A NEW ROBUST CENTRALIZED DMX ALGORITHM

Moharram Challenger
Islamic Azad University
Shabestar Branch, Iran
challenger@engineer.com

Vahid Khalilpour
Islamic Azad University
Shabestar Branch, Iran
khalilpor@iaushab.ac.ir

Peyman Bayat
Islamic Azad University of
Tafresh Branch, Iran
Bayat@autb.ac.ir

Mohammad Reza Meibodi
Amir-Kabir University,
Tehran - Iran
Meibodi@aku.ac.ir

ABSTRACT

In a distributed system, process synchronization is an important agenda. One of the major duties for process synchronization is mutual exclusion. This paper presents a new centralized fault tolerant distributed mutual exclusion algorithm based on Agrawala and El-Abbadi's algorithm. In new algorithm, once coordinator crashes, algorithm can recover lost data and return the coordinator in earlier situation. Thus fault tolerance will ascend and centralize algorithm's "single point of failure" will be omitted. So based algorithm will be more reliable. The only trade off is consuming some inappreciable time in case of coordinator's crash.

KEY WORDS

Distributed System, Mutual Exclusion, Fault Tolerance, Centralized Algorithm

1. Introduction

The mutual exclusion problem states that only a single process can be allowed access to a protected resource, also termed as a CS[*], at any time. Mutual exclusion is a form of synchronization and is one of the most fundamental paradigms in computing systems. To do mutual exclusion in single/multi processor systems, which has shared memory, there are various approaches like Semaphor, Monitor, TSL, Piterson and so on [17], [14], [15]. All mentioned solutions need to have a shared memory such that all involved processes can access common objects or variables. Considering problems and limitations like scalability in DSM[†] creation and relinquishing DSM, we can't employ former named ways to synchronize processes in distributed systems. Thus the new distributed-system-based approaches have been invented.

1.1. Background

Mutual exclusion has been widely studied in distributed systems, which has no shared memory, where processes communicate by asynchronous message passing, and a comprehensive survey is given in [1], [2]. For a system with N processes, competitive algorithms have a message complexity between log(N) and 3(N-1) messages per access to the CS, depending on their features.

Distributed mutual exclusion algorithms are either token-based or nontoken-based [16]. In token-based mutual exclusion algorithms, a unique token exists in the system and only the holder of the token can access the protected resource. Examples of token-based mutual exclusion algorithms are Suzuki-Kasami's algorithm [3] (N messages), Singhal's heuristic algorithm [4] ([N/2, N] messages), Raymond's tree-based algorithm [5] (log(N) messages), Yan et al.'s algorithm [6] (O(N) messages), and Naimi et al.'s algorithm [7] (O(log(N) messages). Nontoken-based mutual exclusion algorithms exchange messages to determine which process can access the CS next. Examples of nontoken-based mutual exclusion algorithms are Agrawala and El-Abbadi's algorithm [18] (3 messages), Lamport's algorithm [8] (3(N-1) messages), Ricart-Agrawala's algorithm [9] (2(N-1) messages), Carvalho-Roucairol's variant of the Ricart-Agrawala algorithm [10] ([0, 2(N-1)] messages), Maekawa's algorithm [11] ([3√N, 5√N] messages), and Singhal's dynamic information structure algorithm [12] ([N-1, 3(N-1)/2] messages). Sanders gave a theory of information structures to design mutual exclusion algorithms, where an information structure describes which processes maintain information about what other processes, and from which processes a process must request information before entering the CS [13].

1.2. Classification

Distributed mutual exclusion algorithms are divided into three groups [19], [20]:

1. Centralized Algorithms:

In this bunch of algorithms, a central process or machine services others. One of the well-known algorithms is Agrawala and El Abbadi [18], which has three steps to synchronize processes in entering and releasing CS. This central process is called coordinator and each process should be allowed from this coordinator to use a shared resource. At the end they should inform coordinator in time of leaving CS. This method is simple to implement but has centralization's own problems.

2. Distributed Algorithms [9], [24]:

In this series of algorithms, resource dedication and retrieval is performed in distributed manner. In other words all of the available processes decide about whom enters in CS [8], [11]. Of course, in this way there will be

[*] Critical Section
[†] Distributed Shared Memory

more communication rate and fault-tolerance can be low in the cases of crashing a process.

3. Token-based algorithms:

In this type of algorithms, a virtual ring has been made amongst clients and a token initialize to circle round it [21]. Each of the processes which is possessed the token, can enter CS and after releasing CS it should resume circling the token.

Compression for these three types of DMX algorithms in can be seen in table 1 [22] [23].

Table 1: Comparing Classic DMX algorithms

Algorithm	*Message per entry/release*	*Delay before entry*	*Problems*
Centralize algorithms	2	2	Coordinator crash
Distributed algorithms	2*(N-1)	2*(N-1)	Crash of any process
Token-base algorithms	1 to ∞	0 to N-1	Token lost or process down

Due to the absence of global time in a distributed system, timestamps are assigned to messages according to Lamport's clock [8]. In the context of mutual exclusion, Lamport's clocks are operated as follows: Each process maintains a scalar clock with an initial value of 0. Each time a process wants to access the CS, it assigns that request a timestamp which is one more than the value of the clock. The process sends the timestamped request to other processes to determine whether it can access the CS. Each time a process receives a timestamped request from another process seeking permission to access the CS, the process updates its clock to the maximum of its current value and the timestamp of the request.

Fault-tolerance is a very important criterion for solutions to most real-life resource contention problems. The commonly accepted definition of robustness in the context of mutual exclusion is that a crash on mutual exclusion algorithm does not lead to system crash. Of all the distributed mutual exclusion algorithms in the literature, Agrawala and El-Abbadi's algorithm needs less communication messages per CS exchange, but it is a centralized approach and therefore it may become a bottleneck and single point of failure. These weaknesses can leads to system crash. In this paper, a new complementary algorithm will be proposed which tend to have a fault-tolerance distributed mutual exclusion algorithm.

1.3. Motivation

Concerning different DMX algorithms and their attributes, token-base algorithms have more complexity to implement and are very sensitive to token lost and any process crashes that can result to system crash. Distributed algorithms doesn't have any bottleneck problem like centralized algorithms and decide in a distributed manner, but they increase message communication rate insensitively and according any process crash because of some information lost can lead to system failure or crash. But ultimately Centralized algorithms are simple to implement, have least message complexity and are fast, especially in relatively low number of clients unlikely other methods. The only problem in these kinds of algorithms is being centralized so having a single point of failure that results to less fault tolerance.

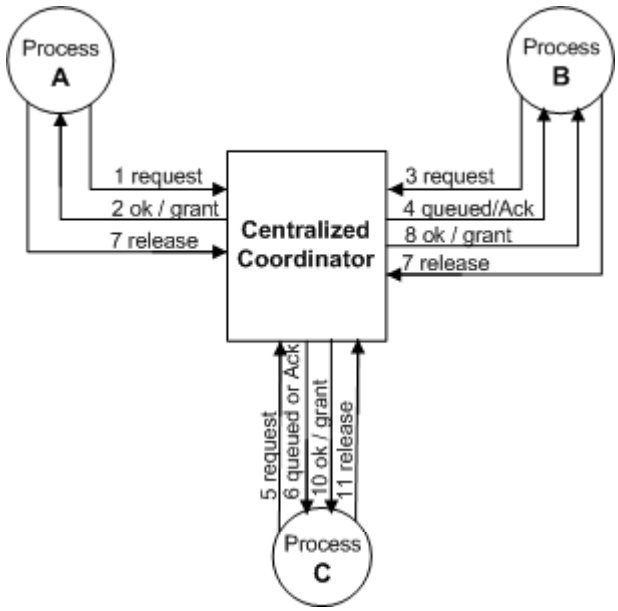

Fig. 1: Agrawala and El Abbadi's algorithm

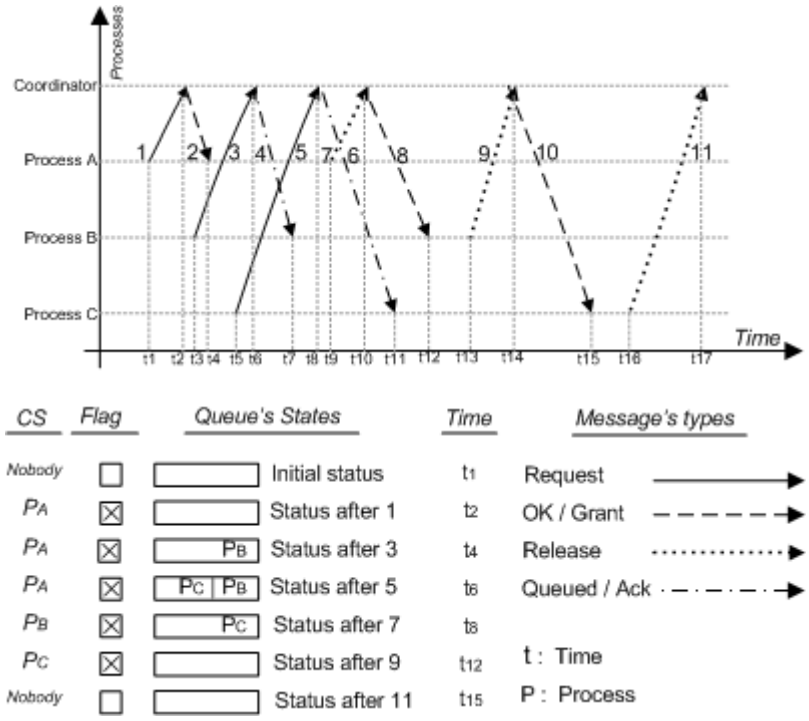

Fig. 2: Timing diagram for Agrawala and El Abbadi's algorithm

In this paper a complementary algorithm to centralized algorithm will be proposed which conduce fault-tolerance in the case of coordinator crash; so simply the bottleneck in centralize algorithm will be omitted. Thus centralized algorithm will be a reliable and applicable algorithm.
In rest of the paper, section 2 has more about Centralize algorithm itself as our major background. Section 3, speaks about powers, weaknesses, and special cases functionality of centralized DMX algorithm. In section 4, to cover some weaknesses centralized algorithm and to extend it we propose our complementary algorithm. Then, in chapter 5 we analyze algorithm in different special cases, also we try to exam various traditional scenarios. After that, in section 6 we proposed algorithm would be evaluated and compared with other algorithm using proves simulations, and analytical descriptions. Finally, in section 7 as a conclusion, we will illustrate our results plus future works.

2. Preliminaries

In this section, we describe the general system model and review the base centralized algorithm, which is a fair distributed mutual exclusion algorithm. The algorithm proposed in Section 3 is an improvement over this base algorithm.

2.1. System model

The base algorithm assumes the following model. There are N processes in the system. The processes communicate only by asynchronous message passing over an underlying communication network which is error-free and over which message transit times may vary. Processes are assumed to operate correctly. We assume FIFO channels in the communication network. Without loss of generality, we assume that a single process executes at a site or a node in the network system graph. Hence, the terms process, site, and node are interchangeably used.
A process requests a CS by sending a REQUEST message and waits for appropriate reply before entering its CS. While a process is waiting to enter its CS, it cannot make another request to enter another CS. Each REQUEST for CS access is assigned a priority and REQUESTs for CS access should be granted in order of decreasing priority for fair mutual exclusion.

2.2. Agrawala and El Abbadi's algorithm

In centralized algorithm, Agrawala and El Abbadi [18], there is a process that is elected to be coordinator and each of processes which is interested in entering critical section, asks a permission sending a request message.

Resource allocation:
In centralized algorithm, according to figure-1and figure-2, each process should send a "*Request*" message to coordinator while the flag related to resource in coordinator is reset (not set). Because the resource is not allocated to another process, coordinator sends back an "*OK*" message as reply to "R*equest*" message to resource-requesting process; so herewith it permit requesting process to enter critical section. Immediately coordinator sets related flag.

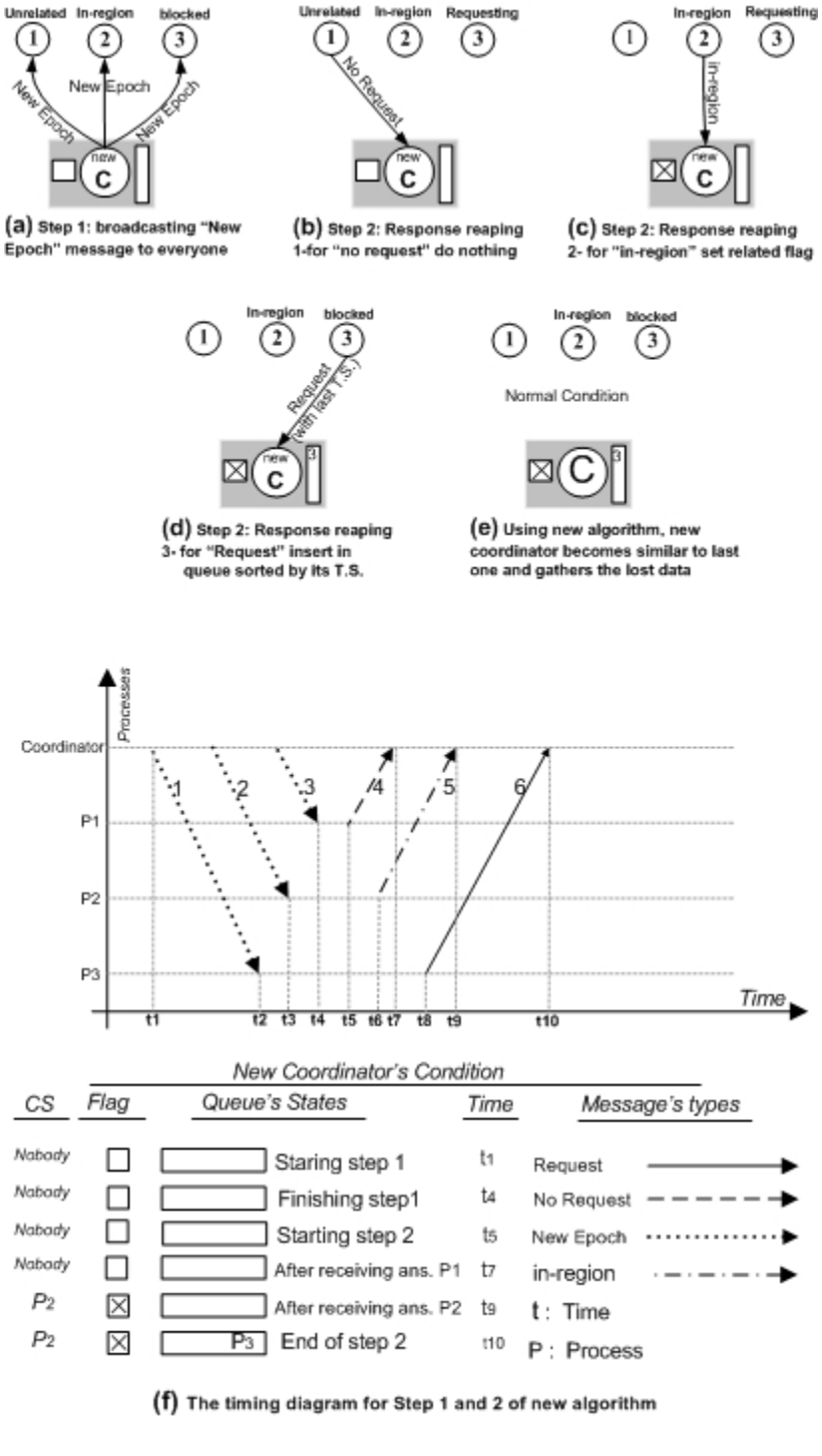

Fig. 3*:* Proposed algorithm's diagrams

CS releasing:
When process which granted CS, exits and wants to leave CS, sends a "*Release*" message to coordinator, so coordinator can manages the next requesting process to use that CS. Coordinator receiving "*Release*" message, removes next process waiting in the queue and send him a "*OK*" message. If there is not any process waiting in queue, reset the related flag. The routine can be find in figure 1 and figure 2.

Blocking a process:
According to figures 1 and 2, if after dedicating a resource to a process and before releasing the resource from that process, another process sends request for same resource, in this case coordinator can recognize that the shared resource is busy (checking its flag) and as result, to block it, coordinator do not send any response. In the result, new requesting process will be waiting till can get the "*OK*" message as permission and theretofore the process will be blocked. Of course distributed-system's reliability or coordinator's "*ACK*" message assures the new requesting process that message is received from coordinator side (recognizing dead coordinator from blocking state). Instead, coordinator puts the new unanswered request in the related resource's queue. At the end when in-region (in critical section) process exits the CS and send a "*Release*" message to coordinator, it remove our deferred process from the queue and sends an "*OK*" message to him. In result process get out of blocking mode and enters in CS.
The centralized algorithm is simple to implement and completely fair, but it has central part, coordinator, which makes system easy to fail. To solve this weakness, new algorithm as a complementary algorithm covers it.

2.3. Analyze of Base Algorithm

In centralized algorithm, a malfunction on communication will be covered in distributed system's operating system. That is because the distributed system is reliable and this means, if a message can not be transmitted, the system waits for a while and then retransmit it and after some repetition using a watch dog timer, system realizes that the destination has been crashed and reports it to the sender so sender makes a decision. In result, we transfer the responsibility of controlling this type of failure to distributed-system and its reliability feature.
If the coordinator crashes all the preserved data will be lost and this can lead to system crash. Due to this problem one of the disadvantages of classic centralized DMX algorithm takes form. That is coordinator's being "single point of failure". We have solved this problem in our new complementary algorithm.

3. Proposed Algorithm

New complementary algorithm starts when a coordinator crashes and a process for first time recognizes that event and then new coordinator is elected. At this moment new elected coordinator (every one can be), starts our algorithm to gather lost data and recover previous coordinator's safe mode.
By the way, new algorithm works well when for a first time distributed system starts to work and no previous coordinator was available before.
After crashing each process that has a request in queue, either the system gets reloaded or after a period there will be an election and the new system will be established. Now the suggested algorithm starts the recovery in the following steps:

Step 1: Broadcasting "New Epoch" message
New coordinator sends a message called "*New Epoch*" to all of the processes in any workstation. Receiving this message, any process finds out that coordinator has been crashed and a new one has just been established. "*New Epoch*" message can have several other information about new coordinator. For example: new coordinator's machine address, new coordinator's DNS name, new coordinator's type (what does he coordinates- kind of resources that will be controlled) and even may be time stamp of election that he was elected as a coordinator (to compare, if it is necessary).

Step 2: Gathering all process replies
All the clients after receiving the "*New Epoch*"" message understand that there is a new system in the distributed system and each of them depending on its situation may send the related answers.
1. Those processes that neither have sent a request nor is in the critical section are called unrelated processes. These processes did not have any info in crashed coordinator so will not deal with new coordinator. Hence, they will send a message called "*No Request*" to coordinator and coordinator receiving this message understands that this process has no related work with new coordinator.
2. Processes, which are in the critical regions of resources that are controlled by new coordinator, send "In-Region" messages including information like, critical region that they are in, process name, process address and process ID. Coordinator after receiving all of the messages, using "*In-Region*" messages just sets each critical section's related flag in coordinator. Now, all of the flags are set exactly like the flags of previous crashed coordinator.
3. All of the processes that have sent a request message before receiving "*New Epoch*" and yet have not received "*OK*" message for entering CS, should resent a new request message including requested critical region name, sender process name, ID, address and finally the time-stamp of first request message which have been sent to previous coordinator.

Step 3: Organizing new coordinator
In this point, new coordinator should check that does not have any case as a "Reset flag and meanwhile not empty queue". This means that there is no process in critical region but there are some processes waiting to enter critical region.

```
// initializing process will be called at the coordinator's
// start time in coordinator   function

void Initialization ( void )
{
        int Flag = 0;
        int Queue [Process_No]= {0};
        Recovery ( );
}  //end of Initialization function

// Recovery function, gathers last coordinator's lost data to rebuild
// new coordinator accurately, according to new complementary algorithm
void Recovery ( void  )
{
    broadcast ("New Epoch");//sends the message to all of the available processes
    while (receives answer from all of processes)
    {
        M= get_message ( );     // gathering answers one by one
         switch (M.message)       // decide over received messages answers
         {
                case "No Request":
                        do_NoP( );     // Do No operation
                        break;
                case "in-region" :
                        Flag = M.process_No;      // seting a CS flag
                        Break;
                 case "Request" :
                        insert (Queue, M.process_No, M.time_stamp);
                        break;
                 default :
                          //we assume other messages (like "Released" message)
                            // as "No Request" message till the end of new
                            // algorithm's recovery procedure
                        do_NoP ( );   // Do No Operation
                        break;
         } // end of switch
    }  // end of while

    if   ( Flag ==0   &&   !IsEmpty (Queue)  )
    {
          i= delete (Queue);
          Flag = i;
          send (i, "OK");
    }
}  // end of Recovery function
```

Fig. 4: Proposed algorithm

These steps are shown in figure 3 and related algorithm is shown in figure 4.

4. Evaluation

In this section we will give formal reasons to show the correctness of algorithm.

Correctness Proof

In this subsection, it will be proved that the new algorithm makes DMX algorithm more fault-tolerant. Therefore, it can be inferred that the new algorithm works accurately to make a robust DMX algorithm.

To do so, it should be proved that the set of data before current coordinator crash in old coordinator and after that, the data in new coordinator, are correspondent. First, we will survey the correspondence of old coordinator's data with the systems' CS-related data. Second, The equality of systems' CS-related data with new coordinator's data will be surveyed (according to new algorithm's routine). Finally, considering the crash time, the fault tolerance of coordinator will be gain.

We consider the system processes as set S; the old coordinator's data as set C and the new coordinator's data as C. However, the set S includes three subset of clients' data (clients in three states: in-region, requesting and unrelated).

As data structure, each coordinator has a queue to store requests of processes and a flag, which saves the in-region process name[‡].

Then we have, $S=\{S1,S2,S3\}$

Such that, $S1 = \{Pi \mid Pi \text{ is in\_region, } P \text{ is a process}\}$

$S2 = \{Pj \mid Pj \text{ is requesting}\}$

$S3 = \{Pk \mid (Pk \notin S1) \,\&\, (Pk \notin S2)\}$

It is observable that set S is partitioned by S1, S2 and S3.

Therefore, $S3 = S - (S1 \cup S2) = S1' \cap S2'$

Also, $C = \{C1,C2,C3\}$

$C1 = \{Pi \mid Pi \in C\_flag, P \text{ is a process}\}$

$C2 = \{Pj \mid Pj \in C\_queue, P \text{ is a process}\}$

$C3 = S - (C1 \cup C2)$

Because the old coordinator was working accurately before its crash, the CS info in distributed system was in old coordinator. In the other words, in-region process's name is in coordinator's flag and each requesting process's name is in the coordinator's queue.

Therefore, we have, $S1 = C1$

$S2 = C2$

So, $S3 = S - (S1 \cup S2) = S - (C1 \cup C2) = C3$

Thus, all of the coordinator's info is correspondent with system's CS-related info.

$S=\{S1,S2,S3\}=\{C1,C2,C3\}=C$

$C' = \{C1',C2',C3'\}$

In the other hand, according to C' definition we have,

$C1' = \{Pi \mid Pi \in C'\_flag, P \text{ is a process}\}$

$C2' = \{Pj \mid Pj \in C'\_queue, P \text{ is a process}\}$

$C3' = S - (C1' \cup C2')$

The new coordinator has gathered its data using new algorithms steps. After sending a "New Epoch" message in step one of new algorithm, according to step 2.a in-region process sends an "In-region" message to new coordinator as an answer. And the new coordinator sets its flag with system's in-region process name. So, $C1' = S1$

In addition, according to new complementary algorithm step2.b those processes, which have sent a request to old coordinator and now are blocked, send request with previous time-stamp again (to new coordinator). Therefore, requesting or blocked processes in system will have a request in new coordinator's queue. So $C2' = S2$

Thus, $C3' = S - (C1' \cup C2') = S - (S1 \cup S2) = S3$

Therefore, $C' = \{C1',C2',C3'\} = \{S1,S2,S3\} = S$

Data Recovery

According to two main results above, we have:

$$C' = S \,\&\, S = C \rightarrow C = C'$$

In result, the data in old coordinator is correspondent with the data in new coordinator, which is gathered with new complementary algorithm. Therefore, the new algorithm can **recover** the lost data or system.

Fault Tolerance

With considering the crash time as T0, the old coordinator was working accurately at T1 exactly before crash, where $T1<T0$. In addition, reestablishment time of new coordinator is T2 (after running new algorithm leading to fully recovery), where $T0<T2$. Thus, we have $T1<T0<T2$. The data in old coordinator at T1 is set C and the data in new coordinator at T2 is set C'. As proved before, we have C=C'. Therefore, the system works correctly and continuously before and after crash. In result system is Fault-Tolerant.

5. Conclusion

One of the original goals of making distributed systems is to make systems more reliable than single-processor systems. The idea is that if a machine goes down, some other machine takes over the job. In other words, theoretically the overall system reliability could be the Boolean OR of the component reliabilities.

Of all the distributed mutual exclusion algorithms in the literature, the non-token based algorithms of Lamport [8] and Ricart-Agrawala [9], RA, are not reliable in the sense described above.

We presented a reliable mutual exclusion algorithm for distributed systems with asynchronous message passing. The saving in message complexity was obtained by exploiting the concurrency of requests and assigning multiple meanings to the requests and replies, whenever there are concurrent requests. However, this is also a drawback of Lamport's algorithm and the RA algorithm. The following improvements can be made to the algorithm. The first improvement saves the number of "REPLY" messages. A process Pi finishing CS (procedure FinCS) sends a "OK" to the concurrently

[‡]We consider that the coordinator and network platform are safe and sound. So they are not virus or something destructive.

requesting process with the next highest priority (if it exists). Also it sends "REPLY"s, to the processes whose "REQUEST"s were deferred. By examining these "REQUEST"s, Pi can determine the relative order in which these processes will execute CS. Using this fact, the following optimization can be made. Assume Pk has the highest priority among these "REQUEST"s. Pi can send "REPLY" just to Pk, apprising Pk of all the information Pi has gathered. Thus, Pi can avoid sending upto m (worst case is m-1) messages. Now it is Pk to take care of the rest. However, this optimization requires a significant increase in message sizes and local data structures. Second way to save the number of "REPLY" messages is by treating deferred "REQUEST" messages as concurrent to the next "REQUEST" of this process (although they are not truly concurrent by definition). If the process exiting the CS knows that it will request CS soon, it can keep deferred "REQUEST" as deferred until it makes its next "REQUEST". At that time, its "REQUEST" acts as a "REPLY" to the deferred "REQUEST", and the deferred "REQUEST" acts as a "REPLY" to its "REQUEST". This optimization could slow down the computation at processes.

6. Applications and Future works

This algorithm can be applied in distributed operating systems, as is discussed in all of the distributed operating system texts [20], and distributed programming cases. For example in java programming it maybe used to create the CS for common resources that are objects or classes. In these cases, we must define a class for managing the CSs. Other applications are: 3D animation [24], for example in a game net an object is shared between N players, so they are in race over the CSs to win. Finally Internet, which is a semi distributed system. In Internet, there are a lot of common resources and other conditions to create the mutual exclusion. However, we can use the presented algorithm in very sensitive or non-sensitive distributed systems.

Acknowledgement

Authors gratefully acknowledge the conference committee for their work on the original version of this document.

References

[1] Chang, Y.I, "A Simulation Study on Distributed Mutual Exclusion", Journal of Parallel and Distributed Computing ,vol. 33, pp. 107-121, 1996.

[2] Singhal, M. "A Taxonomy of Distributed Mutual Exclusion", J. Parallel and Distributed Computing, vol. 18, no. 1, pp. 94-101, May 1993.

[3] Suzuki, I. and Kasami, T., "A Distributed Mutual Exclusion Algorithm", ACM Trans. Computer Systems ,vol. 3 ,no. 4, pp. 344-349, Nov. 1985.

[4] Singhal, M. "A Heuristically Aided Algorithm for Mutual Exclusion in Distributed Systems", IEEE Transaction on Computers, vol. 38, no. 5, pp. 651-662, May 1989.

[5] Raymond, K. "A Tree-Based Algorithm for Distributed Mutual Exclusion", ACM Trans. Computer Systems ,vol. 7, no. 1, pp. 61-77, Feb. 1989.

[6] Yan, Y., Zhang, X. and Yang, H. "*A Fast Token-Chasing Mutual Exclusion Algorithm in Arbitrary Network Topologies*", J. Parallel and Distributed Computing, vol. 35, pp. 156-172, 1996.

[7] Naimi, N., Trehel, M. and Arnold, A. "*A log(n) Distributed Mutual Exclusion Algorithm Based on Path Reversal*", J. Parallel and Distributed Computing, vol. 34, pp. 1-13, 1996.

[8] Lamport, L. "*Time, Clocks and the Ordering of Events in Distributed Systems*", Comm. ACM ,vol. 21, no. 7 ,pp. 558-565, July 1978.

[9] Ricart, G. and Agrawala, A. K. "*An Optimal Algorithm for Mutual Exclusion in Computer Networks*", Comm. ACM ,vol. 24, no. 1, pp. 9-17, Jan. 1981.

[10] Carvalho O. and Roucairol, G. "*On Mutual Exclusion in Computer Networks, Technical Correspondence*", Comm. ACM, vol. 26, no. 2, pp. 146-147, Feb. 1983.

[11] Maekawa, M. "*A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems*", ACM Trans. Computer Systems, vol. 3, no. 2, pp. 145-159, May 1985.

[12] Singhal, M. "*A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed Systems*", IEEE Trans. Parallel and Distributed Systems, vol. 3, no. 1, pp. 121-125, Jan. 1992.

[13] Sanders, B. "*The Information Structure of Distributed Mutual Exclusion Algorithms*", ACM Trans. Computer Systems , vol. 5, no. 3, pp. 284-299, Aug. 1987.

[14] Yang, J. H. and Anderson, J. "*Time Bounds for Mutual Exclusion and Related Problems*", Proc. 26th Ann. ACM Symp. Theory of Computing, pp. 224-233, May 1994.

[15] Yang, H. J. and Anderson, J. "*A Fast Scalable Mutual Exclusion Algorithm*", Distributed Computing vol. 9, no.1, pp. 51-60, Aug. 1995.

[16] Lodha, S. and Kshemkalyani, A. "*A Fair Distributed Mutual Exclusion Algorithm*", IEEE trans. on parallel and distributed systems, vol. 11, no. 6, June 2000.

[17] Tanenbaum, A.S.: "*Operating Systems, Design and Implementation*", Prentice-Hall Int., Inc, 1997.

[18] Agrawal, D., and El Abbadi, A. "*An Efficient and Fault-Tolerant Solution of Distributed Mutual Exclusion*", ACM Trans. on Computer Systems, vol. 9, pp. 1-20, Feb. 1991.

[19] Tanenbaum, A.S. "*Distributed Operating Systems*", Prentice-Hall International, Inc, 1995.

[20] Tanenbaum, A.S., and Steen M.V. "*Distributed Systems Principles and Paradigms*", Prentice-Hall International, Inc, 2002.

[21] Maekawa, M., Oldehoeft, A.E., and Oldehoeft, R.R. "*Operating Systems Advanced Concepts*", Menlo Park, CA: Benjamin/Cumings, 1987.

[22] Michel, T., and Housni A. "*Comparison of Techniques used in Prioritized Mutual Exclusion by Groups*", Int. Conf. on Par. and Dist. Comput. PDCAT 2001.

[23] Mazzacano f., "*A Reliable Multicast Protocol for a Distributed System*", thesis, Boston College Computer Science Department, 2003.

[24] Bayat, P., Challenger, M., and Shiri, M. E.: "*A New Reliable Distributed Mutual Exclusion Algorithm*" Ninth IASTED Conf. on Software Engineering and Applications, AZ, USA, pp. 118-124.