A Comparative Study of Ricart-Agrawala and Maekawa Distributed Mutual Exclusion Algorithm

Mohd Nor, Rizal
Computer Science Department
Kent State University
Kent, Ohio 44240
Email: http://www.cs.kent.edu/˜rnor

***Abstract*—Ricart-Agrawala's and Maekawa's distributed mutual exclusion algorithms were implemented to conduct experiments comparing these two algorithms. It is expected that Ricart-Agrawala's algorithm would perform at $2 * (N - 1)$, while Maekawa's algorithm would perform at $K * \sqrt{N}$, where $3 \leq K \leq 6$. Simulations under different number of processes(N) and different contention load sizes(L), showed that Ricart-Agrawala's algorithm performs as expected while Maekawa's algorithm seems to vary from $3 * \sqrt{N}$ to $4 * \sqrt{N}$. The simulation allows analysis of the data in message exchanges required to enter critical section per node and to help in understanding the underlying behavior of processors under different conditions.**

I. Introduction

Common methods to implement distributed mutual exclusion(DMX) is by using locks and by using tokens. One of the first lock based DMX algorithm was introduced by Lamport's [1] DMX algorithm. In Lamport's DMX algorithm, processors requesting mutual exclusion sents messages to its peers and waits for a reply if it is allowed to enter critical section. Upon completing its critical section, it will notify all its peers by releasing the request. This process is a three message algorithm which consist of REQUEST, REPLY AND RELEASE per critical section. Because of the three messages required per critical section, the number of messages required is $3 * (N - 1)$.

Ricart and Agrawala [2] suggested modifications of Lamport's DMX algorithm to reduce the number of messages required to enter critical section. Since a request is only allowed to enter critical section when a process receives all the REPLY messages from all its peers, a REPLY can be delayed to a node only when a node is done with it's critical section. Hence, they suggested to reduce the number of messages to only REQUEST and REPLY. Thus reducing the number of messages required per critical section to only $2 * (N - 1)$.

Later Maekawa [3] reduce the number of messages required to enter critical section by only a factor of $\sqrt{N}$. He observed that a process does not have to send message to all other processes to lock them. Every process P_i is assigned a request set S_i (quorum) of processes Maekawa showed that minimum quorum size is $\sqrt{N}$ and can be found by solving finite projective planes of N points. For a symmetric set, this can be solved when $N = K * (K - 1) + 1$ (where k is a power of a prime).

In Maekawa's Algorithm, a process enters critical section if it succeeds in acquiring locks from its entire quorum. (Request, Locked Release). This reduces the message complexity of Maekawa algorithm to $3*\sqrt{N}$. However, recall that Maekawa's algorithm has 6 types of messages (Request, Locked Release, Failed, Inquire, Relinquish). The other extra 3 messages are used to avoid deadlock. Hence in very high load, performance could go as large as $6 * \sqrt{N}$.

This report is presented as follows. In section II we presented the experiment parameters used, the generation of quorum sets, the assumptions use to create a realistic simulation. In section III, we present our findings of running the simulation for various parameters and our analysis of the results. Finally in section IV, we summarized our findings and presented our future research interest in this area.

II. Experiment Design

The objective of this experiment was to implement Ricart-Agrawala and Maekawa Algorithm for the purpose of understanding the behavior of the algorithm in different number of nodes(N) and different load factors(L). The program implemented was run several times, with different experiment parameters to collect statistical data. This data is used to plot results and analyze the behavior of the algorithms.

A. Experiment Parameters and Expectations

1) Ricart Agrawala: The parameters used in Ricart-Agrawala simulation started by varying the size of the system, number of nodes(N). The number of nodes used varied from 5 to 50 nodes. In each increment of 5 nodes, I then varied the size of contending nodes, load factor(L) with 1(node), 25%, 50%, 75% and 100% number of contending nodes. It is expected, that in Ricart-Agrawala, the change in load factor(L) will not affect the number of messages being exchanged between nodes. In fact, I expect it to have a constant number of messages being exchange in the system despite the change in load factor. However, the change in the number of nodes(N) should increase the number of messages being exchanged in the system linearly with respect to the number of nodes in the system.

2) Maekawa: The parameters used in Maekawa's simulation started by varying the size of the system, number of nodes(N) for $N =$ 7, 13, 21, 31, 57, 73, 91, 133 , 183, 307 , 381, 553, 871, 993. The chosen N are intentional since these values would result in quorum sets that are symmetric,pairwise,nonnull intersecting sets. I did not consider quorum sets that are not symmetric since acquiring non-symmetric quorum sets were not available. In each increments of N, I then varied the size of contending nodes, load factor(L) with 1(node), 25%, 50%, 75% and 100% number of contending nodes. It is expected that Maekawa's algorithm, the increase in the number of nodes in the system would increase the number of exchanged messages per critical section by a factor of $\sqrt{N}$. In fact, it is expected that when the load is just 1 node requesting critical section, the results should exactly be $3 * \sqrt{N}$. However, it is also expected that as the load factor(L) increases, the amount of messages required per critical section should increase from $3 * \sqrt{N}$ to a maximum of $6 * \sqrt{N}$.

B. Quorum Set Generation

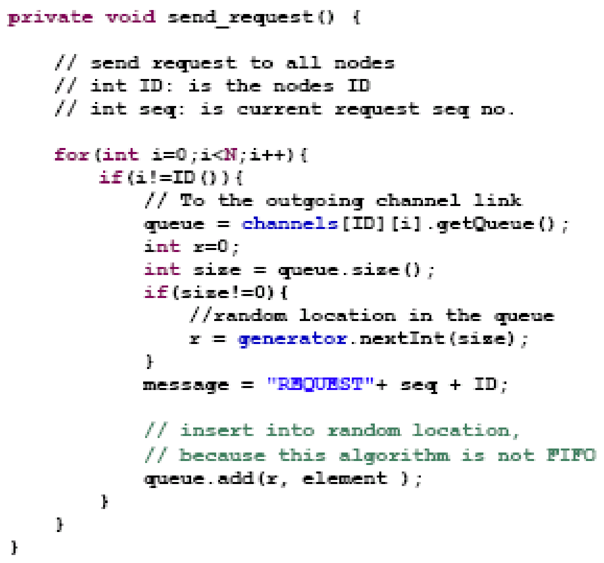
Maekawa's paper [3] did not provide methods for generating the quorum sets. The quorum sets can be generated by solving finite projective planes of N points. However implementation of a generator was not trivial. Therefore in order to save time, I have used data parse from a text file containing all quorum sets. This data is acquired from **Eric Moorhouse's** website [4], who is currently a faculty member in Department of Mathematics at University of Wyoming. The text file contains the set of known solutions for number of nodes $N = 7, 13, 21, 31, 57, 73, 91, 133, 183, 307, 381, 553, 871, 993$.

C. Simulation

The simulator implemented is responsible to handle creation of nodes, selection of nodes by random to be run, and assigning channels to the nodes. A pseudo-random number generator was used to generated a random instance to be run. Each instance of a node was seeded with the $Systemtime * 100 * processID$. The use of a random number generator was necessary to make sure that our simulations were random, such that no two runs have the same computations.

To ensure that the simulation executed as according to the specifications of the algorithm, the channel queues should be implemented differently for each algorithm. For example, in Ricart-Agrawala the channel link is non-fifo. It was then necessary to allow Ricart-Agrawala to have messages arriving not in order sent. To implement this feature, Ricart-Agrawala channel link uses a different kind of queue that would randomized the ordering of messages when a send message is issued. When a node sends a message, it will insert the element in random order of the queue. Thus the head of the queue might change after a send message. Figure 1 illustrate the algorithm used to implement this feature during broadcast of a request.

Eventhough Maekawa uses a fifo channel, it is dependent on a priority queue to keep track of the locking sequence of its members in the quorum set. Hence a separate comparator was needed to handle priority by sequence number and process id of each node's request.



```
private void send_request() {

    // send request to all nodes
    // int ID: is the nodes ID
    // int seq: is current request seq no.

    for(int i=0;i<N;i++){
        if(i!=ID()){
            // To the outgoing channel link
            queue = channels[ID][i].getQueue();
            int r=0;
            int size = queue.size();
            if(size!=0){
                //random location in the queue
                r = generator.nextInt(size);
            }
            message = "REQUEST"+ seq + ID;

            // insert into random location,
            // because this algorithm is not FIFO
            queue.add(r, element );
        }
    }
}
```

Fig. 1. Non-fifo simulation algorithm.

III. Results

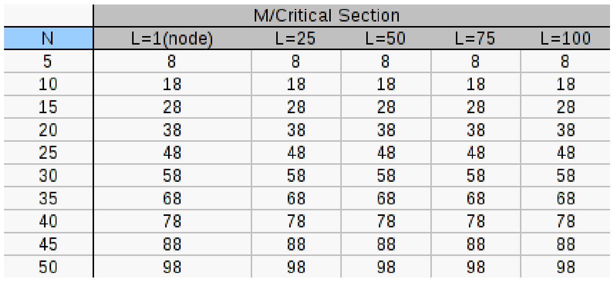
A. Ricart and Agrawala

The results of the experiment for Ricart-Agrawala algorithm for varying number of nodes, N and varying load factors,L is shown in figure 2. It can be seen from the table that for any given fixed number of nodes, N, varying the load factor L does not change the number of messages being exchange in the system. In fact, each row of number of nodes, gives exactly the same results of number of messages per critical section for any given load factor, L. The results is as expected, since even though the load factor, L increases, and hence more contending nodes are sending messages, the amount of messages sent and receive by each node requesting critical section remains the same.

Figure 3 shows a graph of load factors, L vs. number of nodes, N. It shows the messages exchanges for each critical section. The graph shows that regardless for For any given fixed load factor, L, varying the number of nodes gives a linear change in the number of messages per critical section. The results shows that the change is exactly $2 * (N - 1)$ without any variance. This results is as expected because each node requires to send to exactly N number of nodes twice for Request and Reply. Hence, the graph shows a linear growth regardless the load factor, L.

B. Maekawa

Figure 5 shows number of messages per critical section vs. load factor,L. In the graph, it can be seen that an increase in load factor,L increases the messages per critical section. This is true until it reaches a threshold value of load factor of 50%, $L = 50\%$. After the threshold value of 50%, the graph seems to be constant. This results is unexpected since we were



	M/Critical Section				
N	L=1(node)	L=25	L=50	L=75	L=100
5	8	8	8	8	8
10	18	18	18	18	18
15	28	28	28	28	28
20	38	38	38	38	38
25	48	48	48	48	48
30	58	58	58	58	58
35	68	68	68	68	68
40	78	78	78	78	78
45	88	88	88	88	88
50	98	98	98	98	98

Fig. 2. Table of data for Ricart-Agrawala.

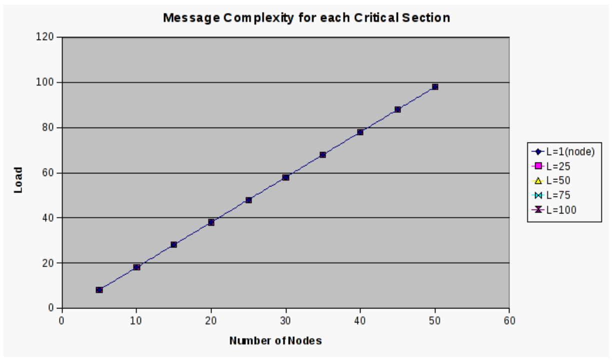
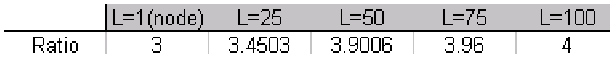


Fig. 3. Load vs. Number of Nodes

expecting a linear growth as the load factor, L increases. To analyzing the threshold point, the ratio of messages per critical section to the quorum size is calculated. It is found that the ratio grows from 3 to 4. As in figure 4, a ratio for $N = 57$ and the ratio of messages per critical section to Quorum size of 7 is calculated. The ratio shows a slow increase from 3 to 4.



	L=1(node)	L=25	L=50	L=75	L=100
Ratio	3	3.4503	3.9006	3.96	4

Fig. 4. Ratio of Number of messages per critial section to quorum size.

It is expected that the growth would be linear from $K * \sqrt{N}$, where $3 \leq K \leq 6$. However, the results was not as expected and it shows that the growth seems to grow only to a point of $K = 4$. Looking at the generated output file, it can be seen that as the load factor, L increases, most nodes requesting nodes will fail and seems to have more failed messages. This extra messages increases the messages required per critical section. However, when the load factor goes above 50%, it seems that most of the nodes are participating. Even in 100% load factor, it doesn't seem to have to wait for all nodes to enter CS. Thus it is doubtful, that a high load would result in $K = 6$.

Figure 6 shows number of messages per critical section vs. number of nodes,N. The Message complexity changed in the order of square root of N. Regardless of the load size, there is a trend of growth for N. The growth is close to a square root of N. This behavior is because as N increases it will require more messages to be sent to the quorum. The quorum size is approximately equals to $\sqrt{N}$.

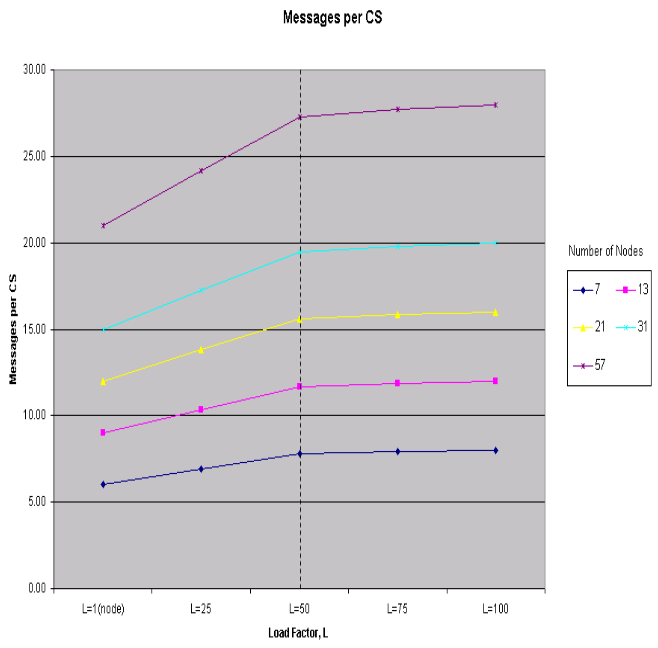


Fig. 5. Messages/CS vs. Load Factor,L.

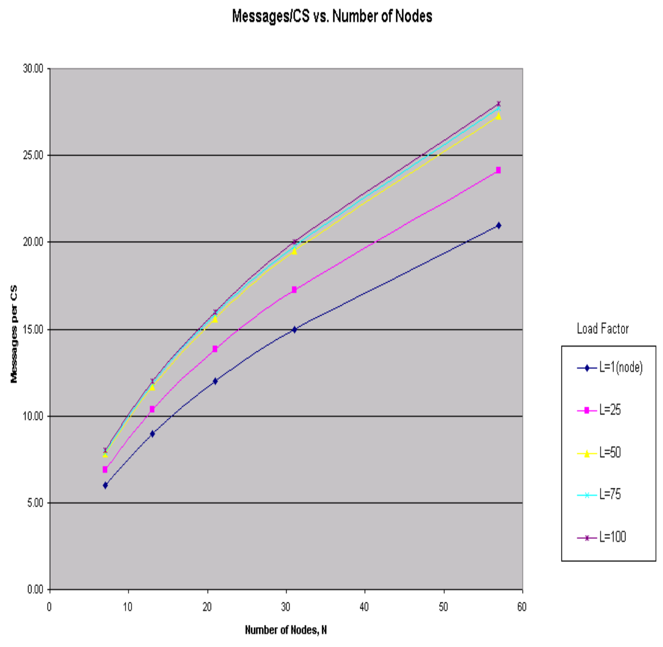


Fig. 6. Messages/CS vs. Number of Nodes, N.

IV. Conclusion and Future Research

In this comparative study, it is found that even though Ricart-Agrawala requires more messages, it is however, simplistic and assures constant performance even in high load.

Maekawa, on the other hands is much more difficult to implement, however, provides fewer messages per critical section even in high load. It is observed that even in high load it seems to perform by at most $4 * \sqrt{N}$ even though the expected upper bound is $6 * \sqrt{N}$

In this study, the quorum sizes were chosen such that the quorum size are symmetric for all sets. A Study of Maekawa's algorithm, when the quorum size is not entirely symmetric is a research I intend to pursue in the future. There are many ways to generate a non-symmetric quorum from an optimized quorum size, however, analyzing which method is most suitable for different scenarios would allow us to create possible practical applications.

Due to its simplistic nature of Ricart-Agrawala, additional algorithm can be used to make Ricart-Agrawala to work in practical network applications. In the future, it would be interesting to study the effects of nodes synchronization for the insertion or deletion of a node in the system. Making Ricart-Agrawala more robust would allow it to be used for practical network implementations.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[2] G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks," *Commun. ACM*, vol. 24, no. 1, pp. 9–17, 1981.

[3] M. Maekawa, "A n algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, pp. 145–159, 1985.

[4] "Projective planes of small order," `http://www.uwyo.edu/moorhouse/pub/planes/`.