Distributed Node System with Local Lomet Algorithm

Overview

This distributed system implements a modified version of the Lomet algorithm for resource management across multiple nodes. The system uses a star topology with a RabbitMQ cluster at its center for message routing and queue management.

System Architecture

Components

- **Nodes**: Java applications that can both request and provide resources
- **RabbitMQ Cluster**: Central message broker (2 instances for high availability)
- **Each node consists of two main parts**:
  - Node Component: Handles resource requests and management
  - Resource Component: Manages access to local resources

Topology

- Star topology with RabbitMQ cluster at the center
- All nodes connect directly to RabbitMQ cluster
- No direct node-to-node communication required (only REST for health checks)
- High availability through RabbitMQ cluster redundancy

Local Lomet Algorithm Implementation

Resource Management

1. **Preliminary Resource Requests**
   - Nodes must request resources before usage
   - Each request is added to the resource's rabbitmqqueue (FIFO)
   - Resources maintain own dependency graph (reduced to queue - since we are implementing local lomet algorithm)
   - Resources sends back updated queue to the node

2. **Queue-based Access Control**
   - Each resource maintains its own queue of requesting nodes
   - Nodes can only access a resource when they're first in its queue
   - Queues replace traditional dependency graphs for simplicity
   - Deadlock prevention through queue ordering

3. **Resource Usage Flow**:

```
Node                Resource
 |                     |
```

```
  |-- REQUEST_ACCESS -->|
  |<-- QUEUE_UPDATE ----|
  |                     |
  |-- ENTER (if first)  |
  |                     |
  |-- RELEASE_ACCESS -->|
  |<-- QUEUE_UPDATE ----| (to other nodes in the queue)
```

Node States

- **IDLE**: Ready to make new requests
- **WAITING_FOR_RESOURCES_QUEUES**: Waiting for queue information
- **READY_TO_ENTER**: Received all queue information
- **WORKING**: Currently using a resource
- **WAITING_IN_QUEUE_FOR_RESOURCE**: Waiting for resource access

Fault Tolerance

Node Failure Detection

- Health checks between nodes and resources
- Automatic queue cleanup on node failure
- Resource release on node disconnection

RabbitMQ Cluster Resilience

- Automatic failover between RabbitMQ instances
- Message persistence across broker restarts
- Automatic reconnection handling

Setup Instructions

1. **RabbitMQ Cluster Setup**: First run the VMs and see what IPs are assigned to them. Then run the following scripts on both VMs (you will need to copy the Erlang cookie from rabbit1 to rabbit2)

```
# On first server:
./rabbit1_setup.sh <rabbit1_ip> <rabbit2_ip>

# On second server:
./rabbit2_setup.sh <rabbit1_ip> <rabbit2_ip>
```

2. **Configuration**: Run the Node VMs and see what IPs are assigned to them. Edit `src/main/resources/config.yml`:

```
rabbitmq:
  hosts:
    - host: "<rabbit1_ip>"
```

```
      port: 5672
    - host: "<rabbit2_ip>"
      port: 5672
  username: "myuser"
  password: "mypassword"

nodes:
  "<node_ip>":
    id: "node1"
    port: 7070
    resource: true
    ip: "<node_ip>"
```

3. **Build and Run**:

```
# Build the project
mvn clean package

# Run the node
java -jar target/node-1.0-SNAPSHOT.jar
```

System Requirements

- Java 11 or higher
- Maven 3.6+
- RabbitMQ 3.8+
- Network connectivity between all nodes and RabbitMQ servers

Logging

- All logs are stored in the 'logs' directory
- Separate log files for node and resource components
- Console output for debugging