CSE (4)586 Distributed System
Phase-IV Report

Team Members:

Apoorva Bhapkar – (50442001) abhapkar@buffalo.edu
Prashant Upadhyay – (50419393) pupadhya@buffalo.edu

Introduction:

Raft is a consensus algorithm designed to be easy to understand. It is comparable to Paxos in terms of fault tolerance and performance. The difference is that it is broken down into relatively independent subproblems and explicitly deals with all the key elements needed for real systems. We hope that Raft will make consensus accessible to a wider audience and that this broader audience will be able to develop many better-quality consensus-based systems than other existing system.

A consensus protocol tolerating failures must have the following features:

- Validity: If one process decides (read/write) a value, it must be suggested by another correct process
- Agreement: Every correct process must agree on the same value
- Termination: Every correct process must terminate after a finite number of steps.
- Integrity: If all correct processes decide on the same value, then any process has the said value.

Now, there can be two types of systems assuming only one client (for the sake of understandability):

- Single-server system: The client interacts with a single-server system without backup. There is no problem in reaching consensus in such a system.

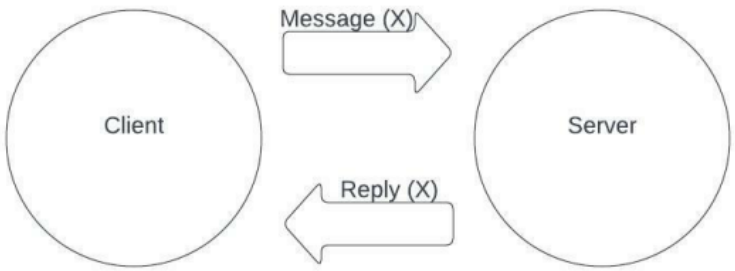


Fig 1. Single server RAFT visual

Multi-server system: The client interacts with a system with many servers. These systems can be of two types:

- Symmetric: Any of the multiple servers can respond to the client and all the other servers are supposed to sync up with the server that responded to the client's request.

- Asymmetric: Only the elected leader server can respond to the client. All other servers then sync up with the leader server.

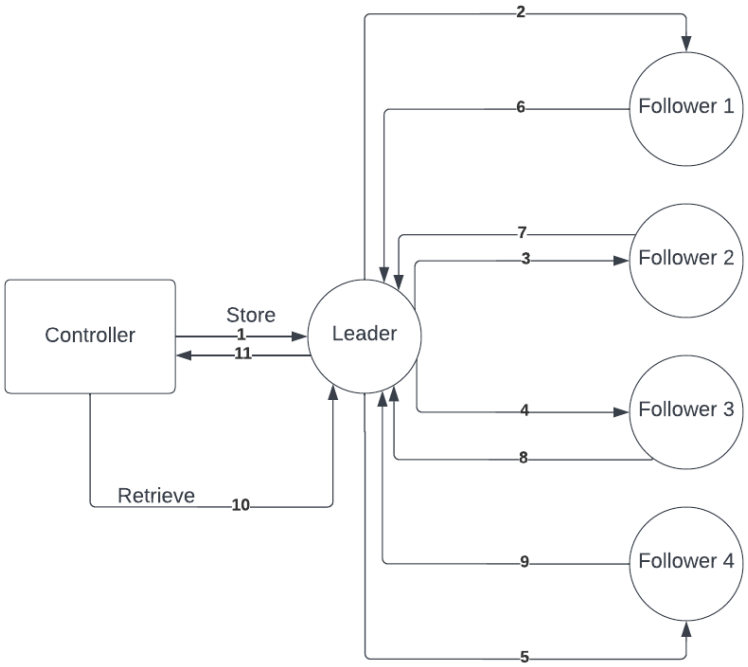


Fig 2. Multiple Server labeled RAFT visual.

In the above diagram label 2,3,4,5 are heartbeat and label 6,7,8,9 are their response with their term.

Leader - Only the server elected as the leader can interact with the client. All other servers sync with the leader. There can be at most one leader at any time.

Followers - The followers server synchronizes a copy of their data with the leader's data at periodic intervals. When the leader's server crashes (for whatever reason), one of the registrants can participate in an election and become the leader.

Candidates - When contesting to choose the leading server, the servers can request votes from other servers. So, they are called candidates when they ask for votes. Initially, all servers are in Candidate state.

Design Overview:

To maintain authority as the leader of the cluster, the leader node sends a heartbeat to show dominance over other follower nodes. Leader election occurs when a follower node times out waiting for a heartbeat from the leader node. At this point, the expiring node changes its state to a candidate state, votes for itself, and issues a Request Votes RPC to establish a majority and try to be the leader.

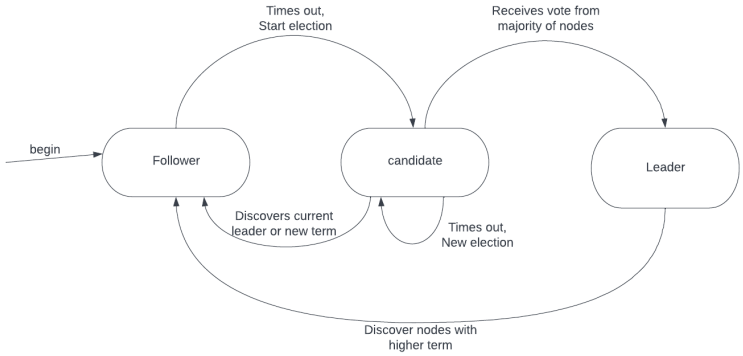


Fig 3. Raft consensus algorithm cycle.

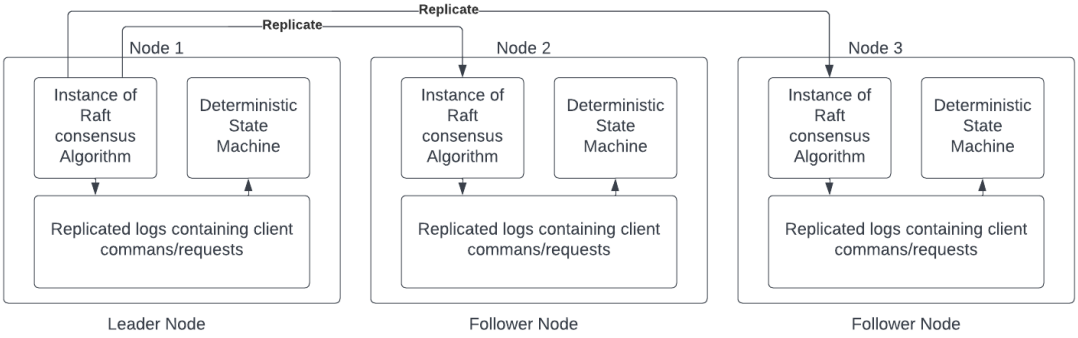
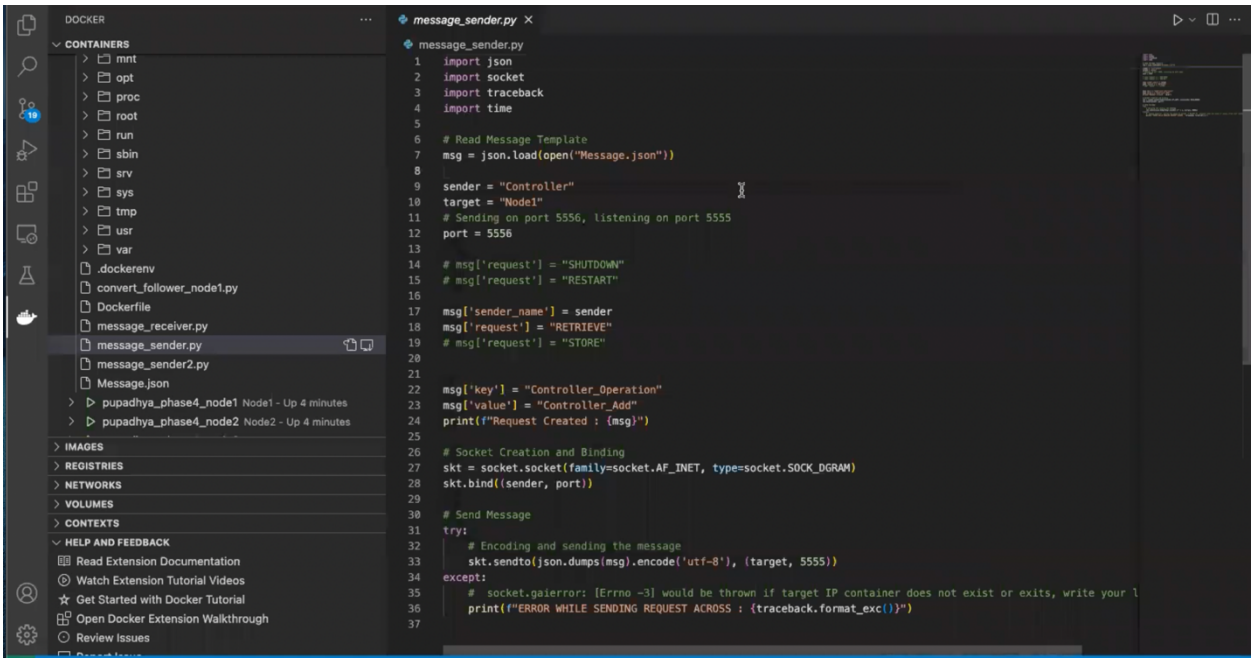
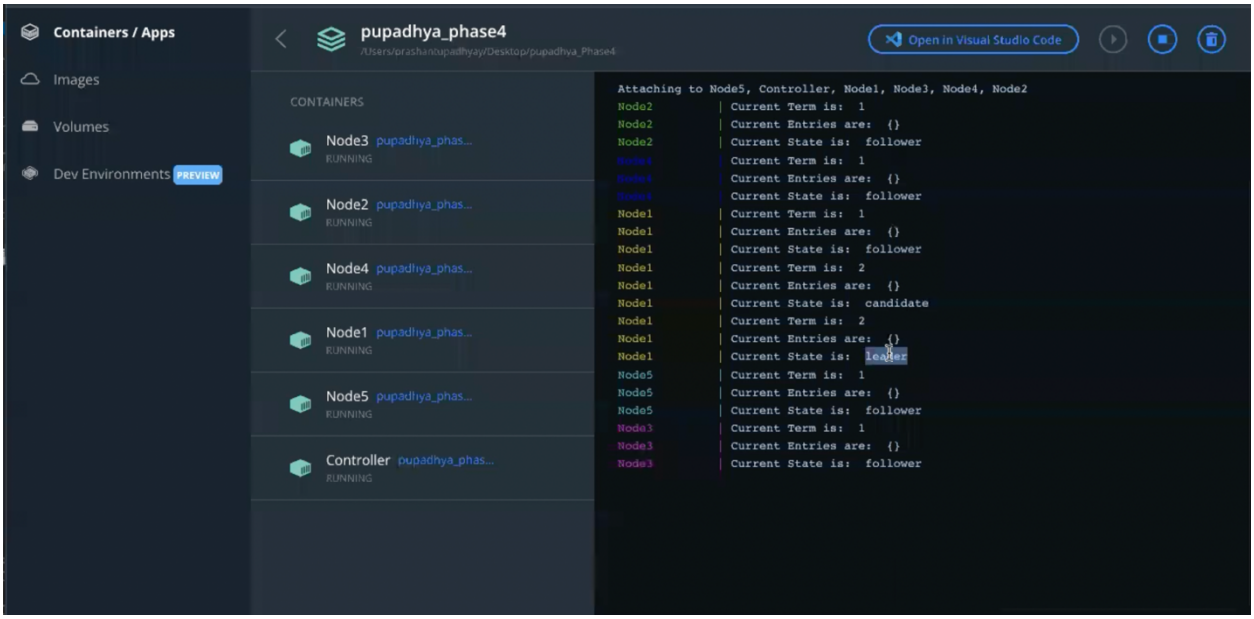


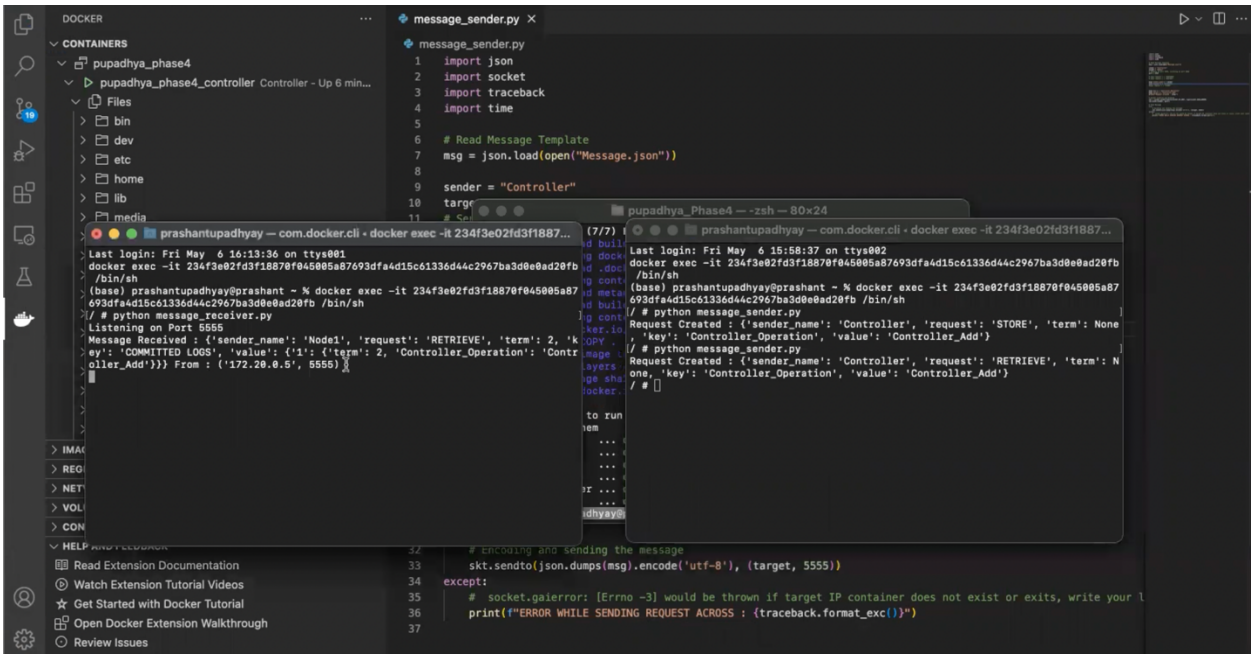
Fig 4. Leader to Follower Replication.

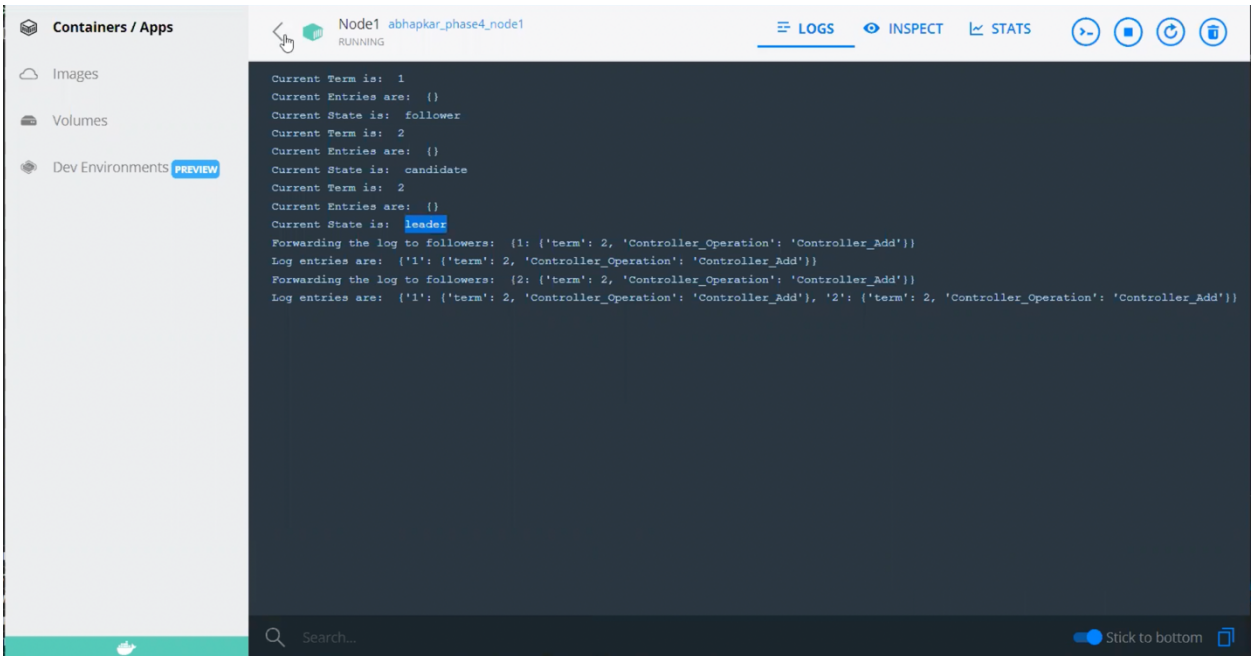
- The candidate node becomes the leader by getting the majority of votes from the nodes in the cluster. At this point, it updates its status to the Leader and starts sending heartbeats to notify the other servers of the new Leader.
- The Candidate node does not receive a majority of votes in the election and thus the term ends without a leader. The Candidate button returns to the Follower state.
- If the number of terms of the candidate node requesting votes is less than the number of terms of the other candidate nodes in the cluster, the RPC Append Entries is rejected, and the other nodes keep their candidate state. If the number of terms is more, the candidate node is elected as the new leader

Implementation:

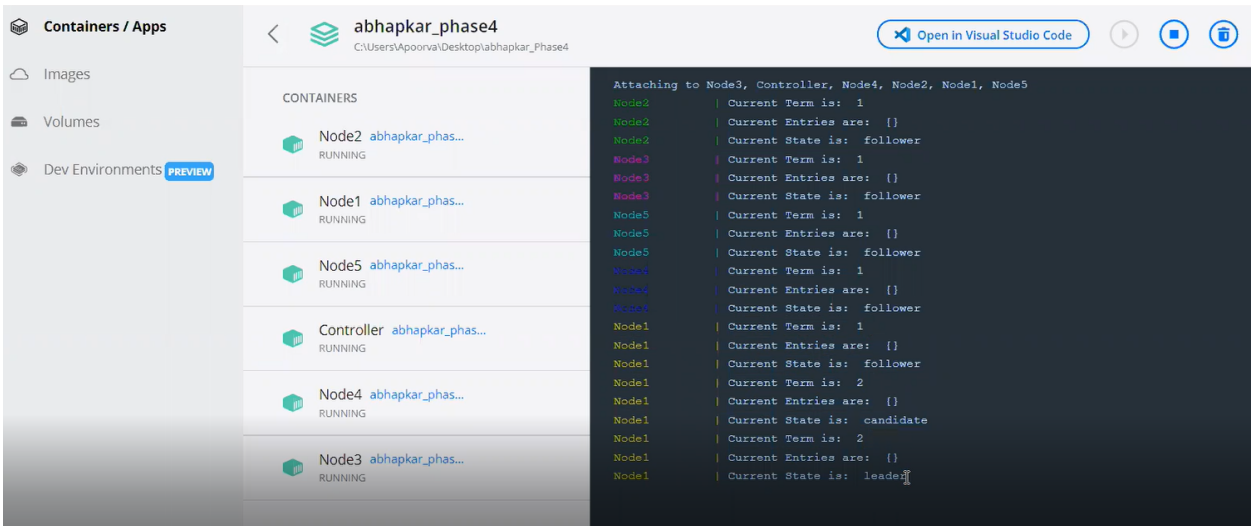


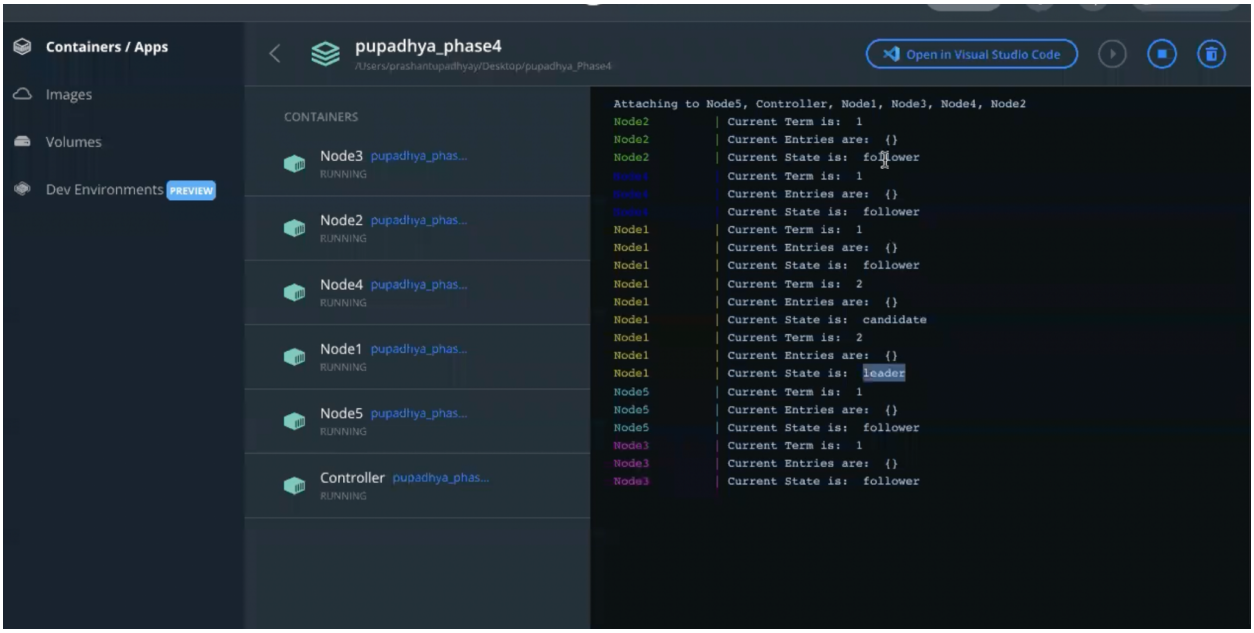



DOCKER
CONTAINERS
pupadhya_phase4
pupadhya_phase4_controller Controller - Up 6 min...
Files
bin
dev
etc
home
lib
media
message_sender.py
import json
import socket
import traceback
import time
# Read Message Template
msg = json.load(open("Message.json"))
sender = "Controller"
# Encoding and sending the message
skt.sendto(json.dumps(msg).encode('utf-8'), (target, 5555))
except:
# socket.gaierror: [Errno -3] would be thrown if target IP container does not exist or exits, write your l
print(f"ERROR WHILE SENDING REQUEST ACROSS : {traceback.format_exc()}")
prashantupadhyay — com.docker.cli • docker exec -it 234f3e02fd3f1887...
Last login: Fri May 6 16:13:36 on ttys001
docker exec -it 234f3e02fd3f18870f045005a87693dfa4d15c61336d44c2967ba3d0e0ad20fb /bin/sh
(base) prashantupadhyay@prashant ~ % docker exec -it 234f3e02fd3f18870f045005a87693dfa4d15c61336d44c2967ba3d0e0ad20fb /bin/sh
/ # python message_receiver.py
Listening on Port 5555
Message Received : {'sender_name': 'Node1', 'request': 'RETRIEVE', 'term': 2, 'key': 'COMMITTED LOGS', 'value': {'1': {'term': 2, 'Controller_Operation': 'Controller_Add'}}} From : ('172.20.0.5', 5555)
pupadhya_Phase4 — -zsh — 80×24
prashantupadhyay — com.docker.cli • docker exec -it 234f3e02fd3f1887...
Last login: Fri May 6 15:58:37 on ttys002
docker exec -it 234f3e02fd3f18870f045005a87693dfa4d15c61336d44c2967ba3d0e0ad20fb /bin/sh
(base) prashantupadhyay@prashant ~ % docker exec -it 234f3e02fd3f18870f045005a87693dfa4d15c61336d44c2967ba3d0e0ad20fb /bin/sh
/ # python message_sender.py
Request Created : {'sender_name': 'Controller', 'request': 'STORE', 'term': None, 'key': 'Controller_Operation', 'value': 'Controller_Add'}
/ # python message_sender.py
Request Created : {'sender_name': 'Controller', 'request': 'RETRIEVE', 'term': None, 'key': 'Controller_Operation', 'value': 'Controller_Add'}
/ #
Read Extension Documentation
Watch Extension Tutorial Videos
Get Started with Docker Tutorial
Open Docker Extension Walkthrough
Review Issues


Containers / Apps
Images
Volumes
Dev Environments PREVIEW
Node1 abhapkar_phase4_node1
RUNNING
LOGS
INSPECT
STATS
Current Term is: 1
Current Entries are: {}
Current State is: follower
Current Term is: 2
Current Entries are: {}
Current State is: candidate
Current Term is: 2
Current Entries are: {}
Current State is: leader
Forwarding the log to followers: {1: {'term': 2, 'Controller_Operation': 'Controller_Add'}}
Log entries are: {'1': {'term': 2, 'Controller_Operation': 'Controller_Add'}}
Forwarding the log to followers: {2: {'term': 2, 'Controller_Operation': 'Controller_Add'}}
Log entries are: {'1': {'term': 2, 'Controller_Operation': 'Controller_Add'}, '2': {'term': 2, 'Controller_Operation': 'Controller_Add'}}
Search...
Stick to bottom

Validation:


Containers / Apps
Images
Volumes
Dev Environments PREVIEW
abhapkar_phase4
C:\Users\Apoorva\Desktop\abhapkar_Phase4
Open in Visual Studio Code
CONTAINERS
Node2 abhapkar_phas... RUNNING
Node1 abhapkar_phas... RUNNING
Node5 abhapkar_phas... RUNNING
Controller abhapkar_phas... RUNNING
Node4 abhapkar_phas... RUNNING
Node3 abhapkar_phas... RUNNING
Attaching to Node3, Controller, Node4, Node2, Node1, Node5
Node2 | Current Term is: 1
Node2 | Current Entries are: {}
Node2 | Current State is: follower
Node3 | Current Term is: 1
Node3 | Current Entries are: {}
Node3 | Current State is: follower
Node5 | Current Term is: 1
Node5 | Current Entries are: {}
Node5 | Current State is: follower
Node4 | Current Term is: 1
Node4 | Current Entries are: {}
Node4 | Current State is: follower
Node1 | Current Term is: 1
Node1 | Current Entries are: {}
Node1 | Current State is: follower
Node1 | Current Term is: 2
Node1 | Current Entries are: {}
Node1 | Current State is: candidate
Node1 | Current Term is: 2
Node1 | Current Entries are: {}
Node1 | Current State is: leader


Containers / Apps
Images
Volumes
Dev Environments PREVIEW
pupadhya_phase4
/Users/prashantupadhyay/Desktop/pupadhya_Phase4
Open in Visual Studio Code
CONTAINERS
Node3 pupadhya_phas... RUNNING
Node2 pupadhya_phas... RUNNING
Node4 pupadhya_phas... RUNNING
Node1 pupadhya_phas... RUNNING
Node5 pupadhya_phas... RUNNING
Controller pupadhya_phas... RUNNING
Attaching to Node5, Controller, Node1, Node3, Node4, Node2
Node2 | Current Term is: 1
Node2 | Current Entries are: {}
Node2 | Current State is: follower
Node4 | Current Term is: 1
Node4 | Current Entries are: {}
Node4 | Current State is: follower
Node1 | Current Term is: 1
Node1 | Current Entries are: {}
Node1 | Current State is: follower
Node1 | Current Term is: 2
Node1 | Current Entries are: {}
Node1 | Current State is: candidate
Node1 | Current Term is: 2
Node1 | Current Entries are: {}
Node1 | Current State is: leader
Node5 | Current Term is: 1
Node5 | Current Entries are: {}
Node5 | Current State is: follower
Node3 | Current Term is: 1
Node3 | Current Entries are: {}
Node3 | Current State is: follower

Steps to execute:

steps to implement phase 4
1) Copy all the files to a directory. Open terminal/cmd and go to the directory where the file is stored.
2) Run the below command - docker-compose up –build -d
3) Open file message_sender.py, set target as the leader node and request as STORE.
4) Goto docker,launch controller CLI two times and run message_receiver.py in one and message_sender.py in another. And then verify logs.
5) Open file message_sender.py, set target as the leader node and request as RETRIEVE.
6) Run message_sender.py and then verify logs.
7) Open file message_sender.py, set target as the leader node and request as SHUTDOWN.
8) Run message_sender.py and then verify logs. Check new leader.
9) Repeat steps 5), 6), 7).
10) Open file message_sender.py, set target as the stopped node and request as RESTART.
11) Run message_sender.py and then verify logs.
12) Repeat steps 5), 6), 7).

Question:

For Phase2:

In this phase, there was no way to elect a leader so the log which is being forwarded from the leader is being replicated by the followers.
We implemented TCP in this phase which implies minimal loss of data transfer.

For Phase4:

In this phase, we implemented the raft algorithm which ensures the consensus among all the nodes.
Over here we follow three main procedures which are
a. Election: In this phase unlike the phase-3 the raft ensures that a node with latest logs is always elected as a leader. A node who does not receive a heartbeat initiates an election process. To ensure that there is only one leader we are using randomized timeouts.
b. Safe Replication and log matching: In this part, the leader ensures all the followers are updated. The logs are only committed when the majority of the nodes have applied them. To ensure log matching, whenever a new leader is elected it ensures that all the logs of the entries match with its own log entries.
c.Append Reply: When a follower receives an AppendRPC it has an option to accept or reject the log entry. The follower compares the incoming log entry with its current log entries and if the incoming log entries are not latest, then the follower accepts them or else it denies the log entry.

REFRENCES:

http://thesecretlivesofdata.com/raft/
https://raft.github.io/
https://raft.github.io/raft.pdf
https://www.youtube.com/watch?v=P9Ydif5_qvE
https://www.youtube.com/watch?v=vYp4LYbnnW8
https://www.youtube.com/watch?v=RHDP_KCrjUc
https://www.youtube.com/watch?v=6bBggO6KN_k
https://www.freecodecamp.org/news/in-search-of-an-understandable-consensus-algorithm-a-summary4bc294c97e0d
https://en.wikipedia.org/wiki/Raft_(algorithm)
https://en.wikipedia.org/wiki/Consensus_(computer_science)
https://people.cs.rutgers.edu/~pxk/417/notes/content/consensus.htm
https://www.researchgate.net/figure/State-change-model-for-Raft-algorithm_fig2_337936141
https://loonytek.com/2015/10/18/leader-election-and-log-replication-in-raft-part-1/
http://thesecretlivesofdata.com/raft/
https://www.youtube.com/watch?v=YbZ3zDzDnrw
https://docs.python.org/3/library/socket.html
https://docs.python.org/3/library/threading.html