Assignment No. 04

Semester	B.E. Semester VIII – Computer Engineering
Subject	Distributed Computing Lab
Subject Professor In-charge	Dr. Umesh Kulkarni
Assisting Professor	Prof. Prakash Parmar
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Mutual exclusion while addressing clock synchronization.

Ensuring Mutual Exclusion in a Distributed System with Clock Synchronization Challenges

1. Introduction

In distributed systems, multiple processes often need to coordinate access to a shared resource while ensuring mutual exclusion. Since there is no global clock, achieving synchronization while maintaining efficiency is a key challenge. This document explores how mutual exclusion can be ensured while addressing clock synchronization issues and compares the performance trade-offs of non-token-based algorithms such as Lamport's Algorithm and Ricart–Agrawala's Algorithm.

2. Challenges in Mutual Exclusion with Clock Synchronization

2.1 Lack of a Global Clock

- Distributed systems do not have a single global clock, leading to inconsistencies in event ordering.
- Timestamp-based mutual exclusion algorithms must rely on logical clocks.

2.2 Message Delays and Network Latency

- Network latency can cause out-of-order message delivery, leading to inconsistencies.
- Algorithms must be designed to tolerate delays while ensuring fairness.

2.3 Failure Handling

- Processes may crash or become unreachable, requiring fault tolerance mechanisms.
- Distributed mutual exclusion must handle process failures gracefully.

3. Non-Token-Based Mutual Exclusion Algorithms

3.1 Lamport's Algorithm

Overview

- Uses logical timestamps to order requests.
- A process sends a request message with a timestamp to all other processes.
- Each recipient acknowledges the request and queues it.
- The requesting process enters the critical section when:
    - It has received acknowledgment from all other processes.
    - Its request is at the front of its queue.
- After exiting, the process informs others to update their queues.

Advantages

- Ensures fairness by processing requests in timestamp order.
- Works in asynchronous environments without requiring synchronized clocks.

Disadvantages

- High message complexity: **3(N-1) messages per request** (request, acknowledgment, release).
- Susceptible to failures—if a process crashes, it may prevent others from entering the critical section.

3.2 Ricart–Agrawala's Algorithm

Overview

- Optimizes Lamport's Algorithm by reducing message complexity.
- A requesting process sends a request to all other processes.
- Recipients reply immediately if they are not in the critical section.
- If a recipient is in the critical section or has a lower timestamp request, it defers its reply.
- The requesting process enters the critical section when all replies are received.

Advantages

- Reduces message complexity to **2(N-1) messages per request** (request and reply).
- Ensures fairness and proper ordering of critical section access.

Disadvantages

- Still susceptible to process failures, as waiting for a reply from a failed process can cause indefinite blocking.
- More complex than Lamport's Algorithm in handling failures.

4. Performance Trade-offs

Algorithm	**Message Complexity**	**Fairness**	**Failure Resilience**	**Synchronization Overhead**
Lamport's Algorithm	3(N-1)	High	Low (blocking)	High
Ricart–Agrawala's Algorithm	2(N-1)	High	Moderate (blocking)	Moderate

4.1 Message Complexity

- Ricart–Agrawala's Algorithm has **lower message complexity** than Lamport's

Algorithm, making it more efficient.

4.2 Fairness

- Both algorithms ensure fairness by processing requests in timestamp order.

4.3 Failure Resilience

- Lamport's Algorithm suffers from higher blocking issues if acknowledgments are lost.
- Ricart–Agrawala's Algorithm performs better but still requires additional fault tolerance mechanisms.

4.4 Synchronization Overhead

- Both algorithms rely on logical clocks, reducing the need for physical clock synchronization.
- Lamport's Algorithm has higher overhead due to additional messages for queue management.

5. Conclusion

Both Lamport's Algorithm and Ricart–Agrawala's Algorithm provide effective solutions for ensuring mutual exclusion in a distributed system. While Ricart–Agrawala's Algorithm offers improved efficiency by reducing message complexity, it still faces challenges in handling process failures. Future improvements, such as failure detection mechanisms or hybrid approaches, can further enhance distributed mutual exclusion strategies while addressing clock synchronization challenges.