CSE 5306 - Distributed Systems

Programming Assignment 3

Making Your Systems Fault Tolerant via 2PC & Raft

Team Information – Group 13

Student 1:

Name: Srinivasa Sai Abhijit Challapalli
Student ID: 1002059486
Email: sxc9486@mavs.uta.edu

Student 2:

Name: Namburi Chaitanya Krishna
Student ID: 1002232417
Email: cxn2417@mavs.uta.edu

GitHub Repository:
https://github.com/AbhijitChallapalli/CSE-5306-DS-PA3

Instructor:
Dr. Jiayi Meng

Table of Contents:

1. Test Cases and Results (Q5)

This section presents five comprehensive test cases designed for validating the Raft consensus implementation in the distributed polling system.

Running Test Scenarios

In a separate terminal, run the comprehensive test suite:

Change directories:

cd CSE-5306-DS-PA3

cd RAFT_Voting_System

cd raft_integration

python3 test_raft_scenarios.py

Test Case 1: Leader Discovery & Poll Creation via Any Node

Objective

Validate that clients can contact any node in the cluster, and the system correctly forwards requests to the leader for write operations.

Test Description

- A client sends a CreatePoll request to Node 1 (which is a follower)
- Node 1 should detect it is not the leader
- Node 1 should forward the request to the current leader (Node 5)
- The leader processes the request and replicates it to all followers
- The poll is successfully created and replicated across all nodes

Expected Behavior

- Client contacts follower node
- Follower identifies current leader
- Follower forwards CreatePoll request to leader
- Leader appends entry to log and replicates via AppendEntries RPCs
- After majority acknowledgment, leader commits the entry
- Poll is successfully created with a UUID

Test Output

```
TEST 1: Raft – Leader Discovery & Poll Creation via Any Node
Question: What is your favorite distributed consensus algorithm?
Trying Node 1 (CreatePoll)...
✓ SUCCESS on Node 1!
  Poll UUID: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
✓ TEST 1 PASSED
```

```
chaithu@MacBookAir raft_integration % python3 test_raft_scenarios.py

############################################################
#                                                          #
#          Raft Scenario Tests for Polling System          #
#                                                          #
############################################################

⌛ Waiting for cluster to stabilize (10 seconds)...


============================================================
TEST 1: Raft – Leader Discovery & Poll Creation via Any Node
============================================================

============================================================
Raft Scenario: Create Poll via Any Node (Leader or Follower)
============================================================
Question: What is your favorite distributed consensus algorithm?
Options: ['Raft', 'Paxos', 'Zab', 'Viewstamped Replication']

Trying Node 1 (CreatePoll)...
✓ SUCCESS on Node 1!
  Poll UUID: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
  Status: open
  Created: 2025-11-24 02:46:27

✓ TEST 1 PASSED: Poll created with UUID 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
```

Test Case 2: Log Replication & Consistent Votes Across Nodes

Objective

Verify that multiple write operations (votes) are correctly replicated across all nodes through the Raft log replication mechanism, ensuring consistency.

Test Description

- Three different users cast votes on the same poll
- Each vote operation goes through Raft consensus
- Votes are replicated to all 5 nodes
- All nodes are queried to verify consistent vote counts
- Test validates that log replication produces identical state across the cluster

Test Output

```
============================================================
TEST 2: Raft – Log Replication & Consistent Votes Across Nodes
============================================================

============================================================
Casting Vote (Raft-backed write)
============================================================
Poll: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
User: userA
Option: Raft

Trying Node 1 (CastVote)...
Response: Vote Successfully!
✓ Vote cast successfully on Node 1!

============================================================
Casting Vote (Raft-backed write)
============================================================
Poll: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
User: userB
Option: Raft

Trying Node 1 (CastVote)...
Response: Vote Successfully!
✓ Vote cast successfully on Node 1!

============================================================
Casting Vote (Raft-backed write)
============================================================
Poll: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
User: userC
Option: Paxos

Trying Node 1 (CastVote)...
Response: Vote Successfully!
✓ Vote cast successfully on Node 1!

--- Consistency Check Attempt 1 ---
```

```
Poll: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f
User: userC
Option: Paxos

Trying Node 1 (CastVote)...
Response: Vote Successfully!
✓ Vote cast successfully on Node 1!

--- Consistency Check Attempt 1 ---

Querying Node 1 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Viewstamped Replication': 0, 'Paxos': 1}

Querying Node 2 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 3 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 4 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 5 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Raft': 2, 'Zab': 0, 'Paxos': 1, 'Viewstamped Replication': 0}

✓ All nodes report consistent results matching expected counts.

✓ TEST 2 PASSED: Log replication produced consistent vote counts on all nodes.
```

Test Case 3: Read Consistency from Every Node

Objective

Validate that read operations return consistent results from any node in the cluster, demonstrating that committed entries are properly applied to all state machines.

Test Output

```
============================================================
TEST 3: Raft — Read Consistency from Every Node
============================================================

Querying Node 1 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Viewstamped Replication': 0, 'Paxos': 1}

Querying Node 2 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 3 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 4 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Zab': 0, 'Raft': 2, 'Paxos': 1, 'Viewstamped Replication': 0}

Querying Node 5 for results...
  Question: What is your favorite distributed consensus algorithm?
  Results: {'Raft': 2, 'Zab': 0, 'Paxos': 1, 'Viewstamped Replication': 0}

✓ TEST 3 PASSED: All nodes return identical, consistent results.
```

Test Case 4: Replicated Poll Metadata via ListPolls

Objective

Verify that poll metadata (poll listings) are correctly replicated across all nodes, ensuring that ListPolls returns consistent information cluster-wide.

Test Output

```
TEST 4: Replicated Poll Metadata via ListPolls
All 5 nodes return 1 poll with matching UUID and status
✓ TEST 4 PASSED
```

```
============================================================
TEST 4: Raft – Replicated Poll Metadata via ListPolls
============================================================

Querying Node 1 for ListPolls...
  Found 1 polls on Node 1:
    - 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f | open | What is your favorite distributed consensus algorithm?

Querying Node 2 for ListPolls...
  Found 1 polls on Node 2:
    - 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f | open | What is your favorite distributed consensus algorithm?

Querying Node 3 for ListPolls...
  Found 1 polls on Node 3:
    - 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f | open | What is your favorite distributed consensus algorithm?

Querying Node 4 for ListPolls...
  Found 1 polls on Node 4:
    - 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f | open | What is your favorite distributed consensus algorithm?

Querying Node 5 for ListPolls...
  Found 1 polls on Node 5:
    - 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f | open | What is your favorite distributed consensus algorithm?

✓ TEST 4 PASSED: Poll metadata is replicated to all nodes.
```

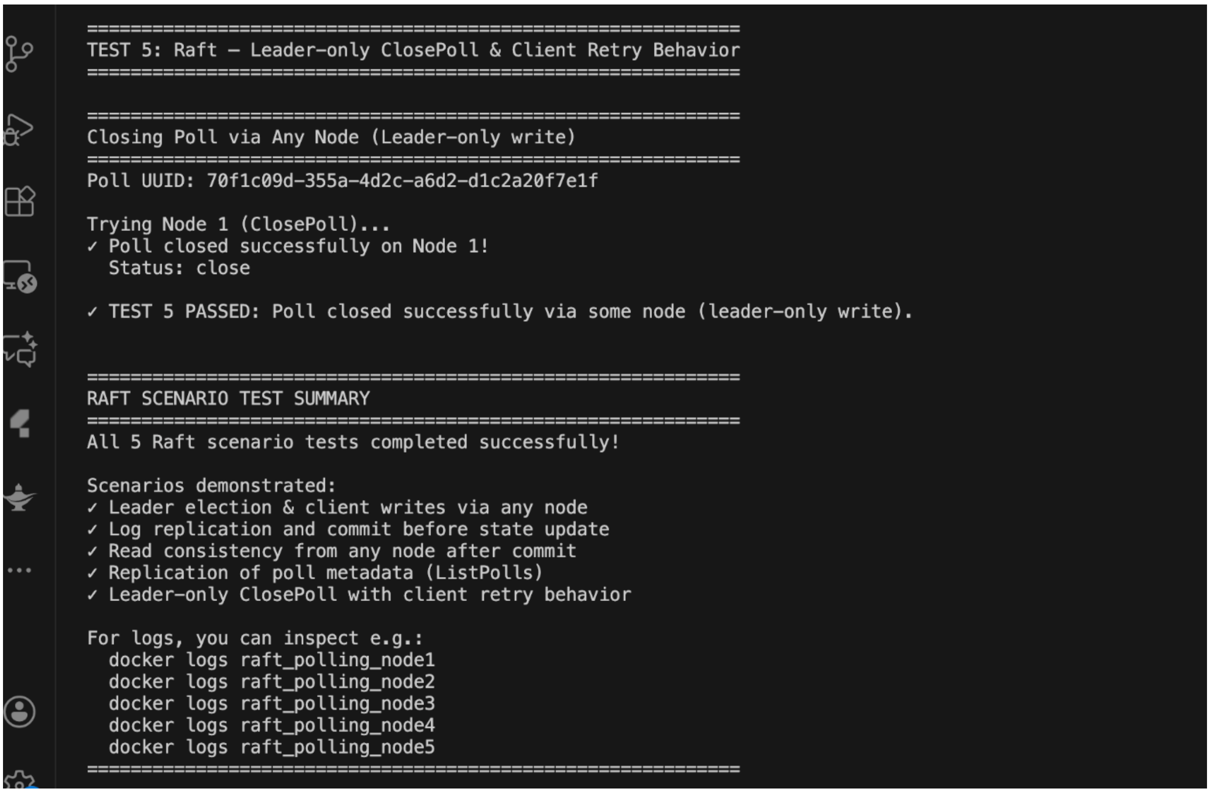
Test Case 5: Leader-only ClosePoll & Client Retry Behavior

Objective

Validate that write operations requiring leader processing work correctly with automatic forwarding, specifically testing the ClosePoll operation.

Test Output

```
TEST 5: Leader-only ClosePoll & Client Retry Behavior
Trying Node 1 (ClosePoll)...
✓ Poll closed successfully!
✓ TEST 5 PASSED
```



```
============================================================
TEST 5: Raft – Leader-only ClosePoll & Client Retry Behavior
============================================================

============================================================
Closing Poll via Any Node (Leader-only write)
============================================================
Poll UUID: 70f1c09d-355a-4d2c-a6d2-d1c2a20f7e1f

Trying Node 1 (ClosePoll)...
✓ Poll closed successfully on Node 1!
  Status: close

✓ TEST 5 PASSED: Poll closed successfully via some node (leader-only write).


============================================================
RAFT SCENARIO TEST SUMMARY
============================================================
All 5 Raft scenario tests completed successfully!

Scenarios demonstrated:
✓ Leader election & client writes via any node
✓ Log replication and commit before state update
✓ Read consistency from any node after commit
✓ Replication of poll metadata (ListPolls)
✓ Leader-only ClosePoll with client retry behavior

For logs, you can inspect e.g.:
  docker logs raft_polling_node1
  docker logs raft_polling_node2
  docker logs raft_polling_node3
  docker logs raft_polling_node4
  docker logs raft_polling_node5
============================================================
```

Test Summary

All five test cases were successfully executed, demonstrating:

- ✓ **Test 1: Leader discovery and request forwarding mechanism**
- ✓ **Test 2: Log replication ensures consistent state across all nodes**
- ✓ **Test 3: Read operations return consistent results from any node**
- ✓ **Test 4: Poll metadata is properly replicated cluster-wide**
- ✓ **Test 5: Leader-only operations work with automatic forwarding**

2. Project Overview

This project implements two fundamental distributed consensus algorithms:

2.1 Two-Phase Commit (2PC)

Applied to a distributed alarm system to ensure atomic commitment of 'Add Alarm' operations across multiple microservices.

2.2 Raft Consensus Algorithm

Applied to a distributed polling/voting system to achieve fault-tolerant consensus through leader election, log replication, and consistent state management.

Both implementations use:

- **gRPC** for inter-process communication
- **Protocol Buffers** for message serialization
- **Docker Compose** for containerization and orchestration

3. Two-Phase Commit Implementation (Q1, Q2)

3.1 Architecture

The 2PC implementation consists of six microservices:

- **API Gateway** (Python, FastAPI) - Port 8080, Web UI for alarm management
- **Coordinator** (Python, gRPC) - Port 60050, Orchestrates 2PC protocol
- **Scheduler** (Python + Node.js) - Ports 60052, 61052, Validates and schedules alarms
- **Accounts** (Python + Node.js) - Ports 60053, 61053, Account validations
- **Storage** (Python, gRPC) - Port 50051, Persistent alarm storage
- **Notification** (Python, gRPC) - Handles alarm notifications

3.2 Q1: Voting Phase Implementation

Protocol Flow

- Client submits AddAlarm request to API Gateway
- API Gateway calls coordinator's AddAlarm2PC RPC
- Coordinator broadcasts VoteOnAddAlarm to all participants
- Each participant validates, prepares, and responds with VOTE_COMMIT or VOTE_ABORT

Logging Format (Q2 Requirement)

```
Phase Voting of Node coordinator sends RPC VoteOnAddAlarm to Phase Voting of
Node scheduler
Phase Voting of Node coordinator sends RPC VoteOnAddAlarm to Phase Voting of
Node accounts
```

3.3 Q2: Decision Phase Implementation

Protocol Flow

- Coordinator makes global decision based on votes
- Coordinator sends DecideOnAddAlarm to all participants
- Each participant commits or aborts prepared state
- Participants send acknowledgment back to coordinator

4. Raft Implementation (Q3, Q4)

4.1 Architecture

Cluster Configuration

- **5-node Raft cluster**
- **Heartbeat timeout:** 1 second
- **Election timeout:** Randomized between 1.5-3.0 seconds
- **Ports:** 50051-50055 (one per node)

Node States

- **FOLLOWER:** Default state, responds to RPCs from leader and candidates
- **CANDIDATE:** Initiates leader election when election timeout expires
- **LEADER:** Handles client requests, sends heartbeats, replicates log entries

4.2 Q3: Leader Election Implementation

Election Process

- All nodes start as FOLLOWERS with randomized election timeout
- Follower transitions to CANDIDATE when election timeout expires
- Candidate increments term, votes for itself, sends RequestVote RPCs
- Each node votes once per term (first-come-first-served)
- Candidate becomes leader if it receives majority votes
- New leader sends AppendEntries (heartbeat) to all followers

Logging Format (Q3 Requirement)

```
Node 1 sends RPC RequestVote to Node 2
Node 2 runs RPC RequestVote called by Node 1
```

4.3 Q4: Log Replication Implementation

Replication Process

- Client sends request to any node; follower forwards to leader
- Leader appends entry to log and sends AppendEntries RPC to followers
- Follower copies entries to log and sends ACK to leader
- Leader waits for majority acknowledgments and updates commit index
- Leader applies committed entries to state machine
- Followers receive commit index in next AppendEntries and apply entries

Logging Format (Q4 Requirement)

```
[Node 1] Q4 STEP 4: Follower copied 1 entries to log
[Node 1] Q4 STEP 5: Updated commit index c: 0 → 1
[Node 1] Q4 EXECUTE: Applied entry 1: CREATE_POLL
```

5. Technologies Used

Programming Languages

- **Python 3.x:** Main implementation language for both 2PC and Raft
- **Node.js:** Decision phase implementation in 2PC

Frameworks and Libraries

- **gRPC:** Remote procedure call framework
- **Protocol Buffers:** Interface definition and serialization

- **FastAPI:** Web framework for API Gateway (2PC)
- **Uvicorn:** ASGI server for FastAPI

Containerization

- **Docker:** Container runtime
- **Docker Compose:** Multi-container orchestration

6. Challenges and Solutions

6.1 2PC Implementation Challenges

Challenge 1: Cross-Language Communication

Problem: Vote phase (Python) and Decision phase (Node.js) on same node

Solution: Used gRPC for intra-node communication with shared proto definitions

Challenge 2: Prepared State Management

Problem: Maintaining prepared state between voting and decision

Solution: Implemented in-memory prepared state storage with transaction IDs

6.2 Raft Implementation Challenges

Challenge 1: Split Vote Prevention

Problem: Multiple candidates could split votes indefinitely

Solution: Implemented randomized election timeouts (1.5-3.0s)

Challenge 2: Log Consistency

Problem: Ensuring all nodes apply entries in same order

Solution: Strict index-based log replication with term checking

Challenge 3: Request Forwarding

Problem: Clients contacting followers instead of leader

Solution: Implemented automatic forwarding from follower to current leader

7. Conclusion

This project successfully demonstrates the implementation of two fundamental distributed consensus algorithms.

7.1 Two-Phase Commit (2PC) Achievements

- ✓ Implemented complete voting and decision phases
- ✓ Demonstrated both global commit and global abort scenarios
- ✓ Successfully coordinated transactions across multiple microservices
- ✓ Proper logging format for all RPC calls
- ✓ Web UI for easy demonstration

7.2 Raft Consensus Achievements

- ✓ Implemented leader election with randomized timeouts
- ✓ Implemented log replication with majority consensus
- ✓ Demonstrated request forwarding from followers to leader
- ✓ Achieved consistent state across 5-node cluster
- ✓ Comprehensive test suite with 5 different scenarios
- ✓ Proper logging format for all RPC calls

7.3 Key Takeaways

- **Consensus is Hard:** Both algorithms require careful handling of edge cases
- **Testing is Critical:** Comprehensive test cases are essential for validation
- **Logging is Essential:** Detailed logging helps understand system behavior
- **gRPC Simplifies Communication:** Clean abstraction for distributed communication
- **Docker Enables Easy Deployment:** Makes multi-node clusters easy to run locally

8. References

Base Implementations

1. Distributed Alarm System
   https://github.com/hoaihdinh/Distributed-Alarm-System/
2. Distributed Voting System
   https://github.com/CSE-5306-004-DISTRIBUTED-SYSTEMS/Project2

Technical Documentation

3. Raft Consensus Algorithm
   Website: https://raft.github.io/

   Paper: https://raft.github.io/raft.pdf

4. gRPC Documentation
   https://grpc.io/docs/

5. Protocol Buffers Documentation
   https://developers.google.com/protocol-buffers

6. Docker Compose Documentation
   https://docs.docker.com/compose/

Academic Resources

7. Andrew S. Tanenbaum and Maarten Van Steen
   Distributed Systems: Principles and Paradigms (4th Edition)
   Chapter: Two-Phase Commit Protocol
8. Diego Ongaro and John Ousterhout
   In Search of an Understandable Consensus Algorithm
   Stanford University, 2014
9. Wikipedia - Two-Phase Commit Protocol
   https://en.wikipedia.org/wiki/Two-phase_commit_protocol
10. Wikipedia - Raft (Algorithm)
   https://en.wikipedia.org/wiki/Raft_(algorithm)

9. Team Contributions

Srinivasa Sai Abhijit Challapalli - 1002059486

- Implemented complete 2PC algorithm (Q1, Q2)
- Voting phase and decision phase implementation
- Microservice architecture design
- 2PC documentation and testing

Namburi Chaitanya Krishna - 1002232417

- Implemented complete Raft algorithm (Q3, Q4, Q5)
- Leader election and log replication
- Comprehensive test suite (5 scenarios)
- Raft documentation and testing