k)
        If(Sc==root)
                Granted=1
                Send reply message to Sr
        Else
                Send exit message to Sp
```

Procedure Interrupt()

If(S_k==request) and (V_k>V_i)

Calculate priority of groups

If(priority(g_i)<priority(g_k)

Send interrupt message to Sp of g1

If(msg==Interrupt)

Msgi=exit

Send Interrupt_ack toS_k

Send reply message to S_k

Call Process_message()

5.0 Theoretical analysis of the algorithm:

This algorithm satisfies safety, liveliness and concurrency properties. Also this algorithm is starvation free.

Proof of safety:

Assertion: If two processes P_i and P_j are executing the critical section concurrently, then the session must belong to same type.

Proof: In this algorithm , it is mentioned that only one process will hold the privilege message. The child of the node which is holding the privilege message will enter the critical section concurrently. Suppose that if Pi is having the privilege message and Pj requests the critical section, then type of Pj must be same as that of Pi and it must belong to same group as Pi, then only Pi will be allowed to enter the critical section. If Pj request type is different, then it's request must be forwarded to the root node and at the root node it is decided whether that the group which Pj belongs will be allowed to enter the critical section or not.

It proves the safety property.

Liveliness:

Assertion: Every process gets a chance to enter the critical section and it avoids unnecessary blocking.

In our algorithm, processes of a particular type forms the group. The group will be sorted in ascending order according to the values calculated in the groups. The value Vi which is calculated is based on different factors such as waiting time , priority and execution time. Whenever the total value of another group exceeds the current value, the Interrupt message will be raised. Also with the passage of time, the value Vi increases. It allows the low priority processes to enter the critical section.

It proves our liveliness property.

Deadlock does not occur in the system:

Assertion: The different nodes will be in deadlock state when there is no node in the critical section and all the requests for critical section cannot be fulfilled.

Proof: In our algorithm , if the parent node Sp has the privilege message, then it will send the exit message when all its children node(including itself) finishes executing the critical section . There is a mechanisms so that other processes can enter the critical section in a mutual exclusion manner. Also we have associated a waiting time with the processes and if the waiting time exceeds , then the messages will be deleted from the local queue. It ensures that processes will keep on executing the critical section.

Hence deadlock does not occurs in the system.

Starvation free:

Assertion: Assertion: Starvation occurs when one process must wait indefinitely to enter the critical section even when other processes are entering and exiting critical section. Starvation is impossible when every request in the critical section is fulfilled.

Proof: In our algorithm , the node does not have to wait for a indefinite time because with the passage of time , the value Vi(t) increases. If a process sends a request with low priority , then at some time it's value will become high thus having greater chances of entering into critical section.

Hence our algorithm is starvation free.

Group mutual exclusion is achieved:

Assertion: Different processes of different groups will be allowed to enter critical section in a group exclusion manner.

Proof: If two same type processes Pi and Pj requests for the critical section at the same time, then they will be allowed to enter the critical section. If some other processes of different type requests for the critical section and their group value exceed the current group value, then after sending the Interrupt message and getting the Interrupt_ack message , these processes can enter the critical section.

Concurrency:

Assertion**:** If two different processes P_i and P_j belongs to the same type, then they can enter the critical section concurrently.

Proof: From the algorithm, it is established that if two processes Pi and Pj requests the critical section and their type is same, they will be allowed to enter the critical section.

Performance analysis: In our algorithm, processes require log2(N) request message, log2(N)reply messages and log2(N) exit messages in the worst case. It also require one privilege message, one Interrupt message and one Interrupt_ack message. So the worst case complexity is 3log2(N)+3 messages. In our algorithm , the messages will be deleted or requests will be cancelled once the waiting time exceeds. It ensures the proper utilization of the bandwidth and give chance to higher priority processes to enter the critical section.

Message complexity:3log2(N)+3 messages

Synchronization delay: The main aim here is to determine the number of message between exiting of critical section and entering of other process into the critical section. At some time, a process must leave the critical section. When one group leave the critical section and sends the exit message, the other group will enter the critical section as soon as it receives the exit message. So the synchronization delay is one message hop.

Concurrency:

The maximum concurrency of this algorithm is n.

If n processes request the critical section and their type is same then they will be allowed to enter the critical section simultaneously.

6.0 Conclusion and future scope:

We have presented a group mutual exclusion algorithm allowing Quality of Service. The Quality of Service is essential in providing Service Level Agreement between the clients and service provider. It is one of the requirement in applications such as cloud computing. Our algorithm is based on group mutual exclusion concept . Here different processes under one group can access the critical section simultaneously. In this algorithm, the group priority is calculated which is based on different factors such as waiting time, execution time and priority of the processes. The requests of the processes whose waiting time is reached will be deleted from the local queue or the global queue. It saves the bandwidth which can be used for granting access to high priority processes. In this algorithm , some extra messages have been used i.e. Privilege, Interrupt and Interrupt_ack message. If some process with high priority request for the critical section , then it's request will be forwarded to the root node . At the root node, the group priority of the requesting process will be calculated. If it exceed the current value, then an Interrupt message is send to the currently executing group and after getting the Interrupt_ack message, the processes under that group can enter the critical section.

In this case we have not considered the case of fault tolerance. Also simulation can be conducted on this algorithm. These can be considered as future work.

References:

1. D. Agrawal and A. E. Abbadi, An efficient and fault-tolerant solution for distributed mutual exclusion, ACM Transactions on Computer Systems, 9(1), 1991, 1–20.
2. D.agarwal,A.El Abbadi,"A token baesd fault tolerant Distributed mutual exclusion algorithm,journal of parallel and distributed computing,24,pp.164-176,1995.
3. Dijkstra, E.W. Solution of a problem in concurrent programming control. Commun. ACM 1965, 8, 569.
4. Edmondson, J., Schmidt, D., & Gokhale, A. (2011). QoSenabled distributed mutual exclusion in public clouds. On the Move to Meaningful Internet Systems: OTM 2011, 542-559.

5. G. Cao and M. Singhal, "A Delay-Optimal Quorum-Based Mutual Exclusion Algorithm for Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, Vol 12, No. 12, pp. 1256-1268, Dec. 2001.
6. G. Ricart, A.K. Agrawala, An optimal algorithm for mutual exclusion in computer networks, Comm. ACM (CACM) 24 (1) (1981) 9–17.
7. Goscinski, A.M. (1990). Two Algorithms for Mutual Exclusion in Real-Time Distributed Computer Systems. J. Parallel Distrib. Comput., 9, 77-82.
8. Housni, A., & Trehel, M. (2001). Distributed mutual exclusion token-permission based by prioritized groups. In Computer Systems and Applications, ACS/IEEE International Conference on. 2001 (pp. 253-259). IEEE.
9. Ichiro Suzuki and Tadao Kasami," A distributed mutual exclusion algorithm", ACM transactions on Computer Systems Vol.3,no4,pp.344-349, nov.1985.
10. KANRAR, S., & CHAKI, N. (2010). FAPP: A New Fairness Algorithm for Priority Process Mutual Exclusion in Distributed Systems. JNW, 5(1), 11-18.
11. Lim, J., Chung, K. S., Chin, S. H., & Yu, H. C. (2012). A gossip-based mutual exclusion algorithm for cloud environments. Advances in Grid and Pervasive Computing, 31-45.
12. Lejeune, J., Arantes, L., Sopena, J., & Sens, P. (2012, May). Service level agreement for distributed mutual exclusion in cloud computing. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012) (pp. 180-187). IEEE Computer Society.
13. Lejeune, J., Arantes, L., Sopena, J., & Sens, P. (2013, October). A prioritized distributed mutual exclusion algorithm balancing priority inversions and response time. In Parallel Processing (ICPP), 2013 42nd International Conference on (pp. 290-299). IEEE.
14. M. Singhal, A taxonomy of distributed mutual exclusion, Journal of Parallel and Distributed Computing, 18(1), 1993, 94–101.
15. Neeraj Mittal and Mohan, " A priority based distributed group mutual exclusion algorithm when group access is non uniform", Journal and parallel Distributed computing Vol. 67,pp. 797-815,2007.

16. Peyman Neamatollahi, Hoda Taheri, Mahmoud Naghibzadeh, "A Distributed Token-based Scheme to Allocate Critical Resources", IEEE 2011
17. Ranganath Atreya and Neeraj Mittal, " A dynamic group mutual algorithm using surrogate quorum", in the procedding of 25th IEEE International conference on distributed computing system,2005 pp. 251-260
18. Ranganath Atreya, Neeraj Mittal, Member, IEEE Computer Society, and Sathya Peri, "A Quorum-Based Group Mutual Exclusion Algorithm for a Distributed System with Dynamic Group Set", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 18, NO. 10, OCTOBER 2007
19. Raymond, K. (1989). A Tree-Based Algorithm for Distributed Mutual Exclusion. ACM Trans. Comput. Syst., 7, 61-77.
20. Ricart, G., Agrawala, A.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. CACM, Vol. 24(1). (1981) 9–17
21. Sandeep Lodha and Ajay Kshemkalyani, A Fair Distributed Mutual Exclusion Algorithm, IEEE Transactions on Parallel and Distributed Systems, Volume 11 , Issue 6 (June 2000), Pages: 537 - 549.
22. Singhal, M.: A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed System. IEEE Trans. Parallel and Distributed Systems, Vol. 3(1). (1993) 94–101
23. Y.-J. Joung, ,"The congenial talking philosophers problem in computer networks", Distributed computing Vol.15,pp 155-175,2002.
24. Y.-J. Joung, " Quorum based algorithm for group mutual exclusion", IEEE transactions on parallel and distributed system, vol 14, no. 5 pp.2003,may2003
25. Y.-J. Joung, Asynchronous group mutual exclusion, Distributed Comput. (DC) 13 (4) (2000) 189–206.