ID 2201 Report

Groupy: A Group Membership Service

Yilai Chen

Introduction

This project is about creating a group of processes that stay in sync even when some processes crash or new ones join. The goal is for all the processes to have the same state and make the same changes, no matter what happens(Process crash or change between old and new nodes). A process that wants to make a change sends the change to the whole group, and the group applies the change in the same order. If a process crashes, the group elects a new leader to keep things going smoothly.

Project Structure:

1. Leader:

The leader is the main process that controls the group. It receives messages from other nodes and multicasts them to the rest of the group. It also handles when new nodes want to join the group by updating the list of group members and broadcasting this new "view" to everyone. The leader makes sure every process stays in sync by controlling the order of messages.

2. Slave:

A slave is a process that forwards messages to the leader. If a slave receives a message from its application layer (like a request to make a change), it sends that message to the leader. It also listens to messages from the leader and passes them to the application layer. If the leader crashes, the slave takes part in the election to choose a new leader.

3. Election:

When the leader crashes, the election process starts. All the remaining slaves know the list of processes in the group and choose the first process in the list as the new leader. If a process finds out it’s the first in the list, it becomes the leader. The new leader resends the last message received to make sure no messages are lost during the change, and the slaves update their leader information.

Test

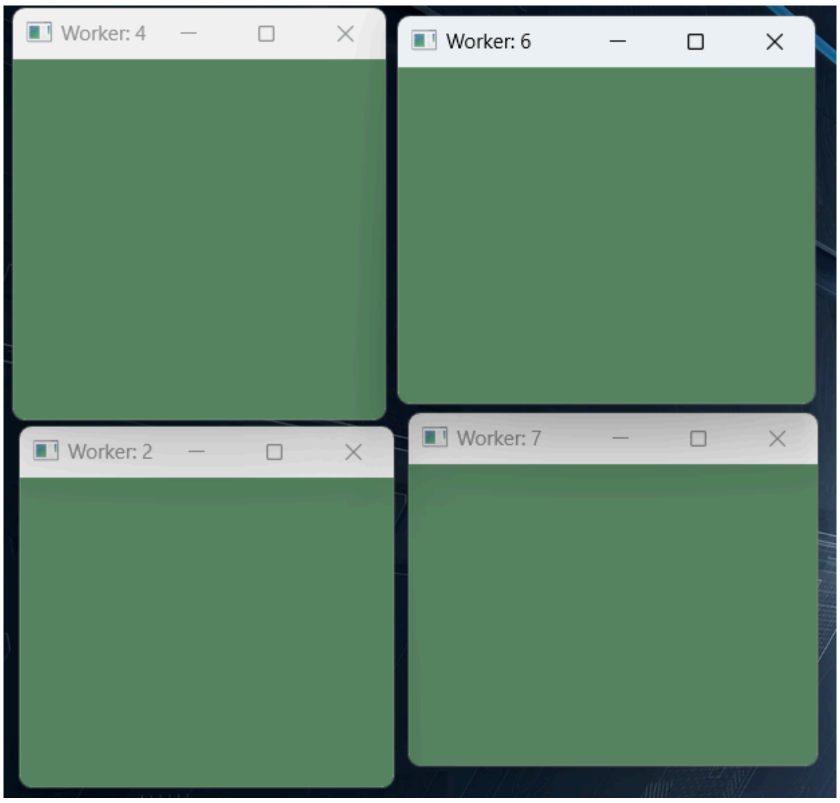
The test is designed to evaluate the behavior of the distributed system as nodes are added and removed, ensuring that the remaining nodes stay in sync throughout. The test involves creating an initial leader and two worker nodes, then dynamically adding and removing nodes, while observing how the system handles these changes.

Test Steps:

1. **Start with a leader (node 1) and two workers (node 2 and 3)**.
2. **Remove node 3**, leaving nodes 1 and 2.
3. **Add node 4**, resulting in nodes 1, 2, and 4.

4. **Add node 5**, creating a group of nodes 1, 2, 4, and 5.
5. **Remove node 5**, reducing the group to nodes 1, 2, and 4.
6. **Add node 6**, updating the group to nodes 1, 2, 4, and 6.
7. **Add node 7**, finalizing the group with nodes 1, 2, 4, 6, and 7.
8. **End the test**.

```erlang
start_test(Module, Sleep) ->
    io:format("Starting test with leader and two workers~n"),
    % Create Leader and Slave2, 3
    W1 = first(1, Module, Sleep),
    W2 = add(2, Module, W1, Sleep),
    W3 = add(3, Module, W1, Sleep),
    timer:sleep(5000),
    % Delete 3,  [1,2]
    io:format("Removing node 3~n"),
    stop(W3),
    timer:sleep(5000),
    % Append 4,  [1,2,4]
    io:format("Adding node 4~n"),
    W4 = add(4, Module, W2, Sleep),
    timer:sleep(5000),
    % Append 5,  [1,2,4,5]
    io:format("Adding node 5~n"),
    W5 = add(5, Module, W4, Sleep),
    timer:sleep(5000),
    % Delete 5,  [1,2,4]
    io:format("Removing node 5~n"),
    stop(W5),
    timer:sleep(5000),
    % Append 6, [1,2,4,6]
    io:format("Adding node 6~n"),
    W6 = add(6, Module, W4, Sleep),
    timer:sleep(5000),
    % Append 7, [1,2,4,6,7]
    io:format("Adding node 7~n"),
    W7 = add(7, Module, W6, Sleep),
    timer:sleep(10000),
    io:format("Test completed~n"),
    ok.
```


Worker: 4
Worker: 6
Worker: 2
Worker: 7