Report 4: Groupy - a group membership service

Emile Le Gallic

October 2, 2024

1 Introduction

In this seminar, the focus is on implementing a group membership service that provides atomic multicast. This task requires synchronizing application-layer processes so they all follow the same sequence of state changes. A key challenge in distributed systems is ensuring that all nodes can maintain coordination, even in the face of network failures and node crashes.

The central topic of this seminar is total order multicast within a distributed system. This is essential because ensuring that multiple nodes perform actions in a synchronized order is a critical component of consistency in distributed systems. Through implementing features such as leader election, failure detection, and message ordering guarantees, this seminar demonstrates how systems can stay resilient and consistent despite failures.

2 Main problems and solutions

During the project, we implemented three Group Membership Service (gms) solutions to address message synchronization and system robustness. Each solution brought specific improvements and challenges:

2.1 gms1: Basic Group Membership Protocol

In **gms1**, the leader is responsible for sending messages to the slaves. While it handles basic group membership and leader election, it has major issues:

- **Single Point of Failure**: If the leader crashes, message synchronization halts, causing nodes to fall out of sync.

Proposed Solution: Implement a leader election to mitigate this, but the overall fault tolerance remains low.

2.2 gms2: Improved Synchronization with Partial Robustness

gms2 attempts to improve the system by ensuring the leader propagates messages to all nodes.

- **Issue**: If the leader crashes before sending messages, not all nodes will receive them, leading to synchronization issues.

Proposed Solution: Introduce mechanisms to detect and recover from leader crashes but synchronization gaps can still occur.

2.3 gms3: Robust Group Membership Protocol

gms3 introduces message acknowledgment and retries, significantly improving system robustness.

- **Leader Crash Handling**: If the leader fails, messages are resent or acknowledged by other nodes, ensuring synchronization across the system.

Summary of Changes: gms1 provides basic functionality but lacks fault tolerance. gms2 improves synchronization, but still suffers from leader crash issues. gms3 adds message acknowledgments and retries, making the system more robust and fault-tolerant.

2.4 gms4: Handling Lost Messages and Fault Tolerance

gms4 introduces message acknowledgment (ACK) handling to ensure that messages are reliably delivered even if some are lost.

- **Lost Messages Handling**: The leader waits for ACKs from all slaves after sending messages, and resends if ACKs are not received within a timeout. This ensures no message is lost, though it may increase latency.
- **Failure Detection**: Since Erlang's failure detector is imperfect and can falsely suspect nodes of crashing, the system is adapted to handle these scenarios by incorporating retries and dynamic timeouts to ensure progress is made.

Summary of Changes: The initial version lacked robustness, while this version (gms4) adds ACKs and retries to handle message loss and false failure detections, improving fault tolerance and reliability in distributed systems.

3 Evaluation

In the first part of the seminar, *gms1*, everything worked smoothly without any issues regarding synchronization. Since no crashes occurred, all nodes were able to process the multicast messages in the correct sequence, ensuring consistent synchronization across the entire system.

However, in the second part, *gms2*, we observed occasional out of synchronization situations. This was due to node crashes, which introduced inconsistencies in the message delivery and group membership views across the distributed system. The following image illustrates one of these out-of-synchronization moments:

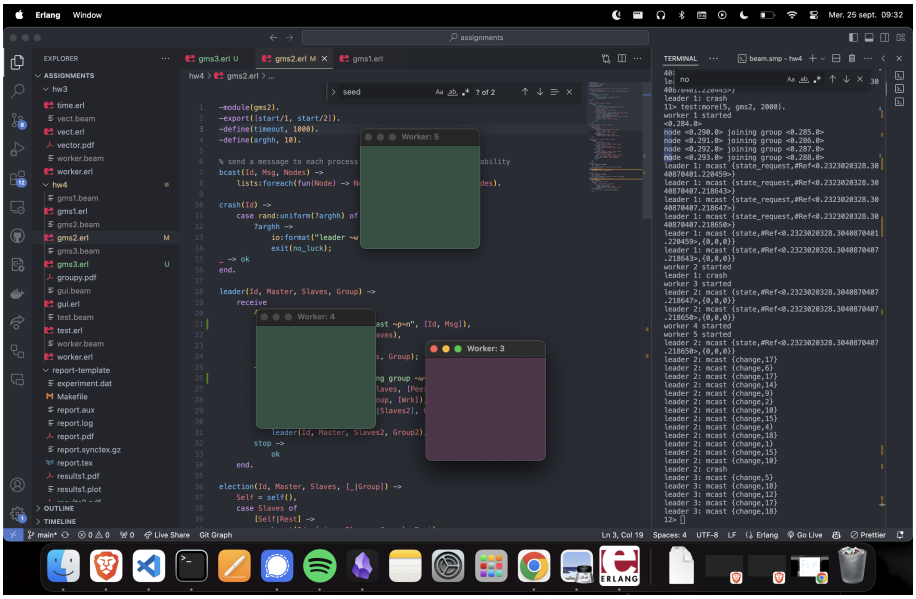

Figure 1: Out-of-synchronization situation in gms2

To fully understand how the test cases were structured and how to set up the tests properly, I had to go through the `test.erl` code. This allowed me to ensure that my setup was correct, and it helped me grasp the behavior of the system under various conditions. Properly setting up these tests was crucial for evaluating how the group membership service handles failures and recovers from them, especially in *gms2*, where crashes occurred.

4 Conclusions

This seminar provided valuable insights into the complexities of distributed systems, particularly in relation to group membership protocols and synchronization. Implementing the system and handling both normal operations and failure scenarios highlighted the importance of proper synchro-

nization to maintain consistency. By working through *gms1* and *gms2*, I was able to observe how crashes impact synchronization and message delivery, emphasizing the need for robust failure recovery mechanisms in distributed environments. Overall, this exercise deepened my understanding of how distributed systems handle coordination and resilience in the face of failures.