An Improved Algorithm for Distributed Mutual Exclusion by Restricted Message Exchange in Voting Districts

Bharath Kumar A.R. and Pradhan Bagur Umesh
National Institute of Technology Karnataka, Surathkal, Managalore, India
a.r.bharathkumar@gmail.com and pradhan@ieee.org

Abstract

This paper presents an improvement to the Maekawa's distributed mutual exclusion 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 in the system. This improvement does not introduce any additional overheads over the existing Maekawa's algorithm which requires 3K to 5K number of messages per critical section invocation, where K is the number of nodes in the voting district ($M \leq K$). This reduction in number of messages is achieved by restricting the communication of any node which wants to execute Critical Section with the Intersection nodes of the voting district, without causing any modification of the basic structure of the algorithm. This improvisation preserves all the advantages of the original Maekawa's algorithm.

1. Introduction

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

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

Garcia-Molina and Barbara [4] first introduced the concept of coterie which could be mainly used to devise permission based DME 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 form nodes of any quorum in the coterie. Maekawa's algorithm [1] was the 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 form all the nodes in its respective group and these groups were created such that any two groups had at least one node in common (referred to as Intersection nodes) which act as arbitrators. In our method, we further restrict the communication of the nodes which wants to execute its CS to the Intersection nodes and achieve DME in lesser number of messages.

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

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

The organization of remainder of paper is as follows. In next section we review Maekawa's algorithm [1]. In section 3, we present the proposed algorithm, including the proof for correctness and deadlock prevention. We illustrate the algorithm using an e.g. in section 4. Then, in section 5 we present the analysis of the proposed algorithm. Finally, we conclude in section 6.

2. Review of Maekawa's algorithm

In this section, we present the system model for our algorithm and review Maekawa's algorithm.

2.1 System Model

We assume the following system model which is common to Maekawa's algorithm and the proposed algorithm. The system has 'N' sites (1, 2, 3..i.., j.., N).We use the terms sites, nodes and process interchangeably. The underlying communication

978-0-7695-3513-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICIT.2008.51

channel is assumed to be error free and reliable, and message passing between nodes to be asynchronous. The nodes communicate by exchanging messages and shared memory does not exist. The propagation delay of messages is unpredictable but finite.

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

We compare the timestamp as follows: $TS_i < TS_j$ iff $(L_i < L_j)$ or $(L_i=L_j$ and $i< j)$.

2.2 The Algorithm.

In Maekawa's algorithm, 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 as voting districts (S_i, $1 \leq i \leq N$). The voting districts are constructed such as to satisfy the following conditions:

1. $\forall i\ \forall j,\ S_i \bigcap S_j \neq \Phi;\ i \neq j,\ 1 \leq i, j \leq N$
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 with in K S_i's, $1 \leq i, j \leq N$

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

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

Entry Section: Site 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 it belongs. When a site j receives the *REQUEST* message, it sends *LOCKED* message to site j 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: Site 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 the nodes of its S_i which restores the right to nodes 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 form any other site i ($i \neq k$). Then, site j sends *FAILED* to site i if $TS_k < TS_i$, otherwise it sends *INQUIRY* message to site k. When such a site 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. Proposed Algorithm

3.1 Improvement

From Maekawa's algorithm [1] it is clear that the role of the arbitrator is to resolve the conflicting requests to enter CS. Every node has the responsibility to become an arbitrator to handle the conflicting requests coming from the voting district to which it belongs. Nodes that belong to more than one voting district (referred to as Intersection nodes), act as inter-voting district arbitrators, resolving conflicting requests arising from nodes of different voting districts. Because of the role played by these intra and inter voting district arbitrators, the mutual exclusion condition is maintained throughout the entire system.

Intersection nodes can also act as Intra-voting district arbitrators since they are also the members of the voting district. Here we see that, since the intersection nodes can act as both inter-voting district and intra-voting district arbitrators, and since every voting district 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 significant reduction number of messages required per CS invocation w.r.t Maekawa's algorithm [1], as all the messages required to communicate with non-intersection nodes can be eliminated.

Hence we propose that: 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 voting district.

In Maekawa's algorithm [1], all nodes in the voting district are intersection nodes (from the 4th rule for construction of voting districts which is outlined in section 2) and hence all nodes work as inter-voting district arbitrators. To ensure that number of intersection nodes in the system is lesser than number of nodes in the voting districts (in other words, to ensure that all the nodes are not arbitrators), we liberalize the conditions for construction of voting districts in Maekawa's algorithm [1]. The voting districts in our algorithm are constructed using the following conditions:

1. $\forall i\ \forall j,\ S_i \bigcap S_j \neq \Phi;\ i \neq j,\ 1 \leq i, j \leq x$ where x is the number of voting districts, $x \leq N$.
2. Node i belongs to at least one of the voting districts.

3. The number of nodes in the voting districts need not be equal.

Here, we have presented the conditions in the same way as done for 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 voting districts to ensure that all nodes perform 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 messages required need not be modified. This improvisation would shift the responsibility to maintain the mutual exclusion condition to the Intersection nodes.

3.2 Proof of correctness

By contradiction, let us assume that, any two nodes i and j are executing the CS simultaneously. Let S_i and S_j be the voting districts of i and j respectively. Let S_i' and S_j' be set 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 $S_i = S_j$ (i.e. i and j belong to same voting district), then choose k from S_i' (say) .Since $S_i = S_j$, we have S_i' = Sj'. 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* messages from all the nodes 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 node can execute the CS at any time.

This proof holds good when i and j belong to same voting district as well as different voting districts. Thus we see that the proposed improvement does not affect the correctness of the algorithm.

3.3 Deadlock Prevention and Starvation

Since no two requests carry same timestamp, total ordering is achieved among requests. If the total ordering condition is followed strictly "Circular wait" condition is not satisfied, and hence deadlock cannot occur [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 timestamp when there is a request with lower timestamp 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, thus violating the "Hold and Wait" condition. If the recipient node has not yet entered the CS and if it receives a *FAILED* message from at least one of the Intersection nodes, then it would send the *RELINQUISH* message to the arbitrator and loses the lock on that node. Then the arbitrator can get locked to the request with lesser timestamp. Here the "No preemption" condition is not satisfied. Thus in any case, a deadlock situation cannot occur in the system.

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

4. Illustration of the Proposed Algorithm

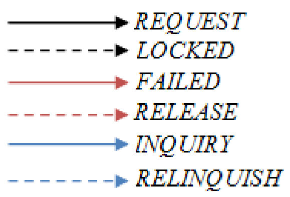

Fig 1. Mapping the arrows to corresponding messages.

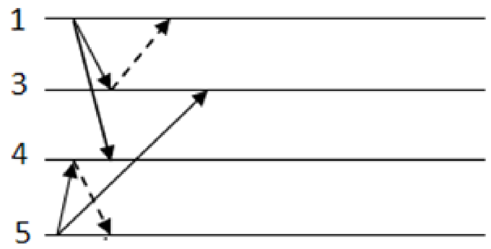

Fig 2. Nodes 1 and 5 want to execute CS.

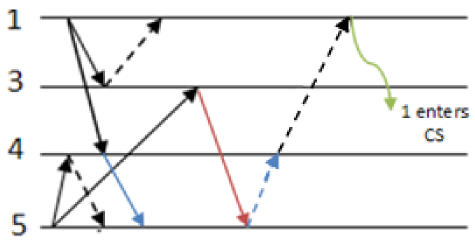

Fig 3. Node 1 enters CS.

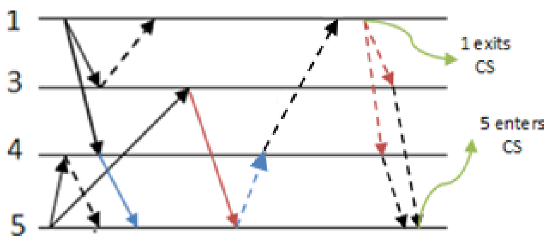

Fig 4. Node 1 exits CS and node 4 enters CS.

Consider a system with five sites and two voting districts $S_1= \{1, 2, 3, 4\}$ and $S_2 = \{3, 4, 5\}$. Assume that nodes 1 and 5 want to execute their respective CS and $TS_5 > TS_1$. Note that node 3 and 4 are intersection nodes. The figures are self explanatory.

5. Analysis

Let 'M' be the number of Intersection nodes in the voting district. 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 in cM, where c varies between 3 and 5.

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

6. Conclusion

A modification to the famous Maekawa's distributed mutual exclusion algorithm has been proposed. Without any modification to the core of the algorithm, significant reduction in the number of messages is being achieved by restricting the all the algorithm related communication to a fewer nodes. the designer of the distributed system now has the freedom to arrange the nodes suitably into voting districts and decide the value of M.

7. References

[1] M. Maekawa, "A √N algorithm for mutual exclusion in decentralized systems", ACM Transactions on Computer Systems Vol. 3, No.2, May 1985, pp.145–159.

[2] M. Singhal, "A taxonomy of distributed mutual exclusion", Journal of Parallel and Distributed Computing, Vol. 18, Yr. 1993, pp. 94–101.

[3] M. Raynal, "A simple taxonomy for distributed mutual exclusion algorithms", ACM Operating Systems Review, Vol.23, No. 2, Yr. 1991, pp. 47–51.

[4] H. Garcia-Molina, D. Barbara, "How to assign votes in a distributed system", Journal for the Association for Computing Machinery, Vol. 32, No. 4, Yr. 1985, pp. 841–860.

[5] Wiesmann, M.; Pedone, F.; Schiper, A.; Kemme, B.; Alonso, G., "Understanding replication in databases and distributed systems",
ICDCS 2000. Proceedings. pp. 464 – 474.

[6] Ananthanarayana V.S, K. Vidyasankar, "Dynamic Primary Copy with Piggy-Backing Mechanism for Replicated UDDI Registry", ICDIT 2006, Lecture Notes in Computer Science, Vol. 4317, Yr. 2006, Springer, pp. 389–402.

[7] Bharath Kumar A.R., Pradhan B. U., Ananthanarayana V.S, "An Efficient Lazy Dynamic Primary Copy Algorithm for Replicated UDD Registry", ICIP 2008, ICIP-2008, pp 564-571.

[8] Pradhan B. U, Bharath Kumar A.R., Ananthanarayana V.S, "An Efficient eager Dynamic Primary Copy Algorithm for Replicated UDD Registry", Proceedings of ICCNS-2008, pp 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, Lecture Notes in Computer Science, *in press*.

[10] L.Lamport, "Time, Clocks and the ordering of Events in a Distributed System", Communications of the ACM, Yr. 1978, pp. 558-565

[11] Cuffman, E. G., M. J. Elphick, and A. Shoshani, "System Deadlocks", ACM Computing Surveys, June 1971, pp. 66–78.

[12] Ricart, G., Agrawala, A. K., "An optimal algorithm for mutual exclusion in computer networks" Communications of the ACM, Vol. 24, No. 1, Jan. 1981, pp. 9-17.