Lab2 Report

Naseem Machlovi

The cluster needs to simulate the following items.

1. Create a file on three nodes.

2. The client can submit MULTIPLE values, one at a time, to the system and the values will be stored in the log file distributedly.

    a. For example, the client can call the RPC, SubmitValue, such as 1, on one of the nodes.

    b. The system needs to write one line into the file, such as write 1.

    c. When the system ensures consistency, it sends back a success message to the client.

3. Simulate the basic RAFT as discussed during lecture 8.

How to run:

git clone https://github.com/machlovi/distributed-system.git

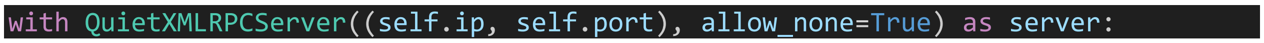
The script is ready to run on GCP, given you must update the ip address in both node and client code as shown below.

In order to run it locally, uncomment the local host form both node and client.py and uncomment the following line in run_server function in node.py file

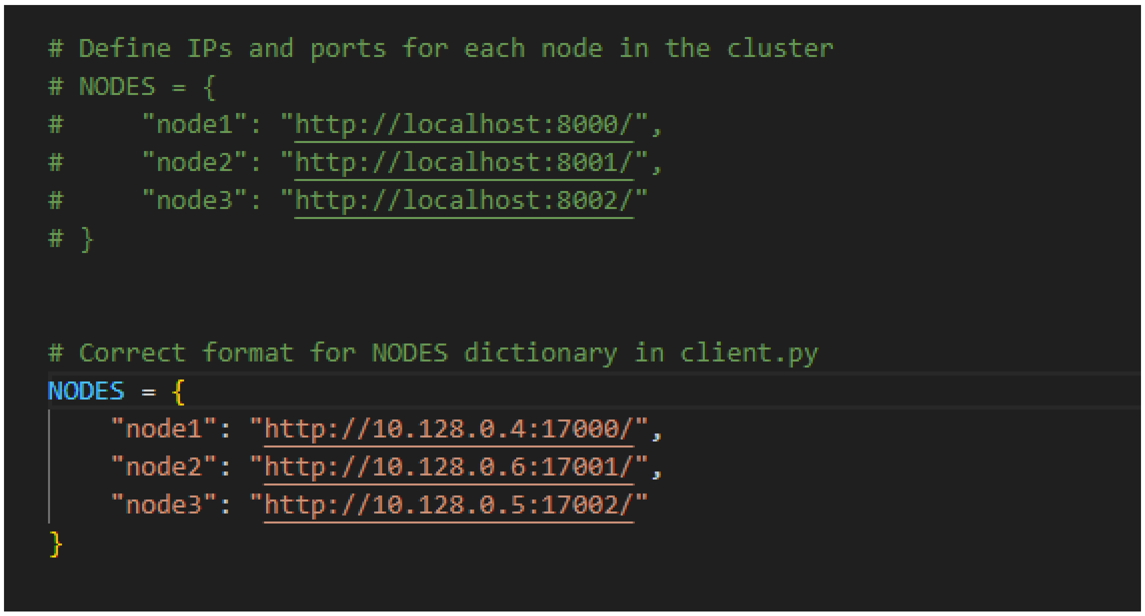
Finally, to tun the code:

python3 node.py node1
python3 node.py node2
python3 node.py node3

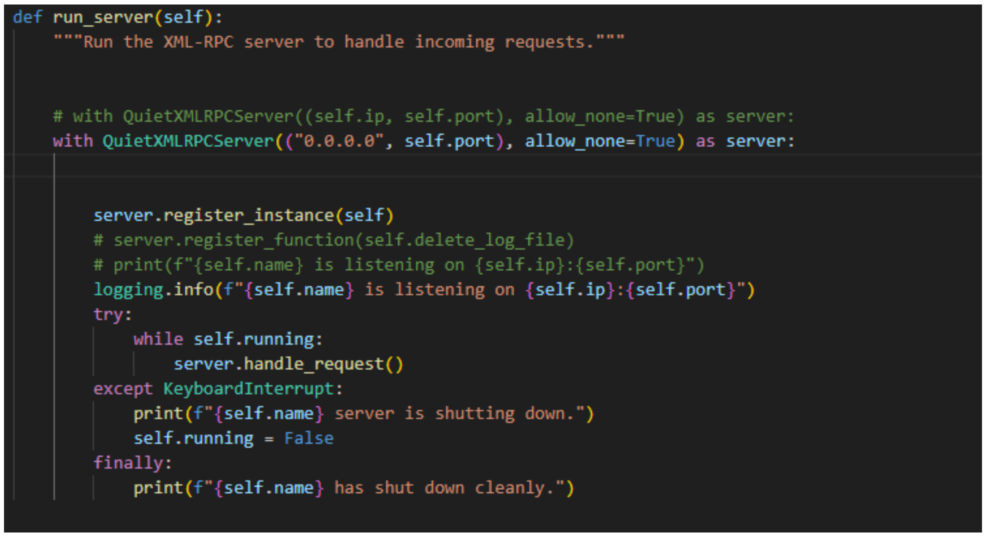
python3 client.py



```
with QuietXMLRPCServer((self.ip, self.port), allow_none=True) as server:
```

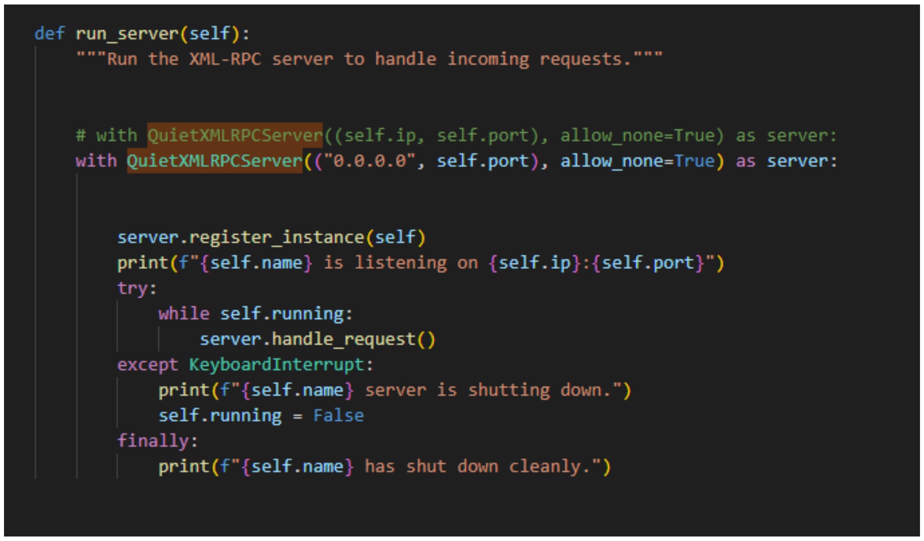


```
# Define IPs and ports for each node in the cluster
# NODES = {
#     "node1": "http://localhost:8000/",
#     "node2": "http://localhost:8001/",
#     "node3": "http://localhost:8002/"
# }


# Correct format for NODES dictionary in client.py
NODES = {
    "node1": "http://10.128.0.4:17000/",
    "node2": "http://10.128.0.6:17001/",
    "node3": "http://10.128.0.5:17002/"
}
```



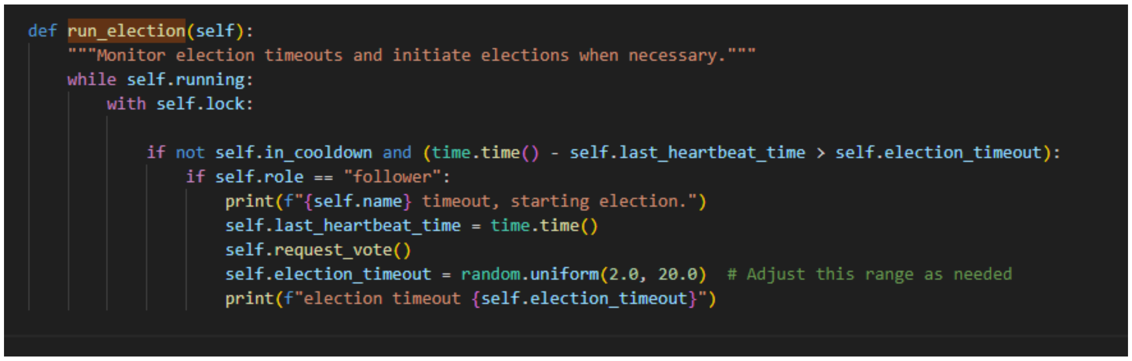
```python
def run_server(self):
    """Run the XML-RPC server to handle incoming requests."""


    # with QuietXMLRPCServer((self.ip, self.port), allow_none=True) as server:
    with QuietXMLRPCServer(("0.0.0.0", self.port), allow_none=True) as server:


        server.register_instance(self)
        # server.register_function(self.delete_log_file)
        # print(f"{self.name} is listening on {self.ip}:{self.port}")
        logging.info(f"{self.name} is listening on {self.ip}:{self.port}")
        try:
            while self.running:
                server.handle_request()
        except KeyboardInterrupt:
            print(f"{self.name} server is shutting down.")
            self.running = False
        finally:
            print(f"{self.name} has shut down cleanly.")
```

In this lab, we have implemented Raft consensus for 3 node cluster mechanism based on RPC protocol. The final implementation has been done GCP cloud using XMLRPC. For the RAFT implementation, we have followed all the protocols as mentioned above and achieved the targeted results. To start the server and they can listen other nodes on designated ports, we have implemented run server function which will be run as a thread in for each node. Each node, has ip address for all the other nodes, in order to process the raft consensus and other mandatory functions such as heart beat, voting and append logs.

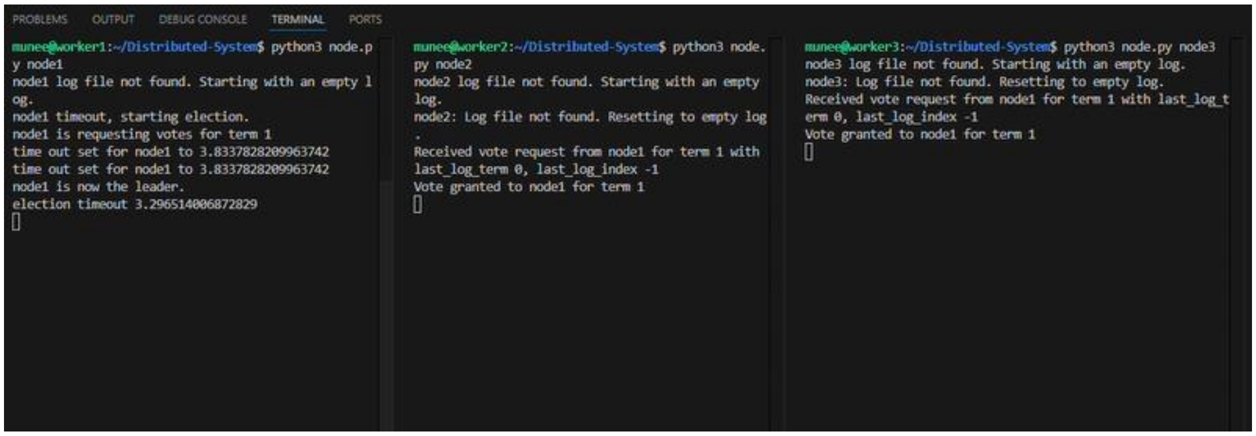


```python
def run_server(self):
    """Run the XML-RPC server to handle incoming requests."""


    # with QuietXMLRPCServer((self.ip, self.port), allow_none=True) as server:
    with QuietXMLRPCServer(("0.0.0.0", self.port), allow_none=True) as server:


        server.register_instance(self)
        print(f"{self.name} is listening on {self.ip}:{self.port}")
        try:
            while self.running:
                server.handle_request()
        except KeyboardInterrupt:
            print(f"{self.name} server is shutting down.")
            self.running = False
        finally:
            print(f"{self.name} has shut down cleanly.")
```

To start the RAFT consensus, we start the election with election function as shown below, where we have called the request vote function, which will ask each client node to vote for the given candidate and update the votes, which will in return start the leader. The leader will start sending the heart beat to each of the follower and incase the follower doesnot receive the heart beat before its timeout, it will start its election and ignore the heartbeat from the lower term leader otherwise we will have the leader working. The implementation has shown below.



```python
def run_election(self):
    """Monitor election timeouts and initiate elections when necessary."""
    while self.running:
        with self.lock:

            if not self.in_cooldown and (time.time() - self.last_heartbeat_time > self.election_timeout):
                if self.role == "follower":
                    print(f"{self.name} timeout, starting election.")
                    self.last_heartbeat_time = time.time()
                    self.request_vote()
                    self.election_timeout = random.uniform(2.0, 20.0)  # Adjust this range as needed
                    print(f"election timeout {self.election_timeout}")
```

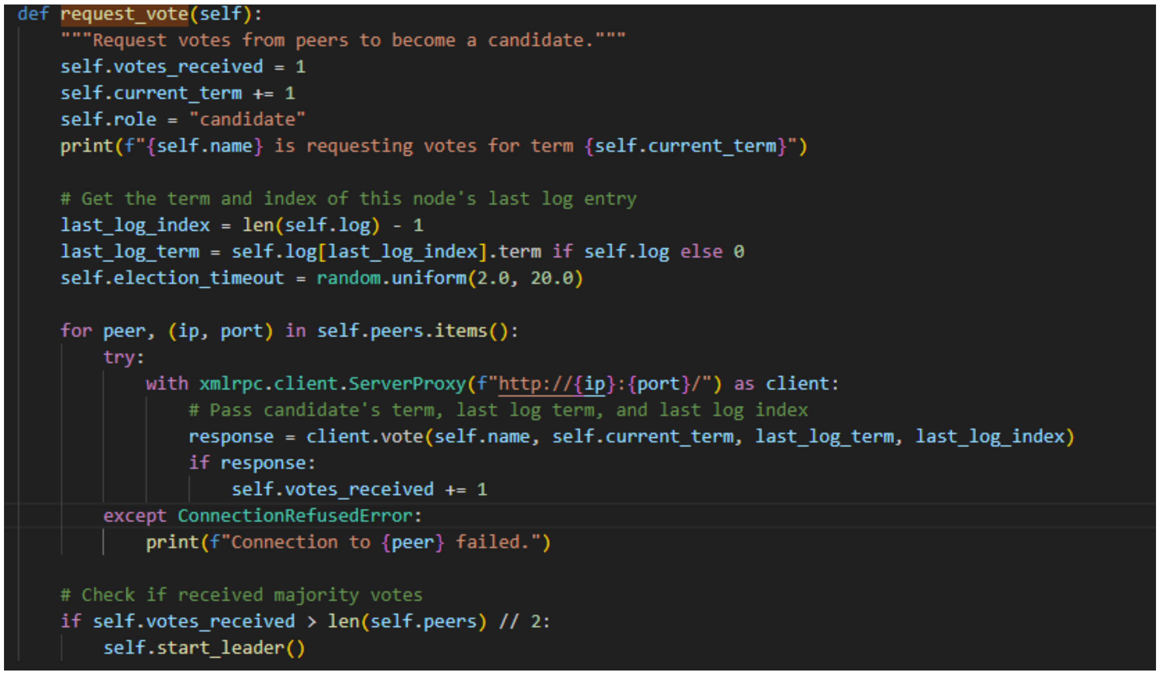


```
munee@worker1:~/Distributed-System$ python3 node.p
y node1
node1 log file not found. Starting with an empty l
og.
node1 timeout, starting election.
node1 is requesting votes for term 1
time out set for node1 to 3.8337828209963742
time out set for node1 to 3.8337828209963742
node1 is now the leader.
election timeout 3.296514006872829
```

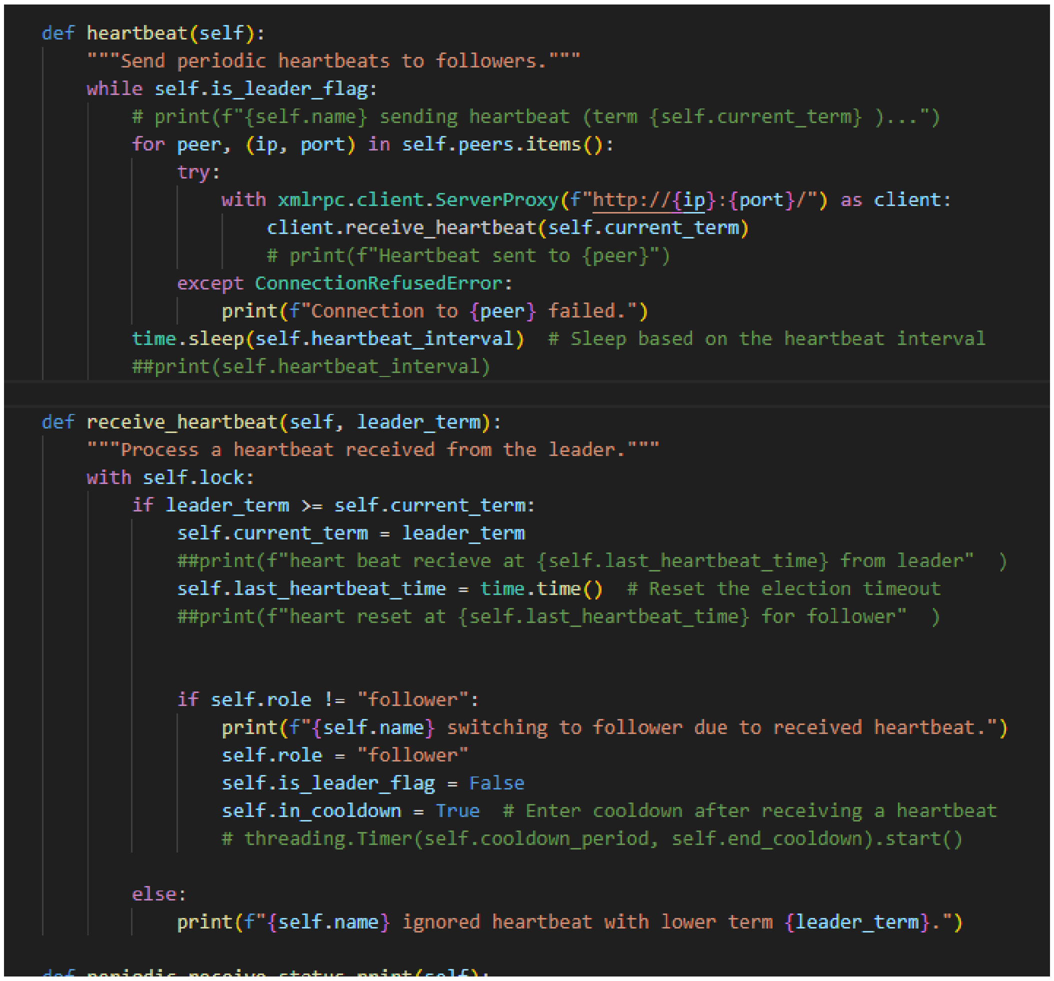
```
munee@worker2:~/Distributed-System$ python3 node.
py node2
node2 log file not found. Starting with an empty
log.
node2: Log file not found. Resetting to empty log
.
Received vote request from node1 for term 1 with
last_log_term 0, last_log_index -1
Vote granted to node1 for term 1
```

```
munee@worker3:~/Distributed-System$ python3 node.py node3
node3 log file not found. Starting with an empty log.
node3: Log file not found. Resetting to empty log.
Received vote request from node1 for term 1 with last_log_t
erm 0, last_log_index -1
Vote granted to node1 for term 1
```

Node is the leader for term1 but the timeout for node has been achievd before receiving any heartbeat from the leader term 1 and therefore a new election started, resulting in node2 as leader.



```python
def request_vote(self):
    """Request votes from peers to become a candidate."""
    self.votes_received = 1
    self.current_term += 1
    self.role = "candidate"
    print(f"{self.name} is requesting votes for term {self.current_term}")

    # Get the term and index of this node's last log entry
    last_log_index = len(self.log) - 1
    last_log_term = self.log[last_log_index].term if self.log else 0
    self.election_timeout = random.uniform(2.0, 20.0)

    for peer, (ip, port) in self.peers.items():
        try:
            with xmlrpc.client.ServerProxy(f"http://{ip}:{port}/") as client:
                # Pass candidate's term, last log term, and last log index
                response = client.vote(self.name, self.current_term, last_log_term, last_log_index)
                if response:
                    self.votes_received += 1
        except ConnectionRefusedError:
            print(f"Connection to {peer} failed.")

    # Check if received majority votes
    if self.votes_received > len(self.peers) // 2:
        self.start_leader()
```

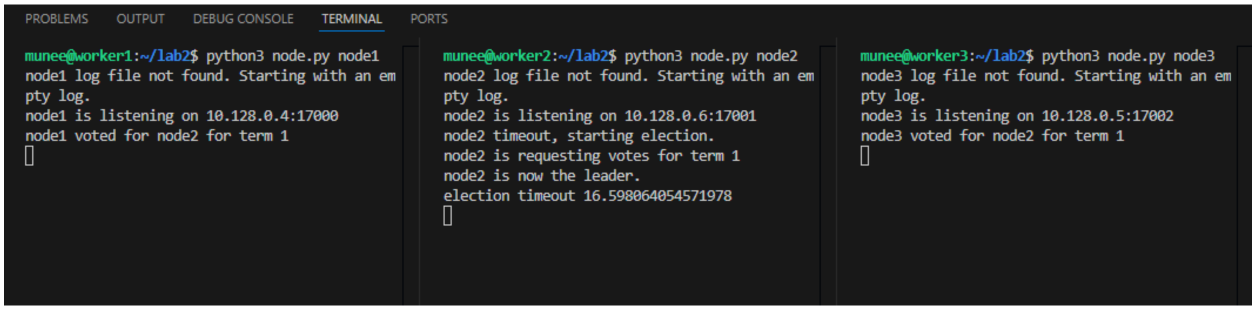


```python
def heartbeat(self):
    """Send periodic heartbeats to followers."""
    while self.is_leader_flag:
        # print(f"{self.name} sending heartbeat (term {self.current_term} )...")
        for peer, (ip, port) in self.peers.items():
            try:
                with xmlrpc.client.ServerProxy(f"http://{ip}:{port}/") as client:
                    client.receive_heartbeat(self.current_term)
                    # print(f"Heartbeat sent to {peer}")
            except ConnectionRefusedError:
                print(f"Connection to {peer} failed.")
        time.sleep(self.heartbeat_interval)  # Sleep based on the heartbeat interval
        ##print(self.heartbeat_interval)

def receive_heartbeat(self, leader_term):
    """Process a heartbeat received from the leader."""
    with self.lock:
        if leader_term >= self.current_term:
            self.current_term = leader_term
            ##print(f"heart beat recieve at {self.last_heartbeat_time} from leader"  )
            self.last_heartbeat_time = time.time()  # Reset the election timeout
            ##print(f"heart reset at {self.last_heartbeat_time} for follower"  )


            if self.role != "follower":
                print(f"{self.name} switching to follower due to received heartbeat.")
                self.role = "follower"
                self.is_leader_flag = False
                self.in_cooldown = True  # Enter cooldown after receiving a heartbeat
                # threading.Timer(self.cooldown_period, self.end_cooldown).start()

        else:
            print(f"{self.name} ignored heartbeat with lower term {leader_term}.")
```

The above code shows the heart beat implementation for each leader and followers

There are 3 scenarios to be simulated.

a. Leader election phase and everything goes well, NO crashes or slow followers.



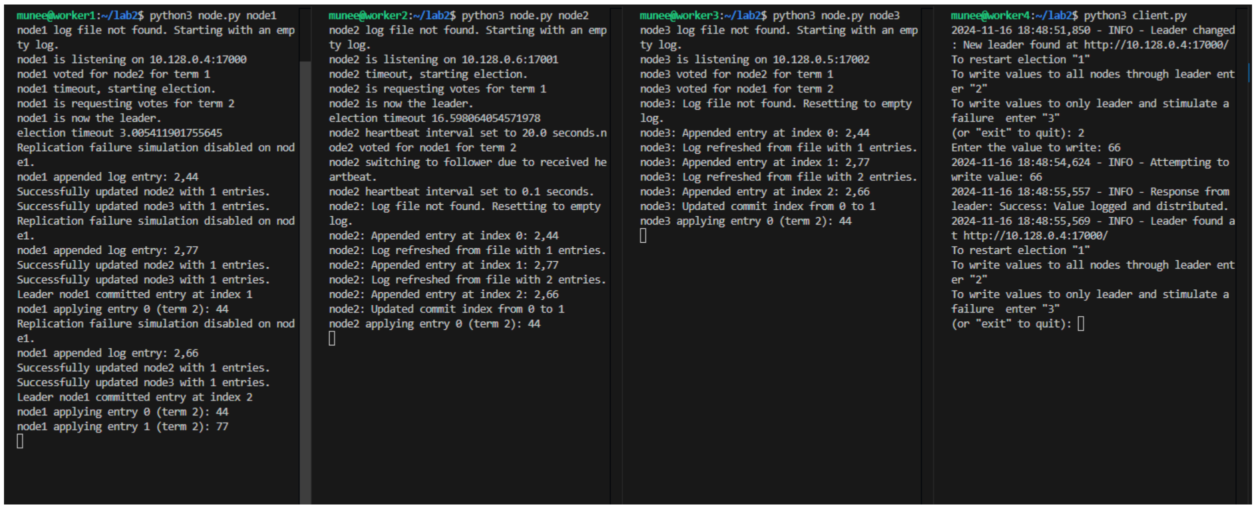
```
munee@worker1:~/lab2$ python3 node.py node1
node1 log file not found. Starting with an em
pty log.
node1 is listening on 10.128.0.4:17000
node1 voted for node2 for term 1
```

```
munee@worker2:~/lab2$ python3 node.py node2
node2 log file not found. Starting with an em
pty log.
node2 is listening on 10.128.0.6:17001
node2 timeout, starting election.
node2 is requesting votes for term 1
node2 is now the leader.
election timeout 16.598064054571978
```

```
munee@worker3:~/lab2$ python3 node.py node3
node3 log file not found. Starting with an em
pty log.
node3 is listening on 10.128.0.5:17002
node3 voted for node2 for term 1
```

b. Simulate leader change in a perfect situation, which means NO left entries partially replicated (slide-8 page 54). It's basically the same as item a with another round of leader election, but picking the best candidate. This is a simplified selection due to the perfect situation. i. The leader change process can start with the client, e.g., by typing a command

We write multiple values through client on leader, which will write them on each node and will reach to raft consensus if has max write achieved, which incase here is 2 node writes.



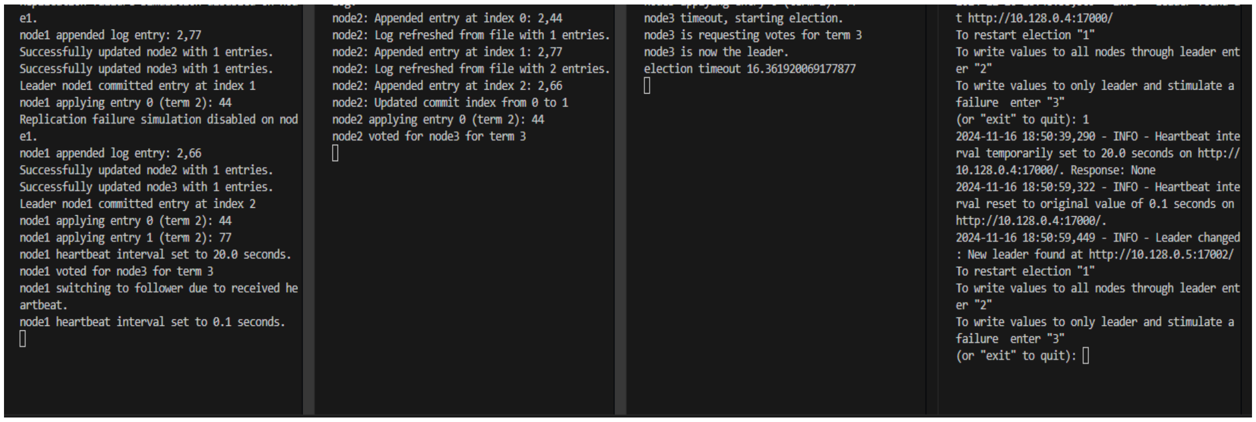
```
munee@worker1:~/lab2$ python3 node.py node1
node1 log file not found. Starting with an emp
ty log.
node1 is listening on 10.128.0.4:17000
node1 voted for node2 for term 1
node1 timeout, starting election.
node1 is requesting votes for term 2
node1 is now the leader.
election timeout 3.005411901755645
Replication failure simulation disabled on nod
e1.
node1 appended log entry: 2,44
Successfully updated node2 with 1 entries.
Successfully updated node3 with 1 entries.
Replication failure simulation disabled on nod
e1.
node1 appended log entry: 2,77
Successfully updated node2 with 1 entries.
Successfully updated node3 with 1 entries.
Leader node1 committed entry at index 1
node1 applying entry 0 (term 2): 44
Replication failure simulation disabled on nod
e1.
node1 appended log entry: 2,66
Successfully updated node2 with 1 entries.
Successfully updated node3 with 1 entries.
Leader node1 committed entry at index 2
node1 applying entry 0 (term 2): 44
node1 applying entry 1 (term 2): 77
```

```
munee@worker2:~/lab2$ python3 node.py node2
node2 log file not found. Starting with an emp
ty log.
node2 is listening on 10.128.0.6:17001
node2 timeout, starting election.
node2 is requesting votes for term 1
node2 is now the leader.
election timeout 16.598064054571978
node2 heartbeat interval set to 20.0 seconds.n
ode2 voted for node1 for term 2
node2 switching to follower due to received he
artbeat.
node2 heartbeat interval set to 0.1 seconds.
node2: Log file not found. Resetting to empty
log.
node2: Appended entry at index 0: 2,44
node2: Log refreshed from file with 1 entries.
node2: Appended entry at index 1: 2,77
node2: Log refreshed from file with 2 entries.
node2: Appended entry at index 2: 2,66
node2: Updated commit index from 0 to 1
node2 applying entry 0 (term 2): 44
```

```
munee@worker3:~/lab2$ python3 node.py node3
node3 log file not found. Starting with an emp
ty log.
node3 is listening on 10.128.0.5:17002
node3 voted for node2 for term 1
node3 voted for node1 for term 2
node3: Log file not found. Resetting to empty
log.
node3: Appended entry at index 0: 2,44
node3: Log refreshed from file with 1 entries.
node3: Appended entry at index 1: 2,77
node3: Log refreshed from file with 2 entries.
node3: Appended entry at index 2: 2,66
node3: Updated commit index from 0 to 1
node3 applying entry 0 (term 2): 44
```

```
munee@worker4:~/lab2$ python3 client.py
2024-11-16 18:48:51,850 - INFO - Leader changed
: New leader found at http://10.128.0.4:17000/
To restart election "1"
To write values to all nodes through leader ent
er "2"
To write values to only leader and stimulate a
failure  enter "3"
(or "exit" to quit): 2
Enter the value to write: 66
2024-11-16 18:48:54,624 - INFO - Attempting to
write value: 66
2024-11-16 18:48:55,557 - INFO - Response from
leader: Success: Value logged and distributed.
2024-11-16 18:48:55,569 - INFO - Leader found a
t http://10.128.0.4:17000/
To restart election "1"
To write values to all nodes through leader ent
er "2"
To write values to only leader and stimulate a
failure  enter "3"
(or "exit" to quit):
```

For relection based on that we have no slow followers, as we have shown above all writes have been achieved, we passed a value 1 through client to restart the election, leading to node3 as new leader successfully.



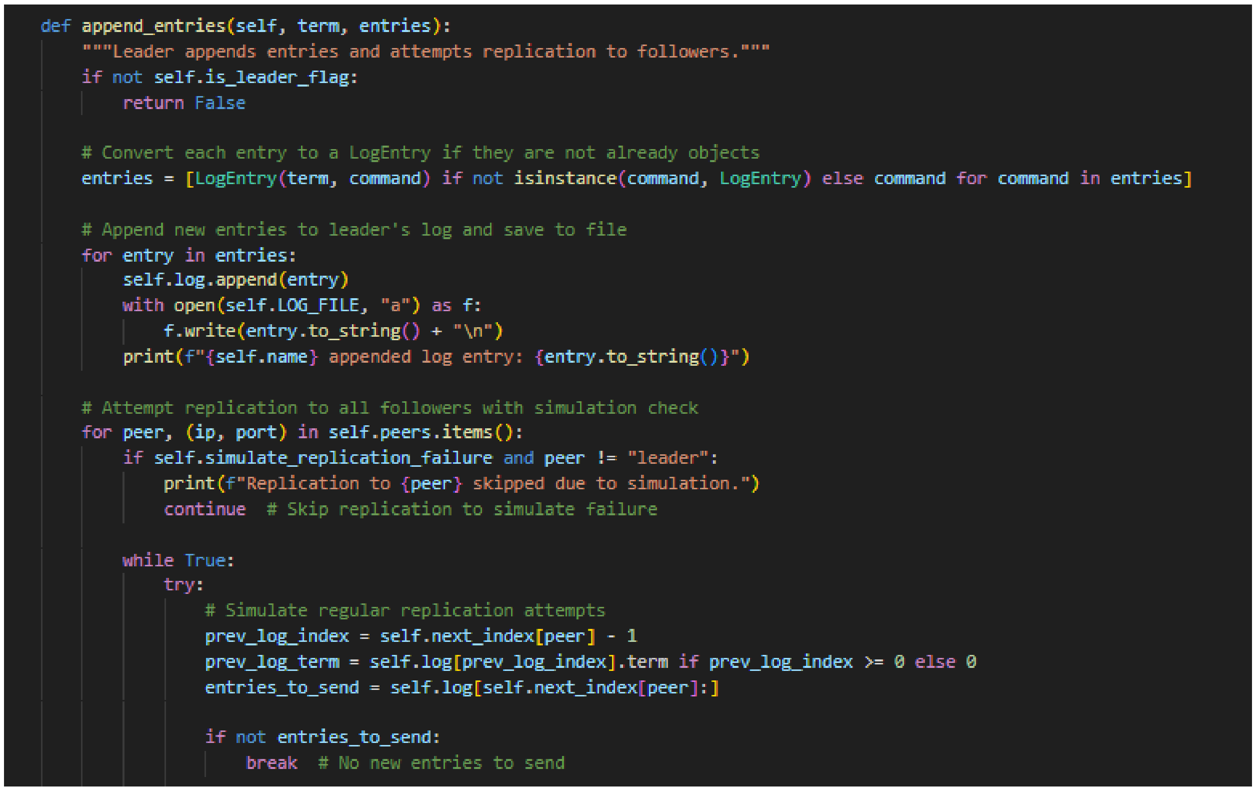
```
e1.
node1 appended log entry: 2,77
Successfully updated node2 with 1 entries.
Successfully updated node3 with 1 entries.
Leader node1 committed entry at index 1
node1 applying entry 0 (term 2): 44
Replication failure simulation disabled on nod
e1.
node1 appended log entry: 2,66
Successfully updated node2 with 1 entries.
Successfully updated node3 with 1 entries.
Leader node1 committed entry at index 2
node1 applying entry 0 (term 2): 44
node1 applying entry 1 (term 2): 77
node1 heartbeat interval set to 20.0 seconds.
node1 voted for node3 for term 3
node1 switching to follower due to received he
artbeat.
node1 heartbeat interval set to 0.1 seconds.
```

```
node2: Appended entry at index 0: 2,44
node2: Log refreshed from file with 1 entries.
node2: Appended entry at index 1: 2,77
node2: Log refreshed from file with 2 entries.
node2: Appended entry at index 2: 2,66
node2: Updated commit index from 0 to 1
node2 applying entry 0 (term 2): 44
node2 voted for node3 for term 3
```

```
node3 timeout, starting election.
node3 is requesting votes for term 3
node3 is now the leader.
election timeout 16.361920069177877
```

```
t http://10.128.0.4:17000/
To restart election "1"
To write values to all nodes through leader ent
er "2"
To write values to only leader and stimulate a
failure  enter "3"
(or "exit" to quit): 1
2024-11-16 18:50:39,290 - INFO - Heartbeat inte
rval temporarily set to 20.0 seconds on http://
10.128.0.4:17000/. Response: None
2024-11-16 18:50:59,322 - INFO - Heartbeat inte
rval reset to original value of 0.1 seconds on
http://10.128.0.4:17000/.
2024-11-16 18:50:59,449 - INFO - Leader changed
: New leader found at http://10.128.0.5:17002/
To restart election "1"
To write values to all nodes through leader ent
er "2"
To write values to only leader and stimulate a
failure  enter "3"
(or "exit" to quit):
```

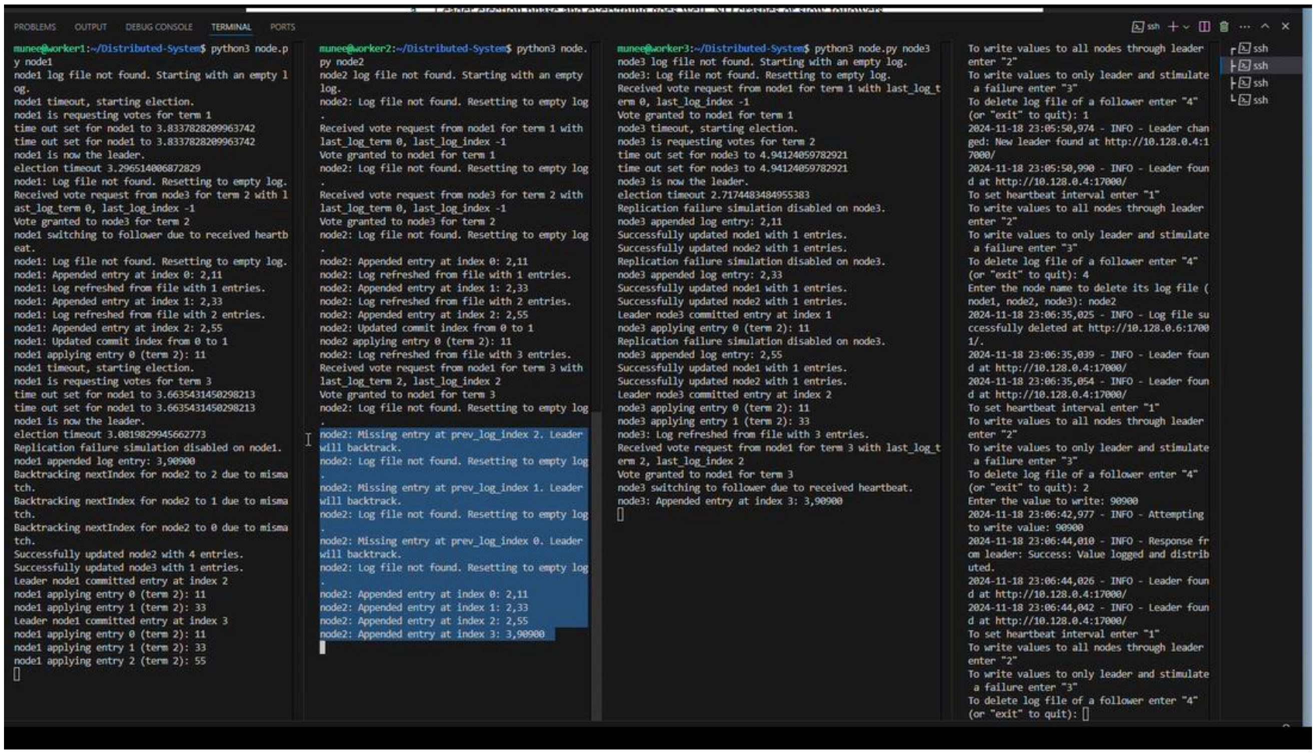
The append entries code implementation is below, which will write the inputs from the leader to each follower logs file and keep trach of indexes.



```python
def append_entries(self, term, entries):
    """Leader appends entries and attempts replication to followers."""
    if not self.is_leader_flag:
        return False

    # Convert each entry to a LogEntry if they are not already objects
    entries = [LogEntry(term, command) if not isinstance(command, LogEntry) else command for command in entries]

    # Append new entries to leader's log and save to file
    for entry in entries:
        self.log.append(entry)
        with open(self.LOG_FILE, "a") as f:
            f.write(entry.to_string() + "\n")
        print(f"{self.name} appended log entry: {entry.to_string()}")

    # Attempt replication to all followers with simulation check
    for peer, (ip, port) in self.peers.items():
        if self.simulate_replication_failure and peer != "leader":
            print(f"Replication to {peer} skipped due to simulation.")
            continue  # Skip replication to simulate failure

        while True:
            try:
                # Simulate regular replication attempts
                prev_log_index = self.next_index[peer] - 1
                prev_log_term = self.log[prev_log_index].term if prev_log_index >= 0 else 0
                entries_to_send = self.log[self.next_index[peer]:]

                if not entries_to_send:
                    break  # No new entries to send
```

c. Simulate a scenario that the leader crashed (e.g., log file deleted) with an inconsistency log (e.g., missing only ONE log entry) on the other two followers. Check out slide-8 page 62. i. Again, it can be started by the client's command.

To implement this part, we deleted the logs files from of the followers file (node2) and showed the back tracking based on raft so it can updated the follower logs files. In our case, we did it for node2 and leader perform the back track to update the values as shown below.



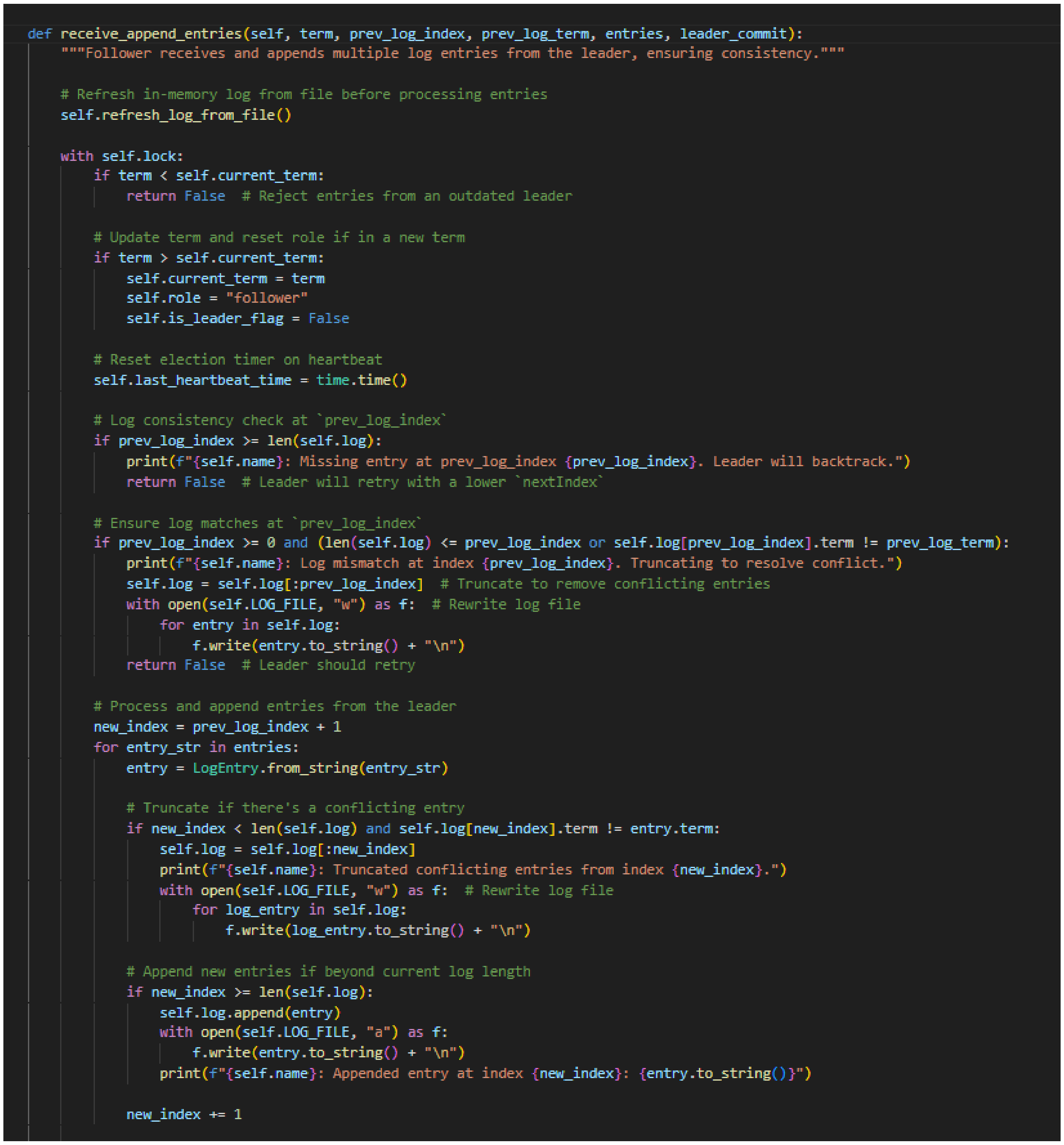
```
munee@worker1:~/Distributed-System$ python3 node.p
y node1
node1 log file not found. Starting with an empty l
og.
node1 timeout, starting election.
node1 is requesting votes for term 1
time out set for node1 to 3.8337828209963742
time out set for node1 to 3.8337828209963742
node1 is now the leader.
election timeout 3.296514006872829
node1: Log file not found. Resetting to empty log.
Received vote request from node3 for term 2 with l
ast_log_term 0, last_log_index -1
Vote granted to node3 for term 2
node1 switching to follower due to received heartb
eat.
node1: Log file not found. Resetting to empty log.
node1: Appended entry at index 0: 2,11
node1: Log refreshed from file with 1 entries.
node1: Appended entry at index 1: 2,33
node1: Log refreshed from file with 2 entries.
node1: Appended entry at index 2: 2,55
node1: Updated commit index from 0 to 1
node1 applying entry 0 (term 2): 11
node1 timeout, starting election.
node1 is requesting votes for term 3
time out set for node1 to 3.6635431450298213
time out set for node1 to 3.6635431450298213
node1 is now the leader.
election timeout 3.0819829945662773
Replication failure simulation disabled on node1.
node1 appended log entry: 3,90900
Backtracking nextIndex for node2 to 2 due to misma
tch.
Backtracking nextIndex for node2 to 1 due to misma
tch.
Backtracking nextIndex for node2 to 0 due to misma
tch.
Successfully updated node2 with 4 entries.
Successfully updated node3 with 1 entries.
Leader node1 committed entry at index 2
node1 applying entry 0 (term 2): 11
node1 applying entry 1 (term 2): 33
Leader node1 committed entry at index 3
node1 applying entry 0 (term 2): 11
node1 applying entry 1 (term 2): 33
node1 applying entry 2 (term 2): 55
```

```
munee@worker2:~/Distributed-System$ python3 node.
py node2
node2 log file not found. Starting with an empty
log.
node2: Log file not found. Resetting to empty log
.
Received vote request from node1 for term 1 with
last_log_term 0, last_log_index -1
Vote granted to node1 for term 1
node2: Log file not found. Resetting to empty log
.
Received vote request from node3 for term 2 with
last_log_term 0, last_log_index -1
Vote granted to node3 for term 2
node2: Log file not found. Resetting to empty log
.
node2: Appended entry at index 0: 2,11
node2: Log refreshed from file with 1 entries.
node2: Appended entry at index 1: 2,33
node2: Log refreshed from file with 2 entries.
node2: Appended entry at index 2: 2,55
node2: Updated commit index from 0 to 1
node2 applying entry 0 (term 2): 11
node2: Log refreshed from file with 3 entries.
Received vote request from node1 for term 3 with
last_log_term 2, last_log_index 2
Vote granted to node1 for term 3
node2: Log file not found. Resetting to empty log
.
node2: Missing entry at prev_log_index 2. Leader
will backtrack.
node2: Log file not found. Resetting to empty log
.
node2: Missing entry at prev_log_index 1. Leader
will backtrack.
node2: Log file not found. Resetting to empty log
.
node2: Missing entry at prev_log_index 0. Leader
will backtrack.
node2: Log file not found. Resetting to empty log
.
node2: Appended entry at index 0: 2,11
node2: Appended entry at index 1: 2,33
node2: Appended entry at index 2: 2,55
node2: Appended entry at index 3: 3,90900
```

```
munee@worker3:~/Distributed-System$ python3 node.py node3
node3 log file not found. Starting with an empty log.
node3: Log file not found. Resetting to empty log.
Received vote request from node1 for term 1 with last_log_t
erm 0, last_log_index -1
Vote granted to node1 for term 1
node3 timeout, starting election.
node3 is requesting votes for term 2
time out set for node3 to 4.94124059782921
time out set for node3 to 4.94124059782921
node3 is now the leader.
election timeout 2.7174483484955383
Replication failure simulation disabled on node3.
node3 appended log entry: 2,11
Successfully updated node1 with 1 entries.
Successfully updated node2 with 1 entries.
Replication failure simulation disabled on node3.
node3 appended log entry: 2,33
Successfully updated node1 with 1 entries.
Successfully updated node2 with 1 entries.
Leader node3 committed entry at index 1
node3 applying entry 0 (term 2): 11
Replication failure simulation disabled on node3.
node3 appended log entry: 2,55
Successfully updated node1 with 1 entries.
Successfully updated node2 with 1 entries.
Leader node3 committed entry at index 2
node3 applying entry 0 (term 2): 11
node3 applying entry 1 (term 2): 33
node3: Log refreshed from file with 3 entries.
Received vote request from node1 for term 3 with last_log_t
erm 2, last_log_index 2
Vote granted to node1 for term 3
node3 switching to follower due to received heartbeat.
node3: Appended entry at index 3: 3,90900
```

```
To write values to all nodes through leader
enter "2"
To write values to only leader and stimulate
 a failure enter "3"
To delete log file of a follower enter "4"
(or "exit" to quit): 1
2024-11-18 23:05:50,974 - INFO - Leader chan
ged: New leader found at http://10.128.0.4:1
7000/
2024-11-18 23:05:50,990 - INFO - Leader foun
d at http://10.128.0.4:17000/
To set heartbeat interval enter "1"
To write values to all nodes through leader
enter "2"
To write values to only leader and stimulate
 a failure enter "3"
To delete log file of a follower enter "4"
(or "exit" to quit): 4
Enter the node name to delete its log file (
node1, node2, node3): node2
2024-11-18 23:06:35,025 - INFO - Log file su
ccessfully deleted at http://10.128.0.6:1700
1/.
2024-11-18 23:06:35,039 - INFO - Leader foun
d at http://10.128.0.4:17000/
2024-11-18 23:06:35,054 - INFO - Leader foun
d at http://10.128.0.4:17000/
To set heartbeat interval enter "1"
To write values to all nodes through leader
enter "2"
To write values to only leader and stimulate
 a failure enter "3"
To delete log file of a follower enter "4"
(or "exit" to quit): 2
Enter the value to write: 90900
2024-11-18 23:06:42,977 - INFO - Attempting
to write value: 90900
2024-11-18 23:06:44,010 - INFO - Response fr
om leader: Success: Value logged and distrib
uted.
2024-11-18 23:06:44,026 - INFO - Leader foun
d at http://10.128.0.4:17000/
2024-11-18 23:06:44,042 - INFO - Leader foun
d at http://10.128.0.4:17000/
To set heartbeat interval enter "1"
To write values to all nodes through leader
enter "2"
To write values to only leader and stimulate
 a failure enter "3"
To delete log file of a follower enter "4"
(or "exit" to quit):
```

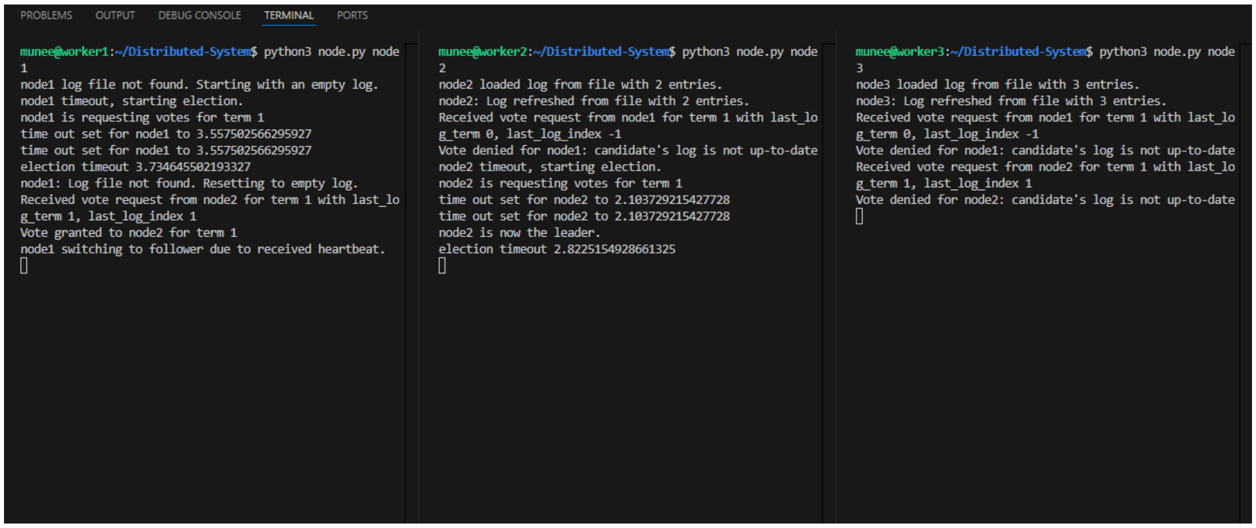
The code snippet for the above method is shown below, which update the logs based on current log in the node based on leader logs



```python
def receive_append_entries(self, term, prev_log_index, prev_log_term, entries, leader_commit):
    """Follower receives and appends multiple log entries from the leader, ensuring consistency."""

    # Refresh in-memory log from file before processing entries
    self.refresh_log_from_file()

    with self.lock:
        if term < self.current_term:
            return False  # Reject entries from an outdated leader

        # Update term and reset role if in a new term
        if term > self.current_term:
            self.current_term = term
            self.role = "follower"
            self.is_leader_flag = False

        # Reset election timer on heartbeat
        self.last_heartbeat_time = time.time()

        # Log consistency check at `prev_log_index`
        if prev_log_index >= len(self.log):
            print(f"{self.name}: Missing entry at prev_log_index {prev_log_index}. Leader will backtrack.")
            return False  # Leader will retry with a lower `nextIndex`

        # Ensure log matches at `prev_log_index`
        if prev_log_index >= 0 and (len(self.log) <= prev_log_index or self.log[prev_log_index].term != prev_log_term):
            print(f"{self.name}: Log mismatch at index {prev_log_index}. Truncating to resolve conflict.")
            self.log = self.log[:prev_log_index]  # Truncate to remove conflicting entries
            with open(self.LOG_FILE, "w") as f:  # Rewrite log file
                for entry in self.log:
                    f.write(entry.to_string() + "\n")
            return False  # Leader should retry

        # Process and append entries from the leader
        new_index = prev_log_index + 1
        for entry_str in entries:
            entry = LogEntry.from_string(entry_str)

            # Truncate if there's a conflicting entry
            if new_index < len(self.log) and self.log[new_index].term != entry.term:
                self.log = self.log[:new_index]
                print(f"{self.name}: Truncated conflicting entries from index {new_index}.")
                with open(self.LOG_FILE, "w") as f:  # Rewrite log file
                    for log_entry in self.log:
                        f.write(log_entry.to_string() + "\n")

            # Append new entries if beyond current log length
            if new_index >= len(self.log):
                self.log.append(entry)
                with open(self.LOG_FILE, "a") as f:
                    f.write(entry.to_string() + "\n")
                print(f"{self.name}: Appended entry at index {new_index}: {entry.to_string()}")

            new_index += 1
```

Optional BONUS: Simulate the crashed node rejoins the cluster. In this case, the leader needs to synchronize with the latest log files. The situation is similar to item c (slide-8 page 62), but with multiple rounds.

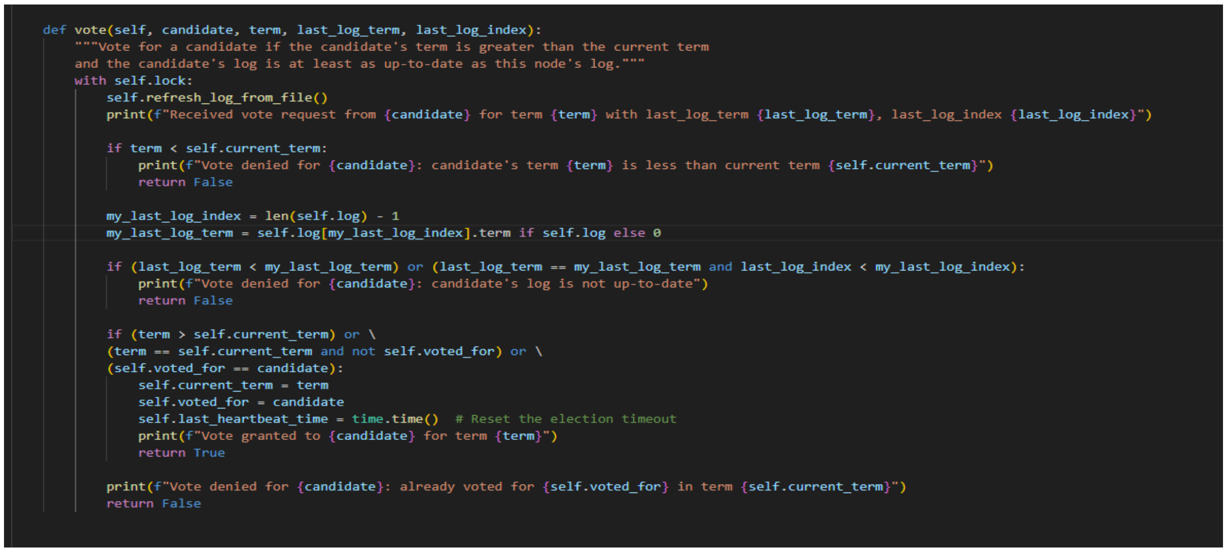
To achieved this, we modified the vote function, which reject the vote request from any candidate whose log is out of date based on it current log. We modified the logs of node1 by deleting it and when the new election phase start, it vote request was denied by both the node2 and node3 as shown below thus, choosing node2 as leader as it has the next request inline.



```
munee@worker1:~/Distributed-System$ python3 node.py node
1
node1 log file not found. Starting with an empty log.
node1 timeout, starting election.
node1 is requesting votes for term 1
time out set for node1 to 3.557502566295927
time out set for node1 to 3.557502566295927
election timeout 3.734645502193327
node1: Log file not found. Resetting to empty log.
Received vote request from node2 for term 1 with last_lo
g_term 1, last_log_index 1
Vote granted to node2 for term 1
node1 switching to follower due to received heartbeat.
```

```
munee@worker2:~/Distributed-System$ python3 node.py node
2
node2 loaded log from file with 2 entries.
node2: Log refreshed from file with 2 entries.
Received vote request from node1 for term 1 with last_lo
g_term 0, last_log_index -1
Vote denied for node1: candidate's log is not up-to-date
node2 timeout, starting election.
node2 is requesting votes for term 1
time out set for node2 to 2.103729215427728
time out set for node2 to 2.103729215427728
node2 is now the leader.
election timeout 2.8225154928661325
```

```
munee@worker3:~/Distributed-System$ python3 node.py node
3
node3 loaded log from file with 3 entries.
node3: Log refreshed from file with 3 entries.
Received vote request from node1 for term 1 with last_lo
g_term 0, last_log_index -1
Vote denied for node1: candidate's log is not up-to-date
Received vote request from node2 for term 1 with last_lo
g_term 1, last_log_index 1
Vote denied for node2: candidate's log is not up-to-date
```



```python
def vote(self, candidate, term, last_log_term, last_log_index):
    """Vote for a candidate if the candidate's term is greater than the current term
    and the candidate's log is at least as up-to-date as this node's log."""
    with self.lock:
        self.refresh_log_from_file()
        print(f"Received vote request from {candidate} for term {term} with last_log_term {last_log_term}, last_log_index {last_log_index}")

        if term < self.current_term:
            print(f"Vote denied for {candidate}: candidate's term {term} is less than current term {self.current_term}")
            return False

        my_last_log_index = len(self.log) - 1
        my_last_log_term = self.log[my_last_log_index].term if self.log else 0

        if (last_log_term < my_last_log_term) or (last_log_term == my_last_log_term and last_log_index < my_last_log_index):
            print(f"Vote denied for {candidate}: candidate's log is not up-to-date")
            return False

        if (term > self.current_term) or \
        (term == self.current_term and not self.voted_for) or \
        (self.voted_for == candidate):
            self.current_term = term
            self.voted_for = candidate
            self.last_heartbeat_time = time.time()  # Reset the election timeout
            print(f"Vote granted to {candidate} for term {term}")
            return True

        print(f"Vote denied for {candidate}: already voted for {self.voted_for} in term {self.current_term}")
        return False
```