Distributed Systems Project

Simulate message delivery guarantees such as FIFO and Arbitrary, and their impact on some Mutual Exclusion Distributed Algorithm

Ankit Pant 2018201035
Harshita Agrawal 2018201014
Ravi Jakhania 2018201018

Outline

Introduction

Introduction

- Distributed systems are quite ubiquitous these days
- Communication between nodes should be robust
- Need a layer of dependable software systems for reliable communication
- This project aims to explore and simulate two modes or channels of communication
  - First In First Out or *FIFO* message ordering channels
  - *Arbitrary* message ordering channels
- We simulate Lamport's Mutual Exclusion Algorithm and measure impact on both message ordering channels

Literature Review

Message Ordering and Group Communication

- Group communication is vital in Distributed Systems
- Order in which the messages are delivered is also important
  - It determines the order of execution of commands and also helps with consistency
- Arbitrary Order
  - No ordering between messages sent by one node to the other
  - Also called *non-FIFO*
  - **Example:** If a process 'P1' sends two messages (m1 and m2) to 'P2' and the timestamp of message *m1* is *t1* and for message *m2* is *t2* where $t1 < t2$ the process 'P2' may either receive message *m1* before message *m2* or it may receive message *m2* before message *m1*

- FIFO Order
  - Messages sent by a process to another is received in the same order that they were sent
  - **Example:** If a process ‘P1’ sends two messages (m1 and m2) to ‘P2’ and the timestamp of message *m1* is *t1* and for message *m2* is *t2* where $t1 < t2$ the process ‘P2’ must necessarily receive message *m1* before message *m2*
  - Relative ordering between multicast messages from different senders is not important

Distributed Mutual Exclusion Algorithm

- Mutual exclusion – one of the core problems
- More than two nodes should not be executing critical section at the same time
- No shared memory $\implies$ semaphores, etc. not possible
- Distributed mutual exclusion algorithms implemented as
    - Token based algorithms
    - Non-token based approach
    - Quorum based approach

Lamport's algorithm

- Token based Distributed Mutual Exclusion Algorithm
- Executes the critical section requests from various processes in the increasing order of timestamps
- Every node keeps a queue called the *request queue*
- Phases of Lamport's algorithm:
    - Requesting the critical section
    - Executing the critical section
    - Releasing the critical section

Lamport's algorithm (Pseudocode)

Requesting the critical section

- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on $request\_queue_i$. ((ts_i, i) denotes the timestamp of the request.)
- When a site S_j receives the REQUEST(ts_i, i) message from site S_i, it places site S_i's request on $request\_queue_j$ and returns a timestamped REPLY message to S_i.

Executing the critical section

Site S_i enters the CS when the following two conditions hold:

L1: S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

L2: S_i's request is at the top of $request\_queue_i$.

Releasing the critical section

- Site S_i, upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.
- When a site S_j receives a RELEASE message from site S_i, it removes S_i's request from its request queue.

Figure 1: Pseudocode for Lamport's Algorithm [4]

Methodology

System Model

- Simulated Distributed System with 6 nodes
- All nodes are non-Byzantine
- Message delivery guaranteed by using TCP as underlying protocol
- Each node in the system can (pre-requisites of Lamport's algorithm)
  - Send and receive messages
  - Request to execute its critical section
  - Communicate with every other node
- Nodes (channels) may arbitrary fail and come back online (handled using timeouts)

Code Walkthrough

Source Code

Please refer to the source code and accompanying video for details on the source code

Experimentation and Results

Results

- *FIFO* message ordering was successfully implemented
- *Arbitrary* message was successfully implemented
- Lamport's algorithm was run on both the ordered channels
- Lamport's algorithm runs correctly on *FIFO* ordered channel
- Lamport's algorithm fails on *Arbitrary* ordered channel
- In both message ordering channels the number of messages exchanged is same $[3 * (N - 1)]$
- Runtime not measured due to random delays in network (can cause inaccurate assessment)

Conclusion

Conclusion

- Message ordering is crucial for correct working of distributed algorithms
- Maintaining Mutual exclusion is also vital in a distributed system
- Lamport's algorithm may fail in an *Arbitrary* ordered channel

Future Scope

Future Scope

- Lamport's algorithm can be modified to run on *Arbitrary* channels
- Resolution may be done by
    - Adding buffer at receivers
    - Buffer orders messages correctly by timestamp
    - After all messages are received and correctly ordered, forward them to the mutual exclusion algorithm

References

- *FIFO executions*, Ajay D. Kshemkalyani & Mukesh Singhal, Page 191, Distributed Computing Principles, Algorithms, and Systems.
- *Causally ordered (CO) executions*, Ajay D. Kshemkalyani & Mukesh Singhal, Page 191, Distributed Computing Principles, Algorithms, and Systems.
- *Distributed mutual exclusion algorithms*, Ajay D. Kshemkalyani & Mukesh Singhal, Page 305, Distributed Computing Principles, Algorithms, and Systems.

Lamport's Algorithm, Ajay D. Kshemkalyani & Mukesh Singhal, Page 309, Distributed Computing Principles, Algorithms, and Systems.