Leader Election and Safe log replication Implementation

1. Introduction

Consensus indicates that several servers agree on the same information, which is required for designing fault-tolerant distributed systems. **RAFT protocol** helps in achieving consensus. As per RAFT algorithm each node in a cluster can exist in three states, namely **Leader, Candidate and Follower**.

The below flow diagram explains the transition of nodes into different states.

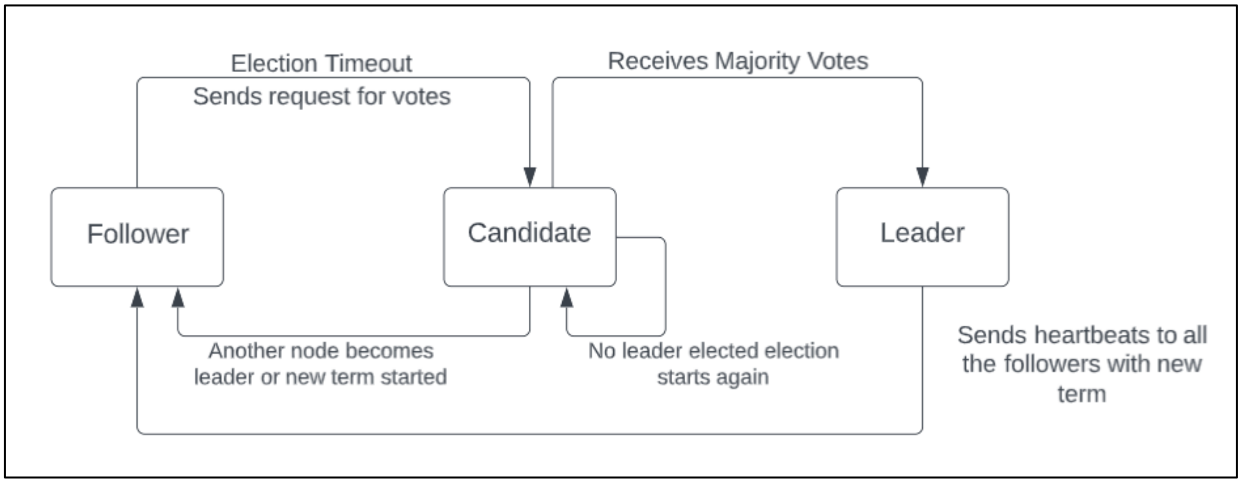

Election Timeout
Sends request for votes
Receives Majority Votes
Follower
Candidate
Leader
Another node becomes leader or new term started
No leader elected election starts again
Sends heartbeats to all the followers with new term

Fig 1.1 Flow diagram of states

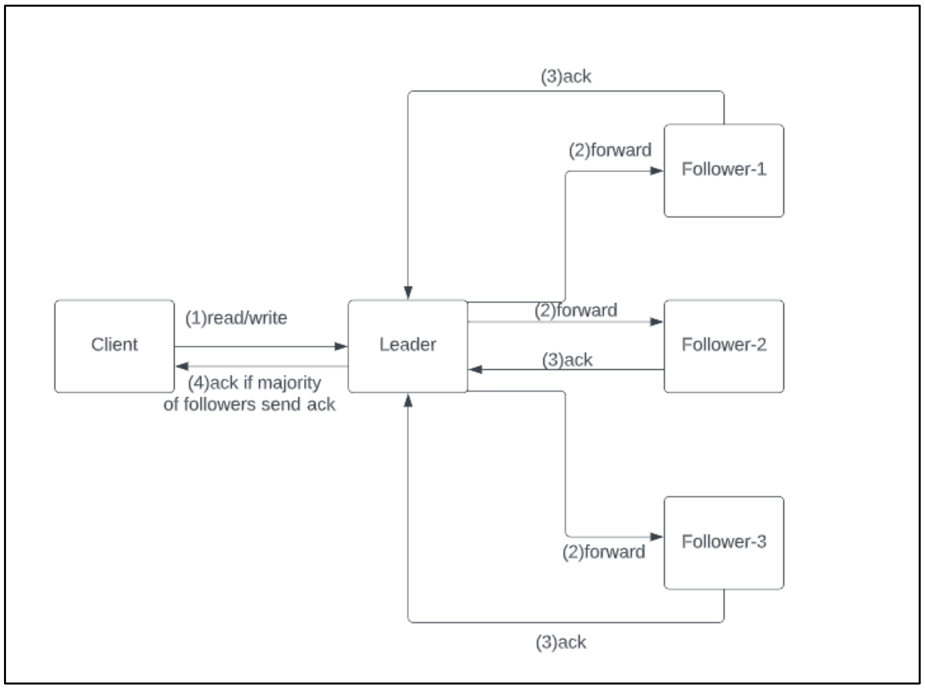
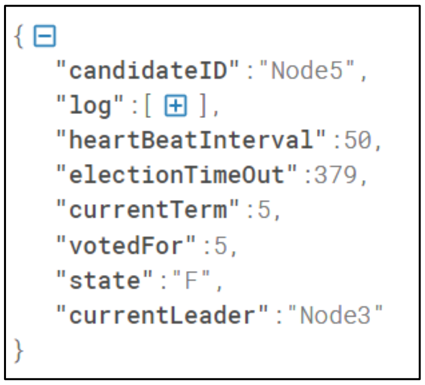


Fig 1.1 Flow diagram of LogReplication

2. Design Overview

2.1. Project Structure

- A **single** image including **client and server**
- Single image is used for creating **5 containers**
- **Client-side technology** – HTML/JSP
- **Server-side technology** – Spring(java)
- **Database** – Text File (**userInput.txt**)
- Each node information is saved in the filesystem of every node using json file. Below are the information which is saved for each node using “**nodeinfo.json**”.



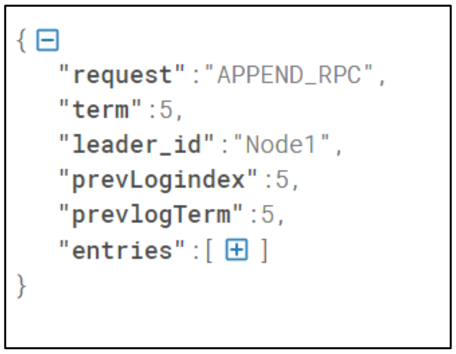
```
{ ⊟
   "candidateID":"Node5",
   "log":[ ⊞ ],
   "heartBeatInterval":50,
   "electionTimeOut":379,
   "currentTerm":5,
   "votedFor":5,
   "state":"F",
   "currentLeader":"Node3"
}
```

2.2. Architecture Details and Configurations

a. **Heartbeat –** Heartbeat which is AppendRPC sent by the leader to all the follower. The Heartbeat rate is fixed as **100 milliseconds** for each node.

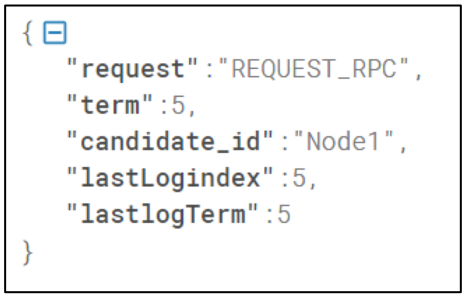
b. **Election Timeout –** The timeout after which the follower becomes candidate if it does not receive heartbeat is termed as election timeout. The election timeout is getting **randomly** calculated for each node between the range **300-600 milliseconds.**

c. **AppendRPC** – Hearbeat is sent as AppendRPC to each and every follower at a hearbeat interval. The following json message is sent as heartbeat in my implementation. The entries will be null if there is no log to append. Entries is the array of logs.



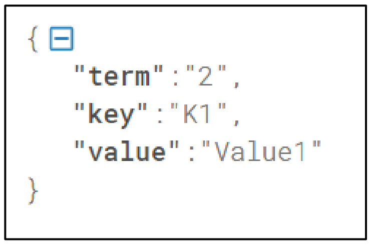
```
{ ⊟
   "request":"APPEND_RPC",
   "term":5,
   "leader_id":"Node1",
   "prevLogindex":5,
   "prevlogTerm":5,
   "entries":[ ⊞ ]
}
```

d. **RequestRPC** – A follower after getting timed out it is converted into candidate and send RequestRPC to get votes to become leader. The following json message is sent as RequestRPC in my implementation.



```
{
  "request":"REQUEST_RPC",
  "term":5,
  "candidate_id":"Node1",
  "lastLogindex":5,
  "lastlogTerm":5
}
```

e. **Log Structure –** A log will contain term, key and value.



```
{
  "term":"2",
  "key":"K1",
  "value":"Value1"
}
```

f. **Threads** – I am using mutiple threads in my implementation to ensure that UDP packets should not get lost and all the communication happens asynchronously. Following threads are used:

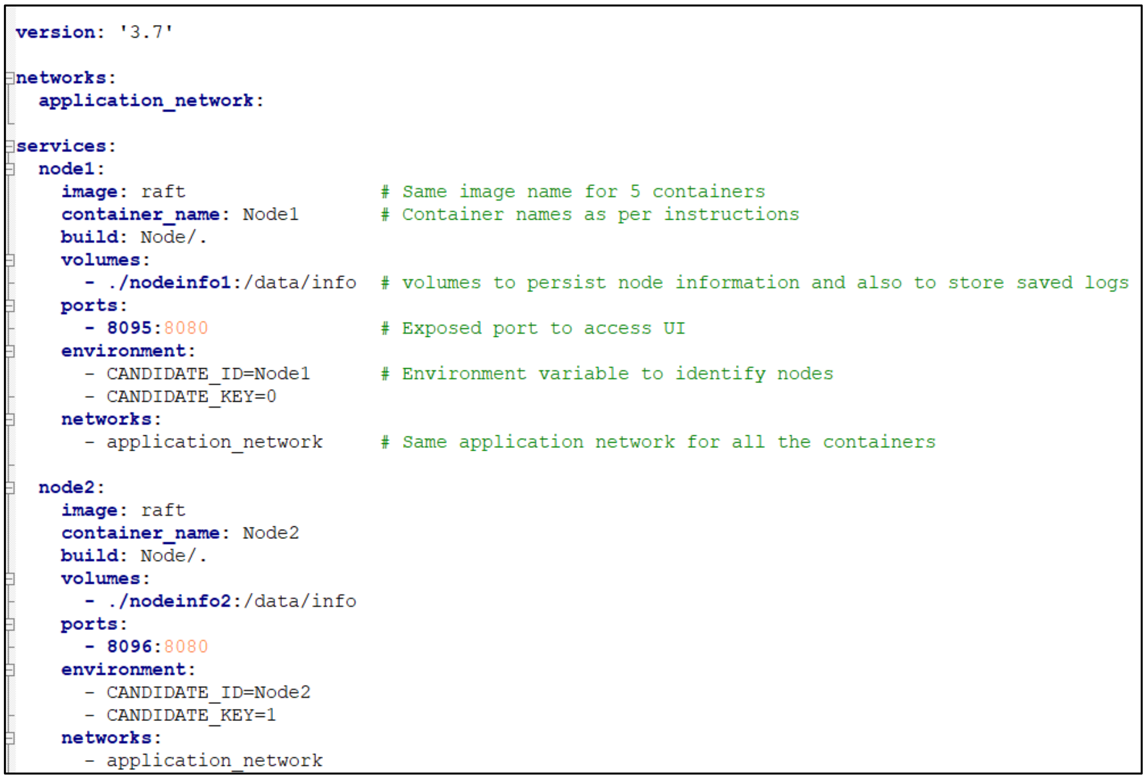
1. **Controller thread** – Listen to the controller request sent by the controller for testing
2. **Listen Thread** – Listen to the AppendRPC and RequestRPC
3. **Election Thread** – Used to send and receive votes once a follower becomes candidate. Vote request is multicasted to all nodes and then the election thread will wait till election timeout for response.
4. **SendHeartbeats** – This thread is used to send Append RPC to follower from the leader
5. **AppendReplyReceive** – This thread opens when a follower becomes Leader and it is used to receive the acknowledgement from the followers when a request is forwarded

Ports – I am using two ports in my implementation

1. **Port 5555** – It is used for communicating with the controller
2. **Port 1234** – It is used for communication within the nodes for sending UDP packets as RPC.

3. Implementation

1. Using **tomcat image** as a base image for RAFT and deployed WAR- Web Application Archive inside the tomcat
2. Created a **Raft** image and **Controller** image
3. All the containers are in the same network bridge.
4. Created a Docker file (dockerfile) which contains the detail of base image, starting command and copying WAR inside the working directory
5. Created docker compose file to create five different containers from the RAFT image and a single container for the Controller image.



```yaml
version: '3.7'

networks:
  application_network:

services:
  node1:
    image: raft                   # Same image name for 5 containers
    container_name: Node1         # Container names as per instructions
    build: Node/.
    volumes:
      - ./nodeinfo1:/data/info    # volumes to persist node information and also to store saved logs
    ports:
      - 8095:8080                 # Exposed port to access UI
    environment:
      - CANDIDATE_ID=Node1        # Environment variable to identify nodes
      - CANDIDATE_KEY=0
    networks:
      - application_network       # Same application network for all the containers

  node2:
    image: raft
    container_name: Node2
    build: Node/.
    volumes:
      - ./nodeinfo2:/data/info
    ports:
      - 8096:8080
    environment:
      - CANDIDATE_ID=Node2
      - CANDIDATE_KEY=1
    networks:
      - application_network
```

Fig-2.3: Docker Compose File showing entries for 2 nodes

6. Used docker compose to build a network bridge and started 5 containers out of the Raft image and one container out of Controller image created at step 3

Execution Steps:

1. Go to the directory where docker-compose file is placed
2. Run $ **docker compose build**
3. Run $ **docker compose up**
4. Wait for some time to allow system to become stable and leader got selected