CS5320: Distributed Computing

Programming Assignment 2
Mutual Exclusion

Design of the program

1. **Global Variables:**

- `n`: Total number of processes.

- `k`: Number of iterations.

- `alph`: Mean of the exponential distribution for out-of-CS time.

- `bta`: Mean of the exponential distribution for in-CS time.

- `timestamp`: Represents the Lamport logical clock timestamp.

- `TimeLock`: Mutex lock for updating the timestamp in a thread-safe manner.

- `inCS`: Atomic boolean flag indicating whether a process is currently in the critical section.

- `requesting`(in RC): Atomic boolean flag indicating whether a process is currently requesting for the critical section.

- `messagesExc`: Atomic integer for keeping track of the number of messages exchanged

- `RD`(in RC): A vector that contains the deferred requests

- `SendReq`(in RC): A vector that contains the information about sent requests

- `SentRep`(in RC): A vector that contains the information about sent replies

- `comm`: MPI communicator for communication among processes.

- `status`: MPI status for message status.

2. **`getQuorumMembers(int id)` Function:**

- Only in Maekawa Alogirthm.

- Inputs: `id` - Rank of the process.

- Outputs: `Quorum` - List of quorum members for the process.

- Calculates the quorum members based on the process's rank in a grid structure.

- Quorum members are determined by the process's row and column in the grid, excluding itself.

3. **`receive(int rnk, vector<int> quorum)` Function:**

- The function doesn’t have quorum for Roucairol and Carvalho’s Algorithm.

- Inputs: `rnk` - Rank of the process, `quorum` - List of quorum members for the process.

- Handles incoming messages from other processes.

- Manages REQUEST , REPLY and MARKER messages in RC Algorithm

- Manages REQUEST, GRANT, RELEASE, INQUIRE, YIELD, FAILED, and MARKER messages in Maekawa Algorithm

- Updates timestamps, maintains request queues, handles failures, and coordinates communication.

4. **`reqCS(int rnk, vector<int> quorum)` Function:**

- The function doesn’t have quorum for Roucairol and Carvalho’s Algorithm.

- Inputs: `rnk` - Rank of the process, `quorum` - List of quorum members for the process.

- Requests entry into the critical section by sending REQUEST messages to quorum members in Maekawa and processes in SentRep and SendReq in RC according to the algorithm.

- Waits until receiving sufficient GRANT (or REPLY) messages from the quorum (or processes) to enter the critical section.

5. **`relCS(int rnk, vector<int> quorum)` Function:**

- Inputs: `rnk` - Rank of the process, `quorum` - List of quorum members for the process.

- Exits the critical section and sends RELEASE messages to quorum members in Maekawa and sends REPLY messages to all the processes in RD(Deferred set).

6. **`working(int rnk, vector<int> quorum)` Function**:

- Inputs: `rnk` - Rank of the process, `quorum` - List of quorum members for the process.

- Simulates local computation, requests entry into the critical section, performs actions inside it, and releases it after completion.

- Uses exponential distributions for generating random out-of-CS and in-CS times.

- Logs local computations, request entries, critical section entries, and exit times into a log file.

7. **`main()` Function**:

- Initializes MPI, reads input parameters from a file, and determines process ranks.

- Identifies quorum members for each process.

- Spawns working and receiving threads for process execution.

- Joins threads and finalizes MPI.

8. **Logging:**

- Logs events such as local computations, request entries, critical section entry, and exit times into the "output_MK.log" file and “output_RC.log” file according to respective algorithms.

- Also logs the total number of messages exchanged along with the execution time in “Statistics_MK.log” and “Statistics_RC.log”.

9. **Message Types:**

- REQUEST (-1), GRANT (-2), RELEASE (-3), INQUIRE (-4), YIELD (-5), FAILED (-6), MARKER (-7) in MK.

- REQUEST(-1), REPLY(-2), MARKER(-3) in RC.

10. **Communication:**

- Processes communicate using **MPI.**

- Timestamps are used to order messages and handle concurrency.

Plots:

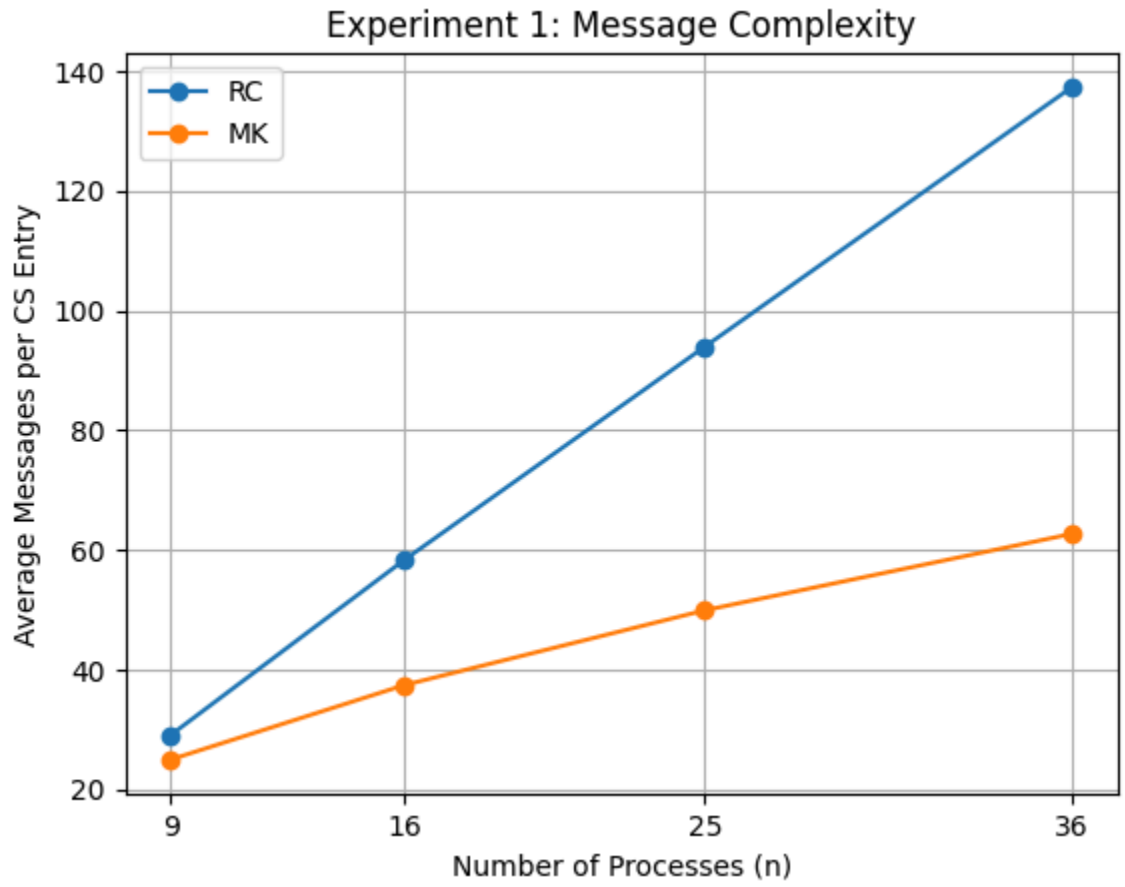
Experiment 1: Message Complexity

k = 15

Number of processes	Average no. of messages per CS entry in RC	Average no. of messages per CS entry in MK
9	29.02668	24.9556

16	58.30332	37.4067
25	93.93974	49.9067
36	137.32	62.68668

Note: The above values are average over 5 runs.



Maekawa's Algorithm (MK) has a lower average number of messages per critical section entry than Roucairol and Carvalho's Algorithm (RC). This suggests that MK is more efficient in terms of message complexity.

The number of processes (n) increases, the average number of messages per critical section entry increases for both algorithms. However, MK consistently demonstrates a lower average compared to RC across different values of n.

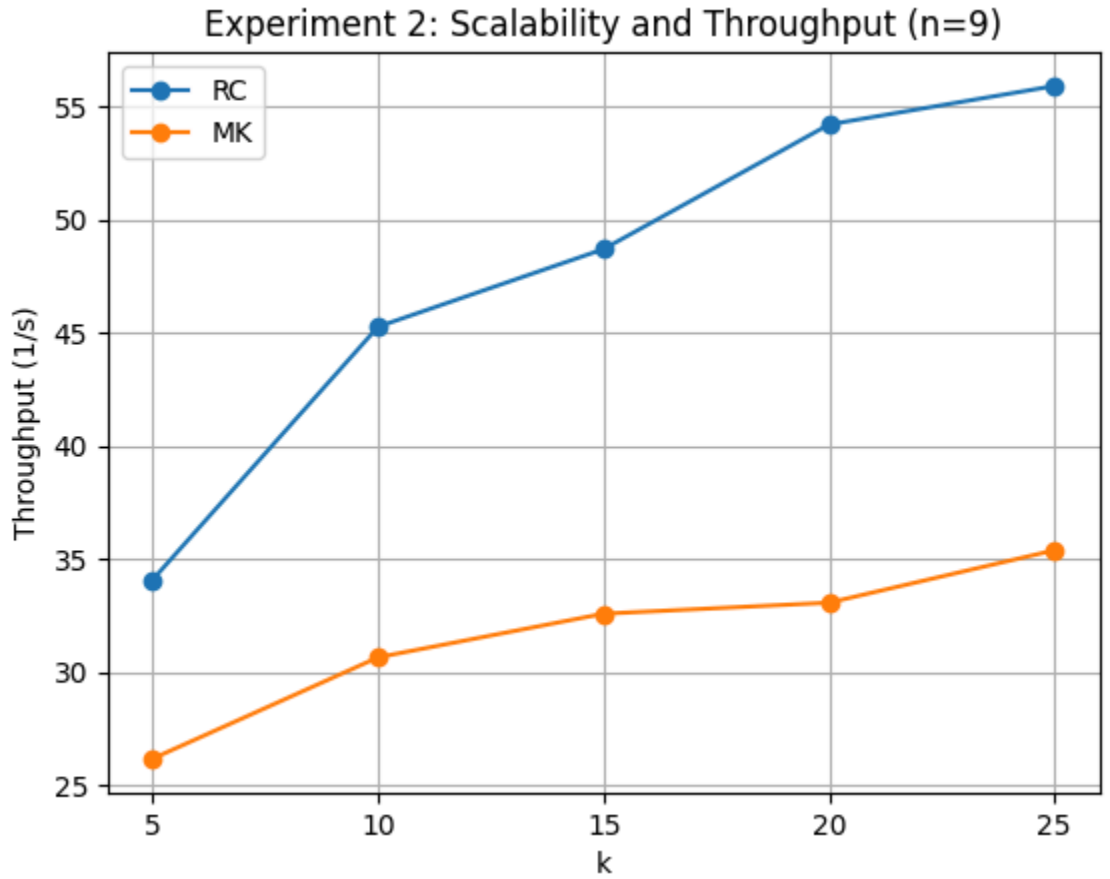
Experiment 2: Scalability and Throughput

n = 9

Throughput = n*k/Execution Time

k	Throughput in RC (1/s)	Throughput in MK (1/s)
5	34.0754	26.1627
10	45.2625	30.6435
15	48.6942	32.5725
20	54.2038	33.0602
25	55.9089	35.3795

Note: The above values are average over 5 runs.



The Roucairol and Carvalho's Algorithm (RC) exhibits higher throughput (number of critical section entries completed per unit time) compared to Maekawa's Algorithm (MK).

As k increases, the throughput for both algorithms tends to increase, but RC consistently outperforms MK in terms of throughput across different values of k.

Maekawa's Algorithm (MK) has an advantage regarding message complexity, with a lower average message overhead per critical section entry than Roucairol and Carvalho's Algorithm (RC).

However, Roucairol and Carvalho's Algorithm (RC) shows better scalability and throughput characteristics.