Semestet project DSVA

Problem type: Leader election

Algorithm: Bully

Language: Java RMI

Functionality: chat

Aleksandr Kross

krossale@fel.cvut.cz

Content

Introduction

Distributed systems are a vital part of modern technology, enabling multiple interconnected nodes to work together to complete tasks. One of the critical challenges in such systems is electing a leader node responsible for coordinating the operations of all other nodes. This project is a practical implementation of a distributed system that uses the Bully Algorithm for dynamic leader election.

The project is developed in Java and leverages Java RMI (Remote Method Invocation) for seamless inter-node communication. Additionally, a REST API is built using Javalin to allow users to manage nodes interactively. The combination of robust backend logic and user-friendly interfaces makes this system both effective and accessible.

Problem description

The project solves the challenge of leader election in a distributed system. Using the Bully Algorithm, nodes dynamically elect a leader based on their IDs, ensuring efficient coordination and fault tolerance.

Key Details:

1. **Problem Type:** Leader election in a distributed network.
2. **Algorithm**: Bully Algorithm, where higher-ID nodes are prioritized as leaders.
3. **Language**: Java with RMI for inter-node communication.
4. **Functionality**: A distributed chat system that enables messaging, leader management, and dynamic node operations.

This system handles node failures, leader recovery, and dynamic membership, making it reliable and user-friendly for real-world applications.

Application description

This project implements a distributed system with the following components:

1. **Leader Election using the Bully Algorithm**:
    - Each node is assigned a unique identifier (ID).
    - A node initiates an election when it detects the leader is unavailable.
    - Nodes with higher IDs are prioritized as leaders.
    - If a node does not receive an acknowledgment from any higher ID node, it declares itself the leader.
2. **Java RMI for Communication**:
    - Nodes communicate using RMI for remote method invocation, allowing seamless message exchange.
3. **REST API for Management**:
    - The Javalin framework is used to provide a RESTful API for user interaction with nodes.
    - API endpoints allow actions like initiating elections, checking leader status, sending messages, and managing node state.
4. **Console-Based Interaction**:
    - A console interface is available for direct node management, supporting commands like startElection, checkLeader, and status.
5. **Fault Tolerance**:
    - Nodes detect the failure of neighbors or the leader and respond by initiating elections or removing unreachable nodes.
    - Nodes can gracefully leave the network and notify others.

Implementation

Key Components

1. **Node Class**:
    - Represents an individual node in the system.
    - Manages neighbors, communicates with other nodes, and handles leader election.
2. **BullyAlgorithm Class**:
    - Implements the Bully Algorithm logic for leader election.
    - Handles election initiation, OK messages, and leader announcements.
3. **MessageReceiver Class**:
    - Listens for incoming RMI calls from other nodes.
    - Handles operations like join requests, election messages, and leader notifications.
4. **CommunicationHub Class**:
    - Provides methods for sending messages between nodes using RMI.
    - Ensures reliable communication and message retries.
5. **APIHandler Class**:
    - Implements a REST API for interacting with nodes.
    - Provides endpoints for election management, message sending, and node state control.
6. **DSNeighbours Class**:
    - Maintains the list of neighbors for a node and the current leader's address.
7. **Address Class**:
    - Represents the network address of a node, including its ID, hostname, and port.
8. **NodeCommands Interface**:
    - Defines the remote methods that nodes can invoke on each other using RMI.

Testing

The project is tested using Bash scripts to automate deployment and testing across multiple virtual machines.

Workflow:

1. **Clone and build:**

   Download the project from GitHub and build it using Maven (mvn clean package).

2. **Detect virtual machine IP:**

   Use a script to find and configure IPs of active virtual machines.

3. **Start and connect nodes:**

   Start nodes with 01_start_nodes.sh and connect them using 02_connect_nodes.sh.

4. **Test scenarios:**

   Get Node Status: scenario_01_get_status.sh

   Send Messages: scenario_02_send_messages.sh

   Stop/Start Nodes: scenario_03_stop_start_rmi.sh

   Check Leader: scenario_04_check_leader.sh

   Kill/Revive Nodes: scenario_05_kill_and_revive.sh

   Leave Nodes: scenario_06_leave_nodes.sh

These scripts test key functionalities like leader election, node failures, and dynamic membership through the API, ensuring the system is reliable and efficient.

Conclusion

This project has been an incredible learning experience in designing and implementing distributed systems. I learned to create a fully functioning chat system powered by Java RMI and integrated it with a REST API using Javalin for simplified node management.

Testing the system involved writing multiple Bash scripts to automate deployment, building, and testing. By dynamically detecting IP addresses of virtual machines and configuring them in my scripts, I was able to seamlessly deploy and manage nodes across different environments.

Through this project, I have gained a deeper understanding of:

1. **Leader Election algorithms:** Implementing the Bully algorithm and understanding its practical application.
2. **Distributed system testing:** Automating tests for dynamic membership, leader elections, and fault tolerance using Bash.
3. **Building and running nodes dynamically:** Compiling the application, deploying it on multiple virtual machines, and managing their lifecycle.

The project showcases the practical implementation of fault tolerance, dynamic membership, and efficient coordination, and serves as a foundation for future applications in distributed systems like databases, schedulers, or chat systems.