Distributed Computing
Course Project
Distributed Mutual Exclusion

ANURAG MODI CS16MTECH11003
SHAMIK KUNDU CS16MTECH11015

April 26, 2017

Contents

1 Abstract

The project presents the implementation of Distributed Mutual Exclusion Algorithms (**1.** Suzuki-Kazami **2.** Neilsen-Mizuno). Both algorithms belong to the category of token based algorithms.

1. Suzuki-Kazami: The algorithm assumes a fully connected physical network. The algorithm was proposed as a better alternative to the Ricart-Agrawala mutual exclusion algorithm in terms of message exchanges. Suzuki-Kazami requires atmost N message exchanges for one mutual exclusion invocation, whereas the Ricart algorithm requires $2 * (N - 1)$ message exchanges per invocation.

2. Neilsen-Mizuno: The algorithm assumes a fully connected, reliable physical network and a directed acyclic graph(dag) structured logical network. Using the best topology, the algorithm attains comparable performance to a centralized algorithm; i.e., **three messages per critical section entry**, and this happens in the case of assuming a star topology as the logical topology for the network.

2 Problem Statement

Implement two distributed token based mutual exclusion algorithms. (1) Suzuki-Kazami and (2) Neilsen-Mizuno. Both the algorithms are implemented in C++.

Further message complexity and average time taken to enter the CS by each process has been analyzed.

3 Suzuki-Kazami

3.1 Main Idea

1. A site can access the lock(critical section) if it has a token.
2. Every process maintains a sequence number (request id).
   (a) A request is of the form (i, m) . This means that P_i wants its m^{th} access to the lock.
   (b) P_i keeps an array $RN_i[1...N]$.
   (c) $RN_i[j]$ is the largest sequence number received from j.
   (d) When P_i receives (j, m) , it sets $RN_i[j] = max(RN_i[j], m)$
3. **Token**
   (a) A queue(Q) of requesting sites.
   (b) An array of sequence numbers LN.
   (c) $LN[i]$ is the sequence number of the latest request that P_i executed.

3.2 Requesting the CS

When node i wishes to enter its critical section and does not hold the token, it increments its request counter $RN_i[i]$ and sends a request message of the form $REQUEST(i, RN_i[i])$, to all other $N-1$ nodes. It then waits for the arrival of the token message. If it has the token or the token arrives, it proceeds to enter its critical section.

The token message has the form $PRIVILEGE(Q, LN)$ where Q is a queue of requesting nodes and LN is an array of size N which contains the sequence number of the request granted most recently to each node.

3.3 Releasing the CS

After exiting its critical section, node i updates $LN[i]$ with its current request granted $RN[i]$. Next, it enqueues all nodes not in Q from whom it has received a request which has not been granted yet. Nodes are inserted at the rear of the queue in an ascending node number order.

If **there exists a process in** Q, the token is sent to the process at the front of it.

If Q **is empty**, node i retains the idle token.

A node holding an idle token can enter its critical section without the need to send a request message.

When a request from *nodej* arrives, $RN_i[j]$ is updated with the most current request number ever received from it in order to discard out-dated information. This would take care of old requests already granted and out-of-order request messages. If *nodei* is holding the token, not requesting its critical section and the most current request from j has not been served yet, then the token is sent to *nodej* .

4 Neilsen-Mizuno

4.1 Main Idea

1. Physically, the nodes are **fully connected** by a reliable network. Logically, they are arranged in a **dag structure** with only one sink node.
2. A **PRIVILEGE** message represents the token; when a node receives a **PRIVILEGE** message, it can enter the critical section.
3. Each node maintains three simple variables: integer variables **LAST** and **NEXT**, and a boolean variable **HOLDING**. The logical dag structure indicates the path along which a **REQUEST** message travels and is imposed by the **LAST** variables in the nodes.

4.2 Requesting the CS

When node i wishes to enter its critical section and does not held the token, i requests for token to the process denoted by it's **LAST** variable and waits for token to arrive.

4.3 Releasing the CS

After a process is done with it's critical section execution and it has token, it sends the token to the process denoted by it's **NEXT** variable.

4.4 Initiation

Since logical structure needs to be created from the physical structure only, the initiator starts broadcasting **INITIAL** message to all it's neighbours where **INITIAL** message contains the pid of the sending process.

5 Testing

For testing, we created the various network topologies. Each node is executing one process. We executed the algorithm and then calculate the time taken by these processes to complete the operations.

Number of nodes	3, 4, 5, 6
Sleep time	Exponention distribution with delay1 = 4μs delay2 = 10μs

6 Results and Comparisons

To compare the performance of these implementations we measure the average time for each operation for both the implementations over a varying number of nodes. The results of these comparisons are given below -

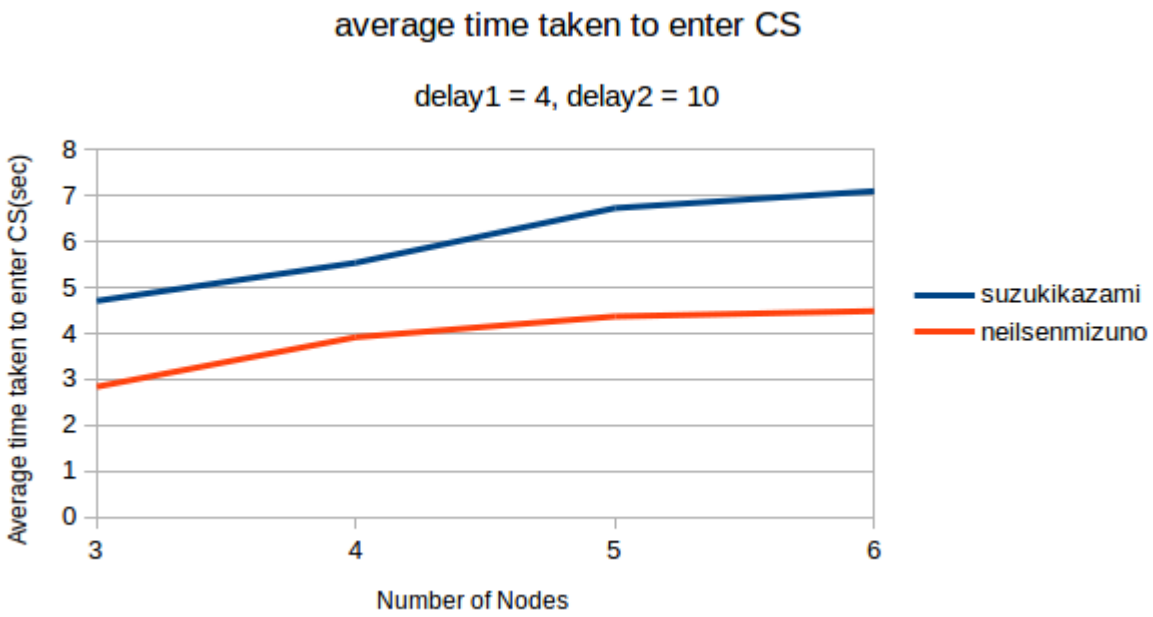

Figure 1: Average time taken to enter CS

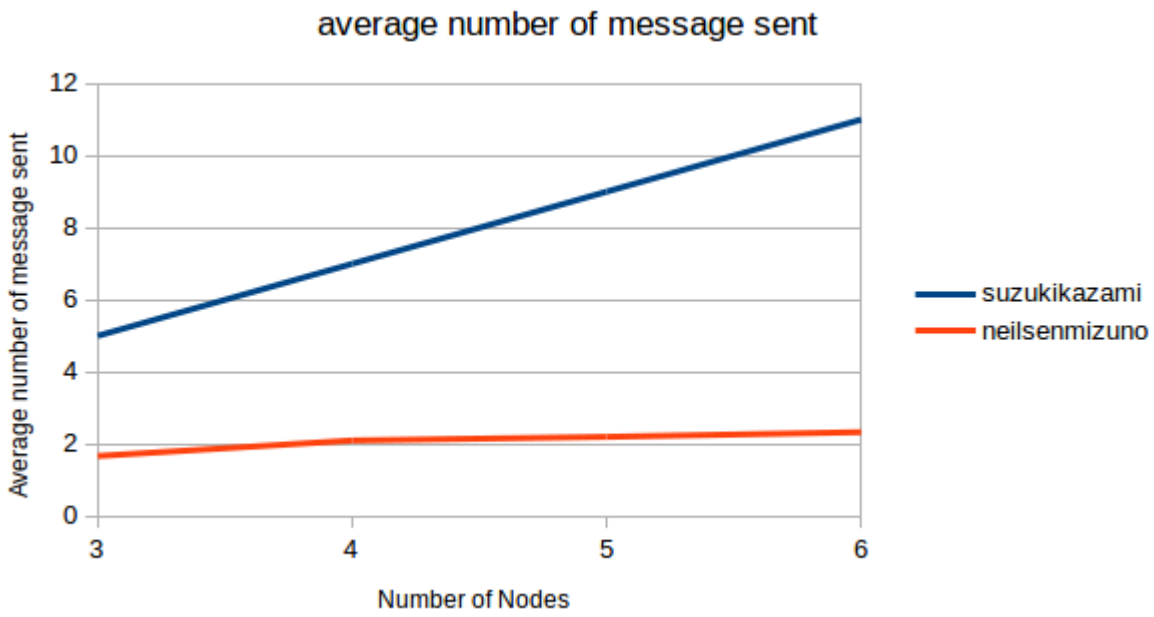

Figure 2: Average number of Message Sent

7 Conclusions

As we can see from the graphs above, Neilsen-Mizuno showed much better performance both in terms of the number of messages sent and time taken to enter in critical section after a request is made on identical setup. And the performance gain is more with increasing number of processes in the distributed system. It can be shown theoretically that the upper bound of messages per critical section entry is O(N) for Suzuki and Kasami's algorithm and O(D+1) for Neilsen-Mizuno algorithm where D is the diameter of the logical structure.

References

[1] Mitchell L. Neilsen , Masaaki Mizuno. *A Dag-Based Algorithm for Distributed Mutual Exclusion.* Department of Computing and Information Sciences, Manhattan, Kansas, 1991.

[2] Ichiro Suzuki , Tadao Kasami. *A Distributed Mutual Exclusion Algorithm.* Osaka University, Japan,1985.