Distributed Systems - Project Report

Diego Oniarti - Alessandra Dalla Verde

Year 2024-2025

Contents

1 Design Choices **1**
1.1 System Layout 1
1.2 Messages 1
1.3 Round system 2
1.4 Testing though Simulation 2

2 Assumptions and Requirements **2**

3 Implementation Features **2**

1 Design Choices

1.1 System Layout

The main classes of our system are the Nodes, the Clients, and the Coordinator.
The nodes and clients serve their roles as described in the project's assignment, while the coordinator is tasked to issue commands and enforce the project's assumptions.

To keep track of ongoing operations we implemented a "rounds system" in which the coordinator issues a group of operations, waits for their conclusion, and repeats the process.

1.2 Messages

The system's components communicate through messages, divided into two main categories: control messages and service messages.

Control messages are exchanged instantaneously between the coordinator and the other actors to issue commands (requests and management operations) and to get their results. With this information the coordinator can guarantee that nodes join and leave, crash and recover one at a time, and only when there are no ongoing operations.

Service messages implement the actual functionalities of the system. They simulate the messages that would be exchanged in a real world scenario, hence they are only sent between nodes and clients (not the coordinator). To make their behaviour more realistic they are sent with a random delay, mimicking the propagation delay of packets.[1]

In **Fig.1** we illustrate all the messages of the system. To reduce the number of messages we adopted the following strategies:

1. **Get** and **Set**: only the nodes responsible for the interested data item are contacted.
2. **Join:** the system only asks the joining node's clockwise neighbor for the data items that it is going to be responsible for, avoiding a number of unnecessary message exchanges.
   The latest version of these data items is then obtained through the Get operation. As stated above, reads already uses the least amount of required messages.
3. **Leave:** the leaving node must transfer its data items to their new responsible nodes. If multiple data items are sent to the same node, they are bundled in the same message.
4. **Crash:** When a node crashes it sends no messages to the other nodes, only setting an internal flag. This operation is triggered by a control message.
5. **Recovery:** when a node enters recovery, it sends a data request to the other nodes to get updated on the status of the system. These messages are only sent to the $N-1$[2] nodes in front and the $N-1$ nodes behind the recovering one. Only these nodes may have the desired data items.

1.3 Round system

There are two types of rounds: ones in which the system resolves client requests (read and write), and ones in which a node performs management operations (join/leave, crash/recover). After sending the initiation message for each operation the coordinator waits for their results and an additional time delay to let the system settle. Only then it will start a new round.

1.4 Testing though Simulation

We perform two kinds of test on the system, both performed by the AppDebug class. Both tests rely on a log file containing a sequence of events in their global order.

The events stored in the file are:

- node storage modification: whenever data items are written or deleted from a node's storage, an entry describing the change is written to the file.
- operations: whenever an operation succeeds or fails, it is registered to the file as a single atomic entry.

Sequential Consistency A validator reads the log file and keeps track of all the successful read operations, storing each one as a node in a graph

2 Assumptions and Requirements

3 Implementation Features

[1] The random delays *could* break FIFO, but we decided to ignore this as per the project's given assumptions

[2] N denotes the system's replication factor

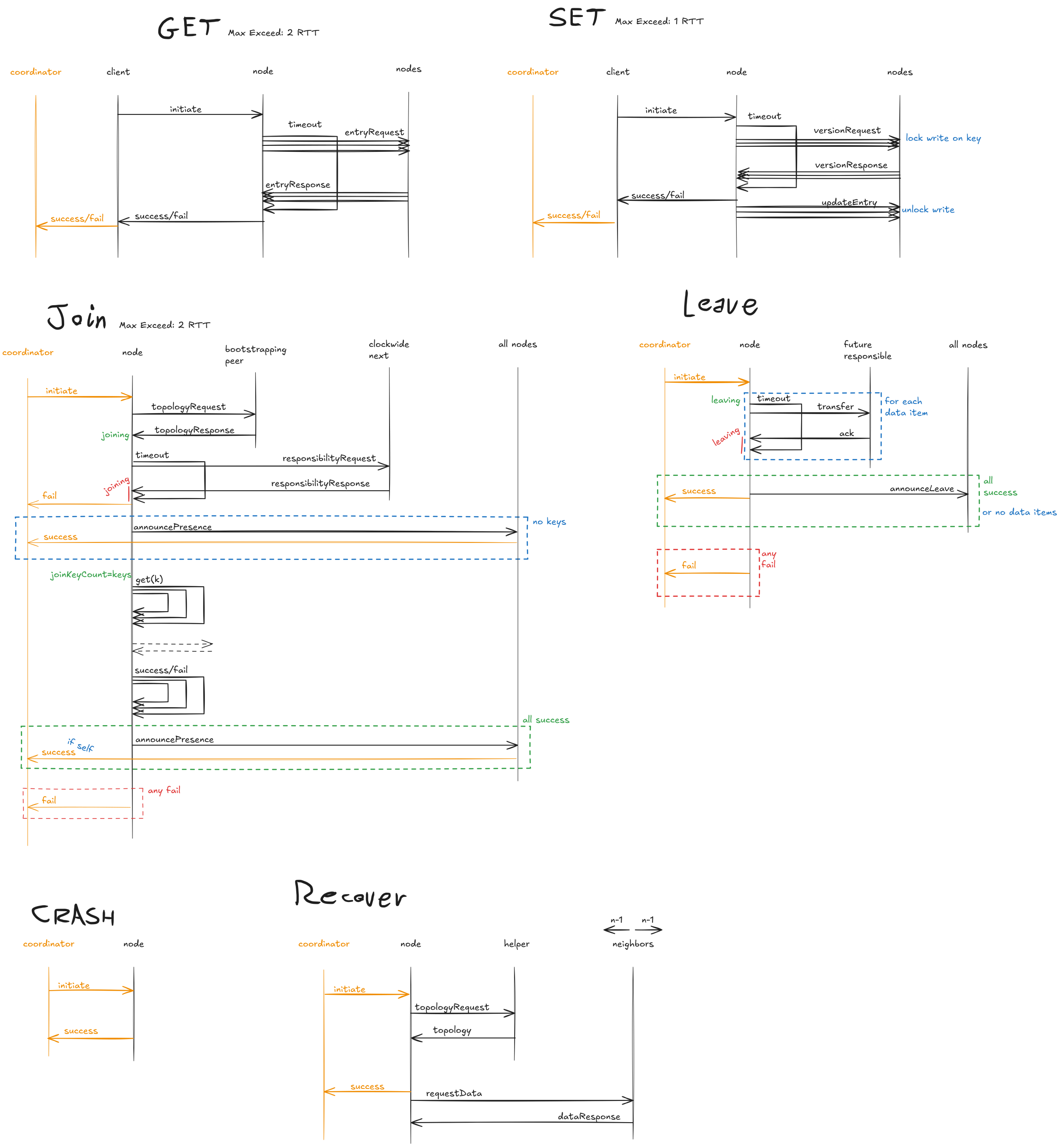

Figure 1: System messages