A Solution to Improve Algorithm for Distributed Mutual Exclusion by Restricting Message Exchange in Quorums

Ousmane Thiare
Gaston Berger University
Department of Computer Science
UFR S.A.T
Saint-Louis. Senegal
othiare@dept-info.u-cergy.fr

ABSTRACT: *Resource management is one of the most important and fundamental problems in distributed systems. Typically, to maintain the integrity of a resource, at most one process should access the resource at any time. As a result, accesses to the same resource (that is, execution of critical sections) by different processes have to be serialized. This problem is referred to as the "mutual exclusion problem". In this paper, we have proposed a permission based distributed mutual exclusion algorithm which is an improvement of Maekawa's algorithm. The number of messages required by the improvised algorithm is in the range 3M to 5M per critical section invocation where M is the number of intersection nodes*[1] *in the system. A reduction in number of message by restricting the communication of any node with the intersection nodes of the quorums, without any modification of the basic structure of the algorithm.*

Keywords: Mutual exclusion algorithm, Resource management, Distributed systems

Received: 12 December 2009, Revised 12 January 2010, Accepted 20 January 2010

1. Introduction

Distributed Mutual Exclusion problem arises when concurrent access to protected resource(termed as Critical Section(CS)) by several sites is involved. In Distributed Mutual Exclusion, the requirement is to serialize the access to CS in the absence of shared memory which further complicates the problem.

Distributed Mutual Exclusion algorithms can be classified as token-based and non-token-based as suggested by [2], or as token-based and permission-based as suggested by [3]. In this paper, we propose a permission based Distributed Mutual Exclusion algorithm which is an improvement of Maekawa's algorithm [1].

Garcia-Molina and Barbara [4] first introduced the concept of coterie which could be mainly used to devise permission based Distributed Mutual Exclusion algorithms. A coterie consists of collection of sets of sites in the system and these sets are called quorums. In general, when a node wants to execute its CS it has to obtain permission from processes of any quorum in the coterie. Maekawa's algorithm [1] wast he first coterie-based algorithm where the nodes of the system are logically arranged into groups. Any node intending to execute its CS has to obtain permission from all the nodes in its respective group and these groups were created such that any two groups had atleast one node in common (referred to as intersection nodes) which act as arbitrators. In this paper, we further restrict the communication of the processes which want to execute its CS to their intersection nodes and achieve Distributed Mutual Exclusion in lesser number of messages.

Maekawa's algorithm[1] uses cK messages to create mutual exclusion in the distributed system, where as our proposed algorithm takes cM ($M < K$) messages per CS invocation where M, K and c are integers and $3 \leq c \leq 5$. However, our proposed algorithm preserves all the advantages of Maekawa's algorithm [1] and remains similar to it. The algorithm proposed in this paper is not fair, the synchronization delay is 2 and the algorithm is starvation free.

[1]The terms processes, sites and nodes will be used interchangeably throughout the paper

The problem of resolving conflicting access to resources also arises in replicated databases, where the emphasis is on resolving read and write conflicts effciently. Many methods [5] [6] [7] [8] [9] have been used to address this issue.

Document Structure. The rest of the paper is organized as follows. Section 2 reviews Maekawa's algorithm [1]. In section 3, we present our proposed algorithm. Section 4 present the proof and correctness. In section 5, we present the analysis of the proposed algorithm. Finally, we conclude in section 6.

2. Maekawa's algorithm

In this section, we present the computational model for the proposed algorithm and a review of Maekawa's algorithm.

2.1 The Computational Model

In this paper, we assume that a distributed system which is common to Maekawa's algorithm and to the proposed algorithm consists of N sites $(1, 2, 3 \cdots i \cdots, j \cdots, N)$. A distributed system is *asynchronous*, i.e., there is no common global clock. Information exchanged between processes is done by asynchronous message passing. Each communication channel is FIFO and each message sent is delivered within finite time, but there is no upper bound on message delivery time. We assume that the system is error-free.

The different types of messages used are *REQUEST, LOCKED, INQUIRY, FAILED, RELINQUISH* and *RELEASE*. Timestamps(TS) at any site i (where $1 \leq i \leq N$), TS_i are ordered par (L_i, i), containing the Lamport's logical clock [10] value L_i and the site id i.

An ordering "¡" on timestamps is defined as: $TS_i < TS_j$ iff $(L_i < L_j)$ or $(L_i = L_j$ and $i < j)$.

2.2 The algorithm

In Maekawa's algortihm, a site does not request permission from all the sites, but only from a subset of sites. The sites of the system is divided into groups called quorums (S_i, $1 \leq i \leq N$). The quorums are construted such as to satisfy the following conditions:

1. $\forall i \ \forall j, \ S_i \cap S_j \neq \theta, \ i \neq j, \ 1 \leq i, j \leq N$
2. $\forall i$, node $i \in S_i$, $1 \leq i \leq N$
3. $\forall i, \ |S_i| = K, \ 1 \leq i \leq N$
4. $\forall j$, node$j$ is within K S_i 's, $1 \leq i, j \leq N$

Condition (non null intersection property) is a necessary condition for the S_i 's so that mutual exclusion requests can be resolved. Condition 2 reduces the number of messages to be sent and received by a node. Condition 3 means that each node needs to send and receive the same number of messages to obtain mutual exclusion (equal work). Finally, Condition 4 signifies that each node serves as an arbitrator for the same number of nodes. This ensures that each nodeis equally responsible for mutual exclusion (equal responsability).

For exemple let us consider the finite projective plane of 7 elements, which consists of a subset such that every subset has exactly 3 elements, every element is contained in exactly 3 subsets, and every two subsets intersect in exactly one element. Processes are labelled as $1, 2, \cdots, 7$. The groups are:

S1 = {1, 2, 3},S2 = {2, 4, 6},S3 = {3, 5, 6}

S4 = {4, 1, 5},S5 = {5, 2, 7},S6 = {6, 1, 7}

S7 = {7, 4, 3}.

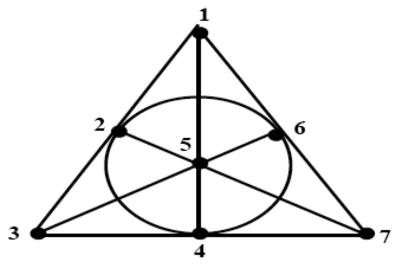

Figure 1. The finite projective plane of order 2 (Fano plane)

Maekawa established the following relationship between N and K: $N = K(K-1)+1$. Hence K can be found approximated to $\sqrt{N}$.

For any node *i* which intends to execute its CS, the algorithm works as follows:

Entry Section: Process *i* multicasts the *REQUEST* message to all the nodes in its S_i including itself. The intersection nodes can send the *REQUEST* messages to any one of the districts to which they belongs.When aprocess j receives the *REQUEST* message, it sends *LOCKED* message to site *i* if it has not yet sent it to any other site from the time it received *RELEASE* message. Or else it queues the *REQUEST*.

CS Execution: Process *i* executes its CS after receiving *LOCKED* message from all the nodes of its S_i .

Exit Section: After executing its CS, site *i* sends *RELEASE* message to all nodes of its S_i which restores node's right to send *LOCKED* message to any other pending requests in the queue.

This basic algorithm is prone to deadlock which is handled as follows: Assume that a site *j* has *LOCKED* message to some site *k* and it later receives a *REQUEST* message from any other site *i* ($i \neq k$). Then, nodej sends *FAILED* = to site *i* if $TS_k < TS_i$, otherwise it sends *INQUIRY* message to site *k*. When such a process *k* receives *INQUIRY* message, it sends *RELINQUISH* message to site *j* if site *k* has received *FAILED* message from at least one site in S_k , and has not received new *LOCKED* message from it(after receipt of *FAILED* message).

3. The Proposed Algorithm

Our proposed algorithm presents an improvement to the Maekawa's distributed mutual exclusion algorithm. From Maekawa's algortihm [1] it is clear that the role oft he arbitratoris to resolve the conflicting requests to enter CS. Every node has the responsability to become an arbitrator to handle the conflicting requests coming from the quorum to which it belongs. Nodes that belong to more than one quorum (referred to as intersection nodes), act as inter quorum arbitrators, resolving conflicting requests arising from nodes of different quorums. Because of the role played by these intra and inter quorum arbitrator,the mutual exclusion condition is maintained throughout the entire network.

Intersection nodes can also act as intra-quorum arbitrators since they are also the members of the quorum. Here we see that, since the intersection nodes can act as both inter-quorum and intra-quorum arbitrators, and since every quorum should have at least one intersection node, all conflicting requests can be resolved by communicating with intersection nodes of the system. This way we can achieve per CS in vocation with respect to Maekawa's algorithm [1], as all the messages required to communicate with non-intersection nodes can be eliminated.

Hence our proposition is: Maekawa's distributed mutual exclusion algorithm can perform better (in terms of number of messages required) by restricting the entire algorithm related communication to be carried out with only the intersection nodes in the quorum.

In Maekawa's algorithm [1], all nodes in the quorum are intersection nodes (from Condition 4 for construction of quorums which is outlined in section 2) and hence all nodes works as inter-quorum arbitrators. To ensure that number of intersection nodes in the network is lesser than number of nodes in the quorums,

we propose to liberalize the conditions for construction of quorums in Maekawa's algorithm [1]. The quorums in our algorithm are constructed using the following conditions:

1. $\forall i \ \ \forall j, \ S_i \cap S_j \neq \theta, \ i \neq j, \ 1 \leq i, j \leq y$ where y is the number of quorums, $y \leq N$.

2. Node i belongs to at least one of the quorums.

3. The number of nodes in the quorum need not to be equal.

Here, we have presented the conditions in the same way as done by Maekawa's algorithm in the previous section so that the reader may note the difference. Conditions 1 and 2 are required to ensure correctness of the algorithm. In Maekawa's algorithm [1], it was required to have *K* number of nodes in all the quorums to ensure that all nodes perform an equal amount of work for each CS invocation which is a desirable feature of a truly distributed system. The system using our algorithm would be a pseudo-distributed system as the non-intersection nodes do not participate in CS invocation of other nodes and hence condition 3 follows. The basic working of the algorithm and the required types of messages need not to be modified. This improvisation would shift the responsability to maintain the mutual exclusion condition to the intersection nodes.

4. Proof of correctness

4.1 Mutual Exclusion

Mutual exclusionis achieved when no pair of processes is ever simultaneously in its critical section. For any pair of processes, one must leave its CS before the other may enter.

Theorem 41 *The proposed algorithm ensures the mutual exclusion property*.

Proof: Byc contradiction. Let us assume that, any two nodes *i* and *j* are executing the CS simultaneously. Let S_i and S_j be the quorums of *i* and *j* respectively. Let S'_i and S'_j be sets of intersection nodes of S_i and S_j respectively. Let *k* be a node that belongs to the intersection of S_i and S_j.

Consider the case when $Si = Sj$ (i.e.*i* and *j* belong to the same quorums),then we choose *k* from S'_i . Since $S_i = S_j$, we have $S'_i = S'_j$. Thus *k* belongs to S'_j , hence *k* belongs to both S'_i and S'_j. If $S_i \neq S_j$, since *k* belongs to both S_i and S_j, *k* is an intersection node, *k* belongs to both S'_i and S'_j .

Since *i* is executing the CS, *i* has captured the *LOCKED* message from all the node belonging to S'_i including *k*. Since *j* is also executing the CS, *j* also should have captured the *LOCKED* messages from all the nodes belonging to S'_j including *k*. Thus *k* has been locked by 2 requests simultaneously. However, according to the algorithm only one request can lock a node at a time. Thus maximum of only one process can execute the CS at any time.

This proof holds well when *i* and *j* belong to same quorum as well as different quorums. Thus we see that the proposed improvement does not affect the correctness of the algorithm.

4.2 Deadlock and Starvation

The system of nodes is said to be deadlocked when no process is in its CS and no requesting process can ever proceed to its CS Starvation occurs when one process must wait indefinitely to enter its CS even though other processes are entering and exiting their own critical sections.

Theorem 42 The proposed algorithm is deadlock and starvation free.

Proof: Since no two requests carry same timestamp (priority), total ordering is achieve among requests. If the total ordering condition is followed strictly "circular wait" condition is not satisfied, and hence deadlock cannot occcurs [11].

If an arbitrator (here an intersection node) finds out that it has actually violated the total ordering condition by sending *LOCKED* message to a request with higher priority when there is a request with lower priority waiting in the request queue, it sends an *INQUIRY* message to the recipient of the *LOCKED* message.Then if the recipient node has already started executing the CS, it will not reply. If the recipient node has not yet entered the CS and if it receives a *FAILED* message from at least one of the intersection process, then it would send the *RELINQUISH* message to the arbitrator and release the lock on that node. Then the arbitrator can get locked to the request with lesser timestamp. Here the"Nopreemption"condition is not satisfied. Thus in any case, adeadlock situation cannot occur in the system.

Since no modification has been done to the way the timestamp of a node is used or updated, even the improvised algorithm is starvation free,similar to the original Maekawa's algorithm [1].

5. Performance analysis

Let *M* be the number of intersection nodes in the quorum. In the best case where there is no relinquishment happening, we have *M REQUEST* messages being sent by the requesting node for every CS invocation. The node receives *M LOCKED* messages. After executing its CS, node sends *M RELEASE* messages. Thus 3M number of messages is required. In the worst case, where every LOCKED message is relinquished, we have additional K number of *INQUIRY* and *RELINQUISH* messages each. Thus $5M(3M + 2M)$ number of messages is required. Hence the number of messages required for every CS execution after modification is cM, where $3 \leq c \leq 5$.

The value of *M* depends on the way the nodes have been distributed into various quorums. When $M = 1$, then the system is similar to a centralized system. When $M = N$ for all the nodes, then the algorithms performs similar to that of Ricart-Agrawala's algorithm [12]. When $M = \sqrt{N}$, the algorithm performs similar to original Maekawa's algorithm. Also, it can be noted that in any case, the number of intersection processes in aquorum is lesser than or equal to the number of messages required by the original algorithm. The system can be designed is such a way that $M < \sqrt{N}$ for all the quorums of the system, in which case the improvised algorithm would require lesser number of messages than the original Maekawa's algorithm.

6. Conclusion

In this paper, we have proposed a permission based distributed mutual exclusion algorithm which is an improvement of Maekawa's algorithm. The proposed algorithm is a modification of Maekawa's ditributed mutual exclusion algorithm and significant reduction in the number of messages is being achieved by restricting the communication of any node which wants to execute CS with the intersection nodes of the quorums. The proposed algorithm does not introduce any additional overheads over the existing Maekawa's algorithm which requires $3K$ to $5K$ number of messages per CS invocation, where K is the number of nodes in the quorum ($M < K$).

References

[1] Maekawa, M (1985). A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems, *A CM Transactions on Computer Systems*, 3 (2) 145-159.
[2] Singhal, M. (1993). A taxonomy of distributed mutual exclusion, *Journal and Parallel and Distributed Computing*, 18, 145-159
[3] Raynal., M. (1991). A simple taxonomy for distributed mutual exclusion algorithms, *ACM Operating Systems Review*, 23 (2) 47-51
[4] Garcia-Molina, H., Barbara, D. (1985). How to assign votes in a distributed system, *Journal for the Association for Computing Machinery*, 32 (4) 841-860
[5] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G. (2000) Understanding Replication in Databases and Distributed Systems, *ICDCS 2000 Proceedings*, p. 464-474
[6] Ananthanarayana, V.S., Vidyasankar., K. (2006). Dynamic Primary Copy with Piggy-Backing Mechanism for Replicated UDDI Registry, *ICDIT 2006, Lectures Notes in Computer Science*, 4317, Springer, p. 389-402
[7] Bharath Kumar, A.R., Pradhan, B.U., Ananthanarayana, V.S (2008). An Effcient Lazy Dynamic Primary Copy Algorithm for Replicated UDD Registry, *ICIP-2008*, p. 564-571
[8] Pradhan, B.U., Bharath Kumar, A.R., Ananthanarayana, V.S. (2008). An Effcient eager Dynamic Primary Copy Algorithm for Replicated UDD registry, *Proceedings of ICCNS-2008*, p. 161-166
[9] Pradhan, B.U.. Bharath Kumar, A.R. Ananthanarayana, V.S A Tree-based Dynamic Primary Copy Algorithm for Distributed Databases, *ICDCN 2009 Lectures Notes in Computer Science*, in press
[10] Lamport, L. Time, Clocks and the ordering of Events in a Distributed System, *Communications of the ACM*, 1978, p. 558-565
[11] Cuffman, E.G., Elphick, J.M., Shoshani (1971). A System Deadlocks, *ACM Computing Surveys*, 66-78
[12] Ricart, G. Agrawala, A.K (1981). Anoptimal algorithm for mutual exclusion in computer networks, *Communications of the ACM*, 24 (1) 9-17