CSE 586
DISTRIBUTED SYSTEMS
PROJECT PHASE 3

Authors: Sarveshwar Singhal(50418642) | UBIT - sarveshw
Abhishek Thosar (50394602) | UBIT - athosar

Date of submission: April 8, 2022

Introduction:

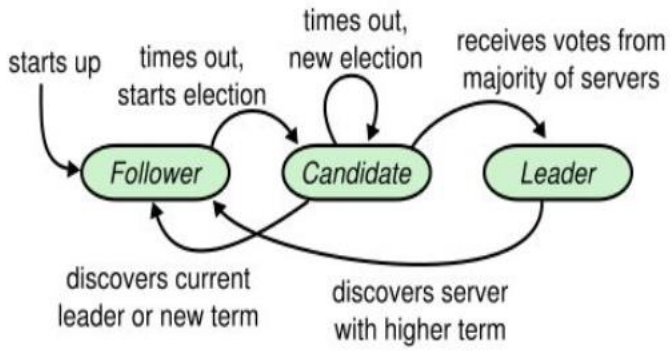
The phase 3 of the project deals with understanding and learning RAFT protocol, socket programming, server-side programming. We learned socket programming, multi-threading, asynchronous call in python to perform the Leader election using the RAFT protocol. Multiple Nodes/servers were created in Docker along with controller to create a distributed system. All the nodes are identical with respect to code.

Initially when the system starts every node is a candidate and when the timeout happens for any one of the node election starts. Post that a leader is elected. Once that leader is elected it periodically sends heartbeats to all other nodes to maintain it's leadership. In the network if any nodes fail or status of a node changes (through controller) appropriate action is triggered automatically. In case if a leader node is shutdown elections are called or appropriate action is taken based on the controller command.

The main motivation came from looking at the services of big tech giants, their systems never went down, so they must be following distributed systems approach to have data redundancy in case if a server fails, they have other servers to cater the user requests with persistent data.

Design Overview:

The below image is the design overview of our Leader election system implementing RAFT algorithm.



Implementation:

As part of the project, we created 5 nodes and 1 controller node in the docker container. To perform leader election within the nodes we implemented RAFT algorithm. All the 5 nodes have identical code. All the operations are done on server side. Initially when the system starts (i.e. we start the docker container) all the nodes are candidates as there is no leader. As soon as one of the nodes timed out it request for elections and elections starts.

We have used python to code the server-side logic. Communication within the nodes is done with the help of web sockets. Server-side code used multi-threading to communicate within nodes asynchronously.

Classes/Methods:

1. RaftServer: This class returns a RAFT server instance.

    a. *get_election_timeout*: This method returns the election timeout.

    b. *socket_listener*: Thread to actively listen to socket communication.

    c. *become_candidate*: Method to change node state to candidate and start election process by timeout.

    d. *Become_leader*: After majority vote this method changes server state to leader.

    e. *send_heartbeat*: This method is used to send heartbeat to all other nodes in the distributed network.

    f. *send_append_entry*: This method is used to send append entry with data to all other nodes in our distributed system.

    g. *handle_vote_request*: This method is used to handle the voting procedure.

    h. *Handle_vote_request_response*: This method parses vote request response *granted or not granted*

    i. *Handle_append_rpc*: This method handles *heartbeat* signals from leader node

j. *convert_to_follower*: This method is used to convert a node to follower state.

k. Check_majority_votes: Verifies node majority nodes

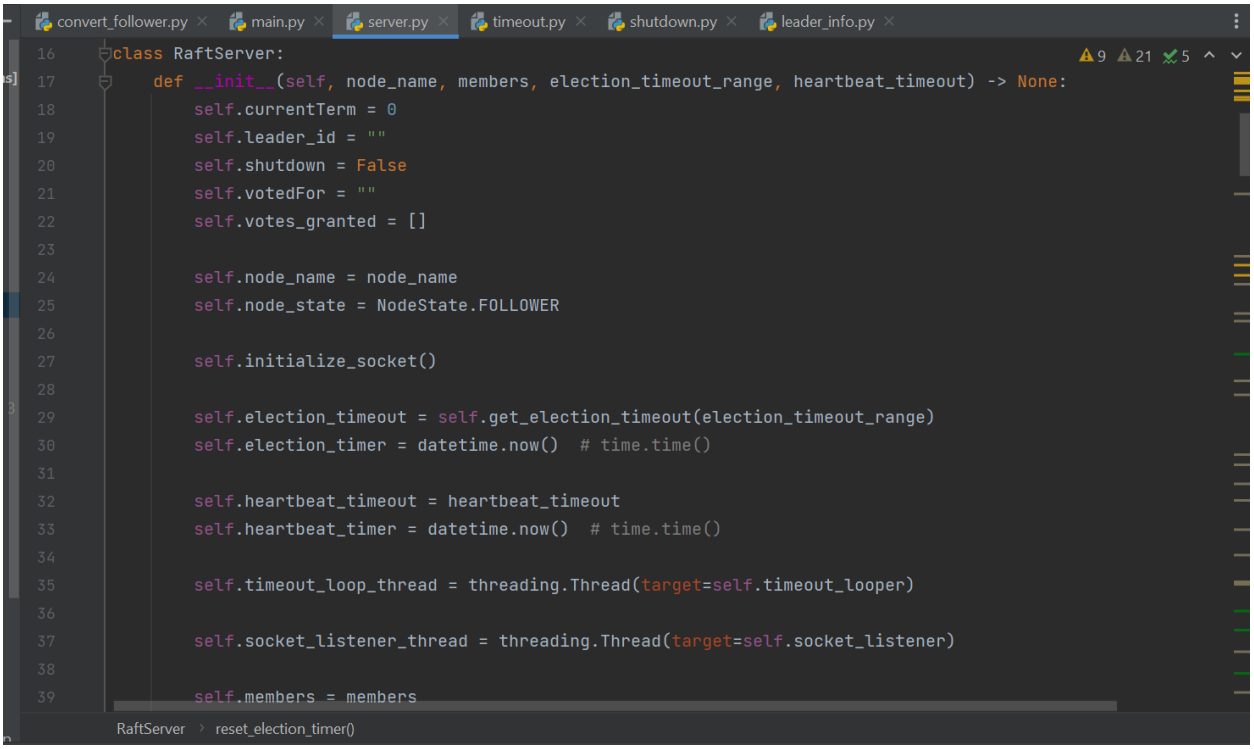
l. Set_election_term: Updates node's current term

m. reset_election_timer: Reset election timer after starting election timeout

n. Reset_heartbeat_timer: Resets heartbeat timer clock.

2. Controller: This method is used to test the implementation of Leader Election.

Validations/Screenshots:



The above image shows the initialization of RAFT server class, where all the variable like heartbeat_time, node_state, election_timeout etc. are set.

```python
def socket_listener(self):
    while True:
        try:
            msg, addr = self.socket.recvfrom(1024)
        except:
            print(f"NO DATA TO READ .....................")

        if not msg:
            continue

        # Decoding the Message received from nodes
        decoded_msg = json.loads(msg.decode('utf-8'))

        # Check for Controller request
        contReqType = decoded_msg['request']
        if contReqType == 'CONVERT_FOLLOWER':
            self.convert_to_follower()
            continue
        if contReqType == 'TIMEOUT':
            self.become_candidate()
            continue
        if contReqType == 'SHUTDOWN':
            self.shutdown = True
```

The above image shows the socket_listener method which is used to listen all the incoming request on socket.

```python
def become_candidate(self):
    self.node_state = NodeState.CANDIDATE
    self.reset_election_timer()
    # increase the term
    self.currentTerm += 1
    self.votedFor = self.node_name
    self.votes_granted.append(self.node_name)

    # Get the votes for each member

    for member in self.members:
        # request vote
        messageObj = RequestVoteMessage(self.node_name, RequestType.VOTE_REQUEST, self.currentTerm, self.
        msg_bytes = json.dumps(messageObj.__dict__).encode()
        print(f"VOTE REQUEST :: SENDING TO :: {member}")
        try:
            self.socket.sendto(msg_bytes, (member, 5555))
        except:
            if member in self.members:
                self.members.remove(member)
            print(f"VOTE REQUEST ERROR:: SENDING TO :: {member}")
            continue

    self.check_majority_votes()
```

The above image shows become_candidate method which is used to send vote request to other candidates (i.e. nodes)

```
127
128     def become_leader(self):
129         self.node_state = NodeState.LEADER
130         self.reset_heartbeat_timer()
131         # send each node update about becoming leader
132         # send appendentries
133         self.send_append_entry()
134
135     def send_heartbeat(self):
136         self.reset_heartbeat_timer()
137         # send append entries request
138         self.send_append_entry()
139
140     def send_append_entry(self):
141         for member in self.members:
142             messageObj = AppendRPCMessage(self.node_name, RequestType.APPEND_RPC, self.currentTerm, self.node_
143             msg_bytes = json.dumps(messageObj.__dict__).encode()
144             try:
145                 self.socket.sendto(msg_bytes, (member, 5555))
146             except:
147                 if member in self.members:
148                     self.members.remove(member)
149                 print(f"send_append_entry ERROR:: SENDING TO :: {member}")
150                 continue
```

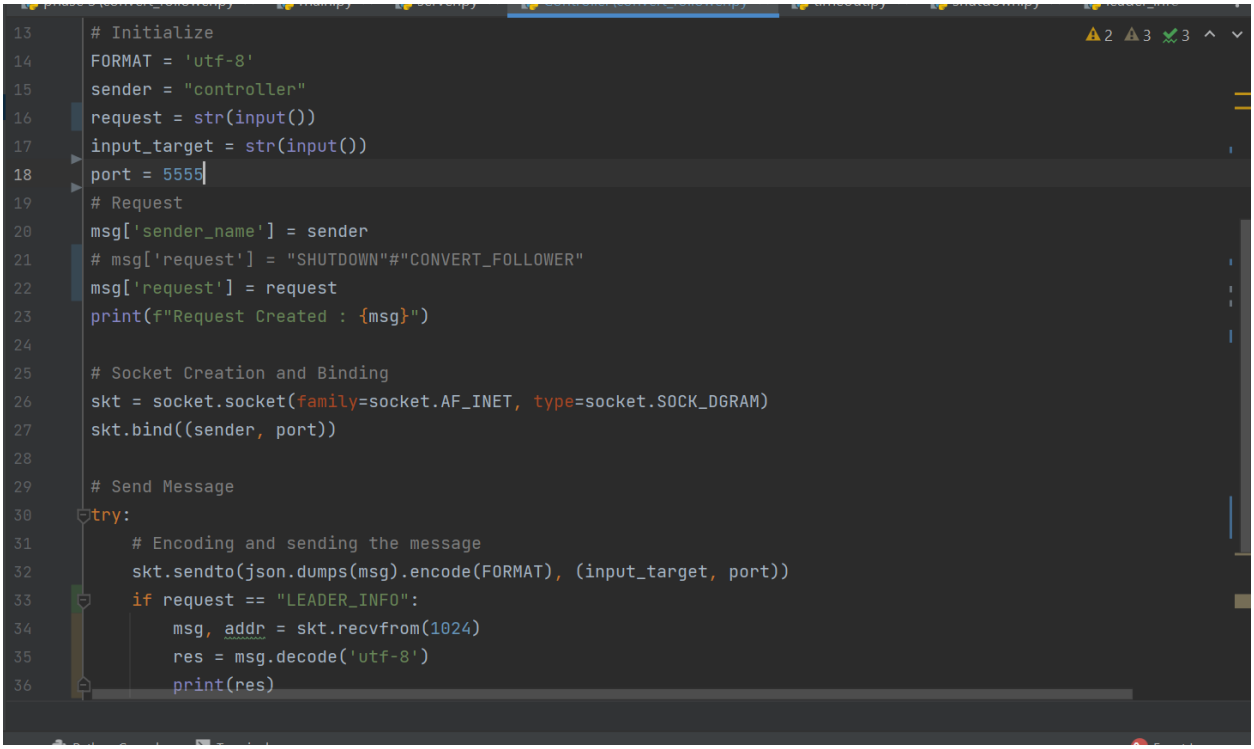
RaftServer › reset_election_timer()

The image shows various methods like become_leader, send_heartbeat, send_append_entry. These methods are used change a node to state to leader, send heartbeat to other nodes and send data(append) entry to all other nodes.

```
152     def initialize_socket(self):
153         self.socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
154         # Bind the node to sender ip and port
155         self.socket.bind((self.node_name, 5555))
156
157     def handle_vote_request(self, msg):
158         # obj = json.loads(msg)
159         requestObj = RequestVoteMessage(**msg)
160         # parse the message
161         self.set_election_term(requestObj.term)
162         # Check if node term is less than or greater than received term
163         granted = self.currentTerm <= requestObj.term
164
165         print(f"self.currentTerm < requestObj.term :: {self.currentTerm} < {requestObj.term}")
166
167         if granted:  # request granted
168             self.votedFor = requestObj.candidate_id
169
170         messageObj = RequestVoteResponse(self.node_name, RequestType.VOTE_ACK, self.currentTerm, granted)
171         msg_bytes = json.dumps(messageObj.__dict__).encode()
172         self.socket.sendto(msg_bytes, (requestObj.sender_name, 5555))
173
174         # SEND:-> voting request response as granted
```

RaftServer › reset_election_timer()

The above image shows handle_vote_request method which is used to check the voting condition for a node. If a node will vote in favor or in against.



```python
# Initialize
FORMAT = 'utf-8'
sender = "controller"
request = str(input())
input_target = str(input())
port = 5555
# Request
msg['sender_name'] = sender
# msg['request'] = "SHUTDOWN"#"CONVERT_FOLLOWER"
msg['request'] = request
print(f"Request Created : {msg}")

# Socket Creation and Binding
skt = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
skt.bind((sender, port))

# Send Message
try:
    # Encoding and sending the message
    skt.sendto(json.dumps(msg).encode(FORMAT), (input_target, port))
    if request == "LEADER_INFO":
        msg, addr = skt.recvfrom(1024)
        res = msg.decode('utf-8')
        print(res)
```

The above image shows the implementation of controller.

References:

https://raft.github.io/
https://raft.github.io/raft.pdf
http://thesecretlivesofdata.com/raft/ (Highly recommended)
https://www.youtube.com/watch?v=YbZ3zDzDnrw
https://docs.python.org/3/library/socket.html
https://docs.python.org/3/library/threading.html