AOS Project Report

Project Members:

- Sruthi Chappidi (sxc105920)
- Ramya Naraparedi (rxn106120)
- Diksha Sharma (dxs134530)
- Kathryn Patterson (knp061000)

Introduction

The purpose of the project is to design a new distributed mutual exclusion algorithm by combining quorum based and token based approaches and compare it to Maekawa's algorithm. This phase of the project is to use the test bed that was created to compare the two algorithms.

Project Description (From Project Requirements Document)

The new distributed mutual exclusion algorithm adheres to the following:

- The algorithm is quorum-based. Like Maekawa's algorithm, algorithm messages are only be sent between members of the same quorum. The maximum quorum size is $2\sqrt{N}$ where N = total number of nodes is in the system.
- The algorithm is token-based. Like Raymond's algorithm, mutual exclusion is granted once the requesting node has the token. Only one token is used, and it can only be passed with an algorithm message (which restricts the token from being passed between nodes that are not in the same quorum).
- Our design includes code to clearly explain the logic behind the algorithm. The code has been written as pseudo code in Java.
- Our design includes an example of the algorithm when two nodes make requests at the same time. The example is diagrammed and we show all messages passed during the algorithm's operation. Since this is a distributed mutual exclusion algorithm, one node's request will obviously be delayed, but eventually is granted.

This project was to implement the design of the testbed that was created the part 1 of the project.

- The testbed should be able to be configurable so that the user can set which nodes make requests and when.
- The testbed is able to support a minimum of 16 system nodes for both algorithms.
- The testbed is able to demonstrate that mutual exclusion is not violated and that all request are all eventually granted.
- The testbed should also be able to perform metric.

This part of the project takes the created algorithm and implemented test bed and creates reports so that the new algorithm and Maekawa algorithm can be compared.

- How long it took for all process to access critical section
- How long it took for each process to access critical section, max, min and average.
- This information will be used to see which algorithm performs better in various systems non-congested fast release system, non-congested slow release system, congested fast release system, congested slow release system, and an unreliable system.

Performance data

In order to run test we set up several test cases. The way the testbed is designed is that whenever a message is passed between nodes it is stored in a database table where the message type, sender, receiver, and timestamps of the message passing are captured. Using this data after each iteration of the application we were able to gather information. Here are the calculations we extracted from the data.

- Total Time:
  - First Request critical section timestamp of sender – Last timestamp recorded.
  - 1 sec (For multiple processes requesting at same time)
  - 1 sec (For processes requesting one at a time)
- Time for each process:
  - Time stamp of Release sent – Timestamp of Request sent
  - 1 sec (For multiple processes requesting at same time)
  - 1 sec (For processes requesting one at a time)

    This was done for each process that had a requested and the max, min and average were found.
- Number of messages sent
  - The number of messages logged

- KN (K is the size of quorum and N is the number of total processes)

When we using these numbers to compare we found that each process was able to enter into critical section faster in the new algorithm. This can be explained in the fact that the new algorithm has fewer messages that need to be passed in order to get to critical section because of the incorporation of the token. However there is also the issue in using time stamps like this to compare because there is no clock synchronization due to which there is no guarantee that the drift did not cause a problem or affect the comparison results.

- A reliable, non-congested network, where granted resources are released in milliseconds.
  - Our algorithm will perform better than Maekawa's Algorithm because in best case our algorithm does not send any message while maekawa's sends the messages to it quorum members in the same state. We thus exchange less number of messages in best case scenario.
- A reliable, non-congested network, where granted resources are released minutes later.
  - Our algorithm will perform similar to Maekawa's Algorithm because the delay in minutes will put both algorithms at comparable performance.
- A reliable, congested network, where granted resources are released in milliseconds.
  - Our algorithm will not perform better than Maekawa's Algorithm because we are exchanging KN messages as compared to maekawa's K*(N-1) messages. Also we have added messages to database for logging.
- A reliable, congested network, where granted resources are released minutes later.
  - Our algorithm will perform worse than Maekawa's Algorithm because we are exchanging KN messages as compared to maekawa's K*(N-1) messages. Also we have added messages to database for logging.
- An unreliable network, where some messages are never delivered
  - Our algorithm assumes a reliable network so would not perform better than Maekawa's Algorithm.

Screenshots from database logging:

	ALGORITHM	RUN	MESSAGE_NO	MESSAGE	SENDER	RECEIVER	WRITE_TIMESTAMP	MESSAGE_TYPE	SEQUENCE_NO	ORIGINAL_SENDER
1	2	5	1233	Requesting Critical Section	4	4	07-AUG-15 06.45.04.000000000 PM	4	1	4
2	2	5	1236	Requesting Critical Section	5	5	07-AUG-15 06.45.04.000000000 PM	4	1	5
3	2	5	1237	Requesting Critical Section	7	7	07-AUG-15 06.45.04.000000000 PM	4	1	7
4	2	5	1238	Requesting Critical Section	5	4	07-AUG-15 06.45.04.000000000 PM	4	1	5
5	2	5	1239	Requesting Critical Section	6	6	07-AUG-15 06.45.04.000000000 PM	4	1	6
6	2	5	1240	Requesting Critical Section	7	6	07-AUG-15 06.45.04.000000000 PM	4	1	7
7	2	5	1241	Requesting Critical Section	5	7	07-AUG-15 06.45.04.000000000 PM	4	1	5
8	2	5	1242	Requesting Critical Section	6	7	07-AUG-15 06.45.04.000000000 PM	4	1	6
9	2	5	1243	Requesting Critical Section	7	5	07-AUG-15 06.45.04.000000000 PM	4	1	7
10	2	5	1244	Requesting Critical Section	6	4	07-AUG-15 06.45.04.000000000 PM	4	1	6

	ALGORITHM	RUN	MESSAGE_NO	MESSAGE	SENDER	RECEIVER	WRITE_TIMESTAMP	MESSAGE_TYPE	SEQUENCE_NO	ORIGINAL_SENDER
1	2	8	1278	Initiate	(null)	(null)	07-AUG-15 06.51.31.000000000 PM	(null)	(null)	(null)
2	2	8	1279	Requesting Critical Section	5	5	07-AUG-15 06.51.33.000000000 PM	4	1	5
3	2	8	1280	Requesting Critical Section	4	4	07-AUG-15 06.51.33.000000000 PM	4	1	4
4	2	8	1281	Requesting Critical Section	5	4	07-AUG-15 06.51.33.000000000 PM	4	1	5
5	2	8	1282	Requesting Critical Section	7	7	07-AUG-15 06.51.33.000000000 PM	4	1	7
6	2	8	1283	Requesting Critical Section	6	6	07-AUG-15 06.51.33.000000000 PM	4	1	6
7	2	8	1284	Requesting Critical Section	5	7	07-AUG-15 06.51.33.000000000 PM	4	1	5
8	2	8	1285	Received message	4	4	07-AUG-15 06.51.33.000000000 PM	4	1	4
9	2	8	1286	Requesting Critical Section	7	6	07-AUG-15 06.51.33.000000000 PM	4	1	7
10	2	8	1287	Requesting Critical Section	6	7	07-AUG-15 06.51.33.000000000 PM	4	1	6
11	2	8	1288	Entering Critical Section	4	4	07-AUG-15 06.51.33.000000000 PM	2	0	4
12	2	8	1289	Requesting Critical Section	6	4	07-AUG-15 06.51.33.000000000 PM	4	1	6
13	2	8	1290	Exiting Critical Section	4	4	07-AUG-15 06.51.34.000000000 PM	2	0	4