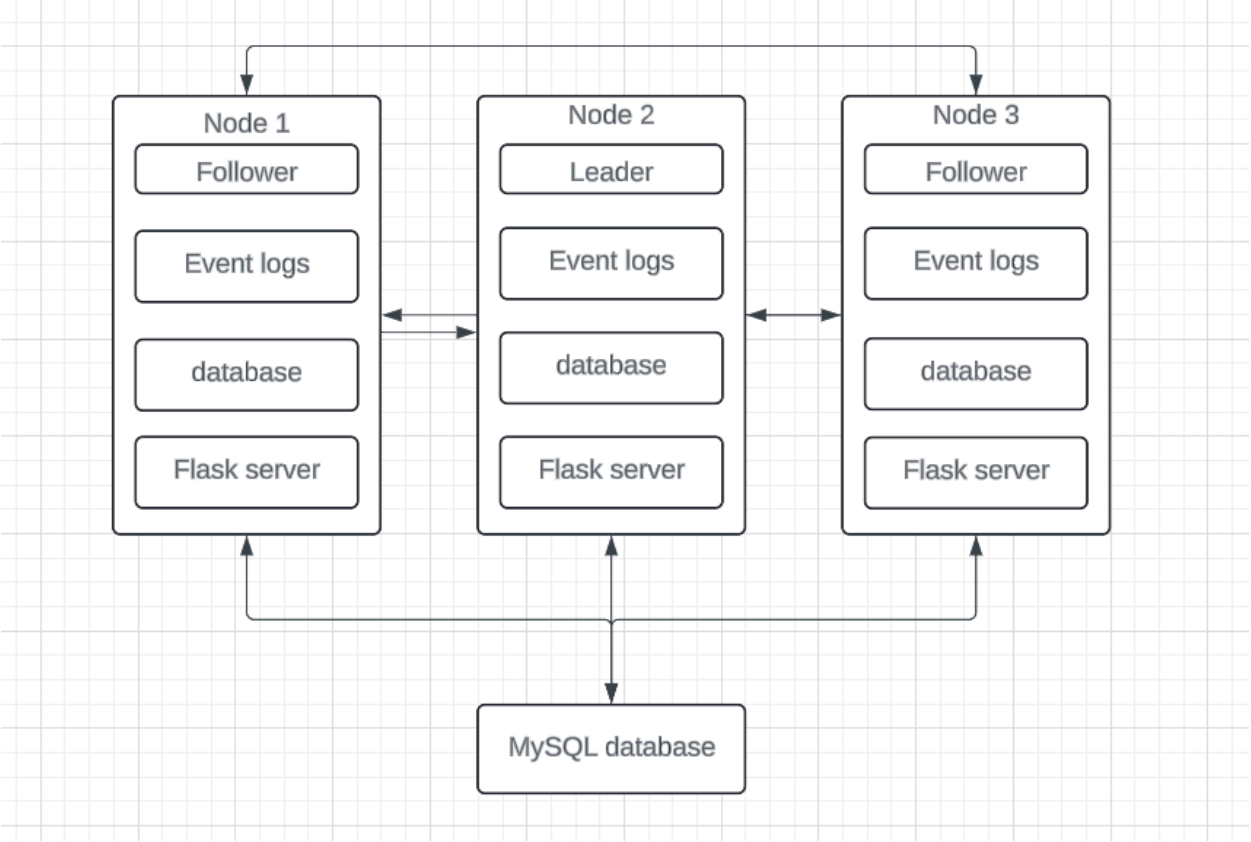
Cloud Computing
Problem Statement 2:
Building a Task Management Application with Raft Consensus Algorithm and MySQL

Weekly Deliverables:
Week1:
1. Designing the Application Architecture
    - Define the architecture of the task management application.
    - Determine the role of each node in the Raft consensus algorithm, such as leader, follower, and candidate.
    - Plan the schema for storing task data in MySQL, considering factors like table structure, relationships, and indexing.

Raft architecture



Task Management Application:

- The task management application will be designed as a distributed system consisting of multiple nodes (THREE), each running an instance of the application.
- Nodes will communicate with each other using the Raft consensus algorithm to ensure consistency and fault tolerance.

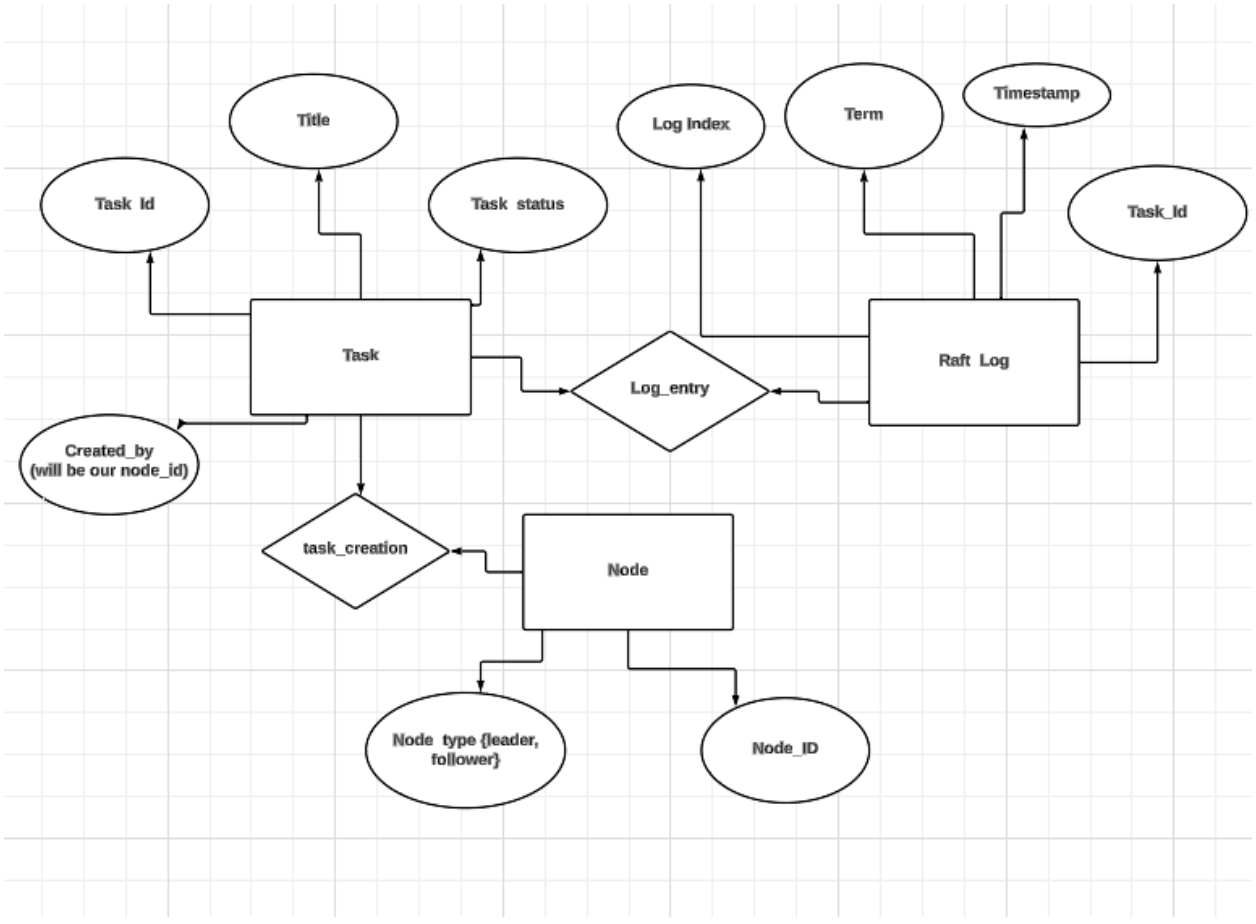
Role of Each Node:

Leader: One node acts as the leader which is responsible for coordinating actions and managing the Raft log. The leader receives client requests and replicates the log entries to followers. The leader is the one who can perform all of the CRUD operations.

Follower: Nodes that are not leaders will act as followers. They replicate the leader's log and respond to the client requests. Followers can also perform read operations.

Candidate: They are nodes that are actively campaigning to become the leader during leader elections rounds. This role would either transition to follower or leader based on the Raft's election protocol.

<u>ER Diagram/ Schema:</u>



Tables:

1. Task Table: Contains information about tasks such as TASK_ID, TITLE, TASK_STATUS, CREATED_BY
2. Node Table: If user management is part of the application, store user information like user ID, username, email, etc.
3. Raft Log Table (internal): Stores Raft log entries for maintaining consensus across nodes.

Relationships:

task_creations: The Task entity is related to the Node entity through a **one-to-many relationship,** established by a foreign key named 'created_by' in the Task table. This foreign key references the 'node_id' primary key in the Node table, indicating the specific Node that created each Task. This relationship enables the tracking of task creation provenance within the distributed system.

Log_entry: **One-to-One Relationship Between Tasks and Raft Logs:** Each task in the system has a dedicated Raft log to track its state changes and ensure consistency using the Raft consensus algorithm. This is achieved through a foreign key constraint, where the Task table's primary key ('task_id') likely acts as a foreign key ('task_id' or similar name) in the Raft Log table.

2. Implementing Raft Consensus Algorithm

● Implement Raft consensus algorithm to ensure fault tolerance and consistency across multiple nodes.

```
#1
import time
import threading
from pyraft import raft
from flask import Flask, Response, request,
copy_current_request_context, jsonify
import sys
```

```
import requests
from flask_cors import CORS
import logging

votes = 0
current_log_info = ''
current_log_id = 0
server_running = False
global app
app = Flask(__name__)
CORS(app)

def set_server_running():
    global server_running
    server_running = True
    return server_running
def get_server_running():
    global server_running
    return server_running
def update_log_id(id):
    global current_log_id
    current_log_id = id
    return current_log_id
def increament_votes():
    global votes
    votes +=1
    return votes
def set_votes():
    global votes
    votes =1
    return votes
def get_votes():
    global votes
    return votes
def set_current_log(log):
    global current_log_info
    current_log_info = log
    return current_log_info
def get_current_log():
    global current_log_info
```

```
    return current_log_info
def get_current_log_id ():
    global current_log_id
    return current_log_id
def increament_current_log_id():
    global current_log_id
    print(type(current_log_id))
    current_log_id = int(current_log_id)
    current_log_id += 1
    return current_log_id

@app.route('/brokers', methods=['POST'])
def get_data_brokers():
    print("reciever data from client :",request.data.decode())
    votes = set_votes()
    set_current_log(str(request.data.decode()))
    for peer in node.peers.values():
        url =
'http://127.0.1.1:'+str(int(peer.port)+1)+'/fromLeader'
        try:
            log_id = int(get_current_log_id())+1
            final_data = str(log_id)+request.data.decode()
            final_data = bytes(final_data,'ascii')
            r = requests.post(url=url, data=final_data)
            print("leader",votes)
            time.sleep(1)
        except:
            print("not able to forward message to
follower",peer.nid)

    return Response('We recieved something…')
#2
@app.route('/confirmation',methods=['POST','GET'])
def leader_confirm():
    print("confirmation from followers :",request.data.decode())
    no_of_nodes = 1 +len(node.peers)
    print(no_of_nodes)
    votes =increament_votes()
    data = get_current_log()
```

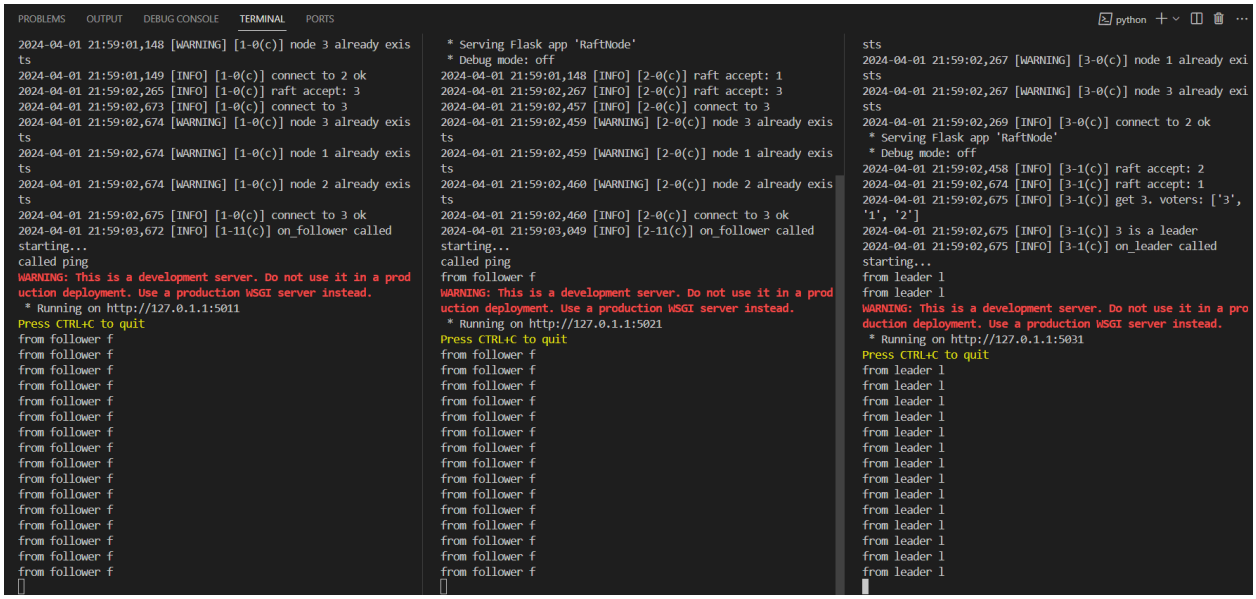
```
    if votes >no_of_nodes-1 :# count the no of nodes and make it
generalized
        log_id = increament_current_log_id()
        final_data = 'Log Id ('+str(log_id)+') : '+data
        filename = 'log'+str(node.nid)+'.txt'
        file = open(filename, "a")  # append mode
        file.write(str(final_data))
        file.close()
        # print("commit action")
    # print("confirmed action",request.data,"votes = ",votes)
    return Response('We recieved something…')
def set_log_id():
    filename = 'log'+str(node.nid)+'.txt'
    with open(filename, 'r') as file:
        lines = file.read().splitlines()
        try:
            last_line = lines[-1]
        except:
            return 0
        print (last_line)
    if last_line == '':
        return 0
    else:
        return last_line[8]
#3
@app.route('/fromLeader', methods=['POST'])
def get_data_leader():
    print("recieved message from leader :",request.data.decode())

    for peer in node.peers.values():
        if(peer.state =='l'):
            url =
'http://127.0.1.1:'+str(int(peer.port)+1)+'/confirmation'
            data = b'data recieved action'
            # try:
            r = requests.post(url=url, data=data)
            id = set_log_id()
            update_log_id(id)
            log_id = int(get_current_log_id())+1
```

```
            # print("data recieved
",request.data.decode(),int(request.data.decode()[0])," == ",log_id)
            if int(request.data.decode()[0]) == log_id:
                final_data = 'Log Id ('+str(log_id)+') :
'+str(request.data.decode()[1:])
                filename = 'log'+str(node.nid)+'.txt'
                file = open(filename, "a")  # append mode
                file.write(final_data)
                file.close()
                increament_current_log_id()
            elif int(request.data.decode()[0]) > log_id:
                print("request for more data")
            else:
                print("waiting for sync...")

    return Response('We recieved something…')

def leader_run_action(node):

    def ping():
        while not node.shutdown_flag:
            time.sleep(2)
            print("I am leader",node.state)
            for peer in node.peers.values():
                # print("peer-address",peer.addr)
                url = 'http://127.0.1.1:'+str(int(peer.port)+1)
                data = b'leader is alive'

    x1 = threading.Thread(target=ping)
    x1.start()
    x1.join()
#4
def leader_callback(node):
    print('starting...')
    node = threading.Thread(target=leader_run_action, args=(node,))
    node.start()

def follower_run_action(node):
    i = 0
```

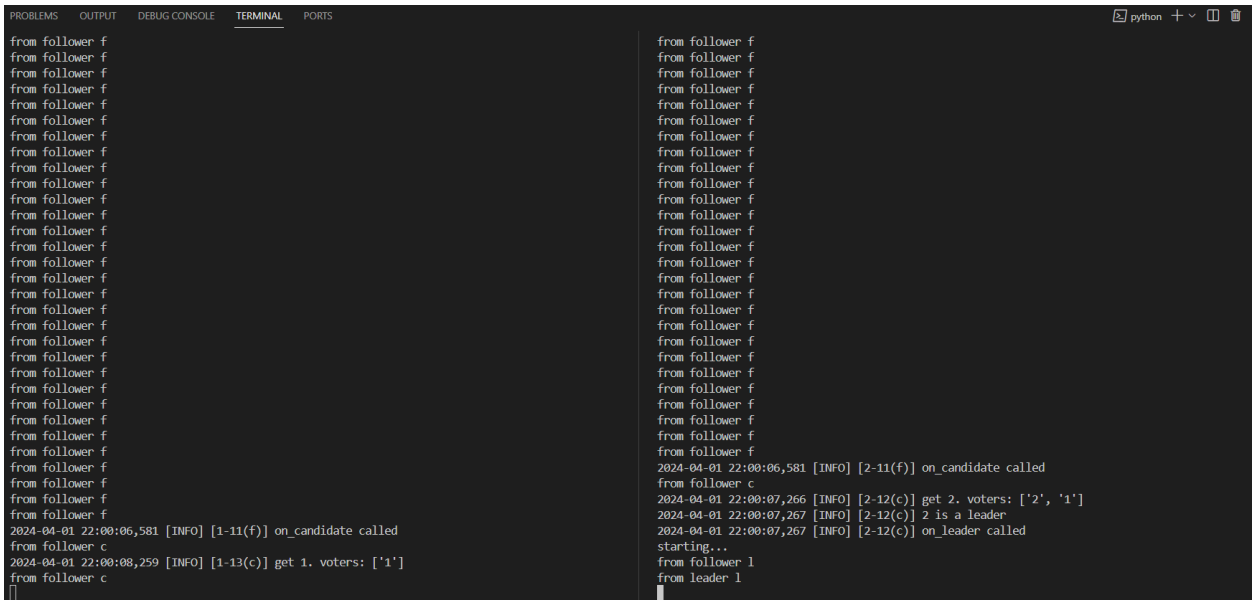
```
    ip = node.ip
    def ping():
        print("called ping")
        while not node.shutdown_flag:
            time.sleep(2)
            print("I am follower",node.state)
            for peer in node.peers.values():
                if(peer.state =='l'):
                    # print("leader-address",peer.addr)
                    url = 'http://127.0.1.1:'+str(int(peer.port)+1)
                    data = b'follower alive'

    x1 = threading.Thread(target=ping)
    x1.start()

def follower_callback(node):
    print('starting...')
    node = threading.Thread(target=follower_run_action,
args=(node,))
    node.start()


node = raft.make_default_node()

port = int(node.port)+1
def start_server():
    if get_server_running() == False:
        app.run(debug=False,port=port,host='127.0.1.1')
x4 = threading.Thread(target=start_server)
x4.start()

node.worker.handler['on_leader'] = leader_callback
node.worker.handler['on_follower'] = follower_callback

node.start()
node.join()
```

- Develop communication protocols between nodes for achieving consensus.
  HTTP Communication

- Test and validate the Raft implementation under different scenarios.



After Killing Leader



One Node Left: