Cloud Computing - Mini Project Report

Implementing Raft Logic in Go

April 2023

Submitted By:

Name :	SHIVA K S	[PES2UG20CS484]
	HARIPRADA	[PES2UG20CS510]
	CHANDAN S A	[PES2UG20CS503]
	CETHAN REDDY H S	[PES2UG20CS505]

VI Semester Section _H
PES University

Short Description and Scope of the Project

Description:

The project aims to implement the Raft consensus algorithm in the Go programming language. The Raft algorithm is a distributed consensus algorithm designed to ensure that all nodes in a distributed system agree on the same state. The algorithm achieves this by electing a leader node, which coordinates the replication of the system's state across all nodes, and ensuring that the majority of nodes agree on the new state.

Implementing Raft in Go will provide a valuable opportunity for learners to develop an understanding of distributed systems, as well as gain practical experience in Go programming. This project will involve implementing Raft's various components, including leader election, log replication, and state machine updates, and writing comprehensive tests to validate the implementation.

Scope:

The scope of the project will involve the following tasks:

Designing the architecture and interfaces: This will involve designing the architecture of the Raft implementation and defining the interfaces that will be used by different components of the system.

Implementing the Raft components: This will involve implementing the various components of the Raft algorithm, including leader election, log replication, and state machine updates. The implementation will be done in Go, using best practices to ensure that the code is clean, modular, and efficient.

Writing unit tests: This will involve writing a suite of unit tests to validate the implementation of each component of the Raft algorithm. The tests will cover all possible edge cases and will ensure that the implementation is correct and robust.

Writing integration tests: This will involve writing integration tests to validate the behavior of the Raft implementation when running on a distributed system. The tests will simulate various network conditions and ensure that the implementation is resilient to different failure scenarios.

Documentation: This will involve documenting the implementation, including the architecture, interfaces, and design decisions. The documentation will help others understand the Raft implementation and how to use it.

The project will not involve developing a complete distributed system using Raft, but rather will focus on the implementation of the algorithm itself in Go. The project will provide learners with valuable experience in designing and implementing distributed systems using Go, as well as developing critical thinking and problem-solving skills.

Methodology

Raft is a distributed consensus algorithm that is designed to ensure that all nodes in a distributed system agree on the same state. It achieves this by electing a leader node, which coordinates the replication of the system's state across all nodes, and ensuring that the majority of nodes agree on the new state.

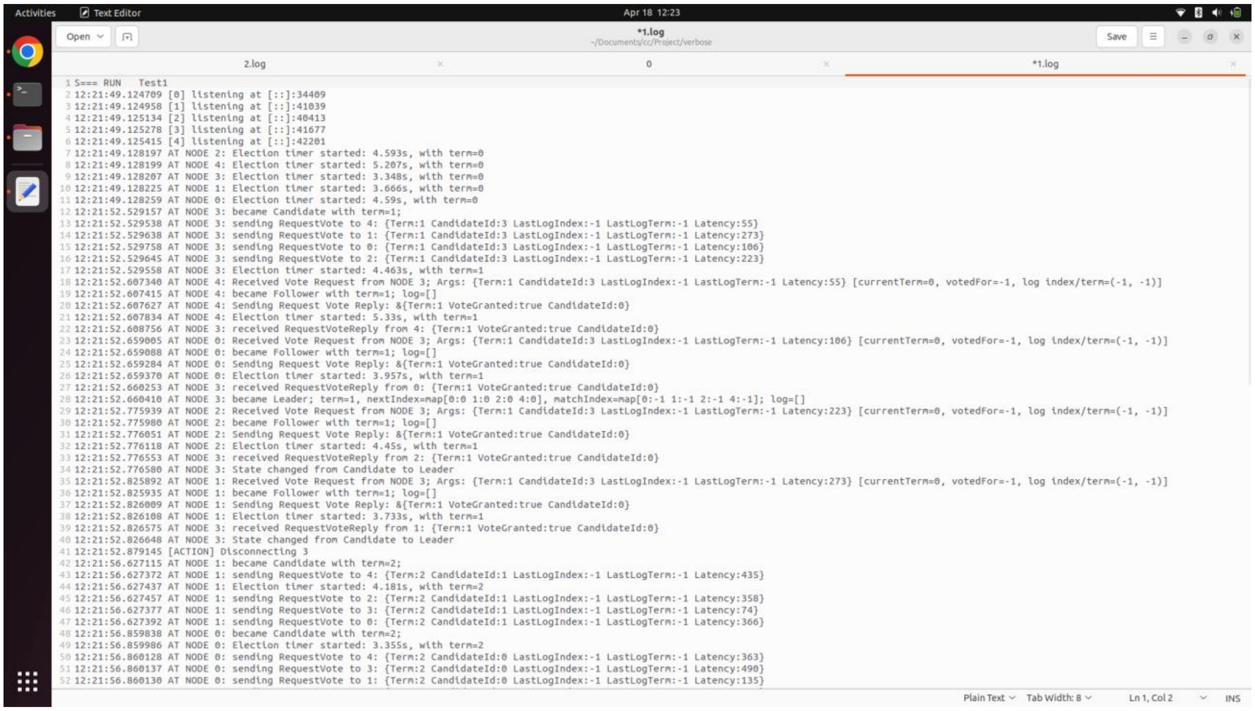
The Raft algorithm is divided into three key components:

Leader election: Raft uses a leader-follower model, where one node is designated as the leader, and the rest are followers. The leader is responsible for coordinating the replication of the system's state, while the followers only respond to requests from the leader. The leader is elected through a voting process, where nodes request votes from each other to become the leader.

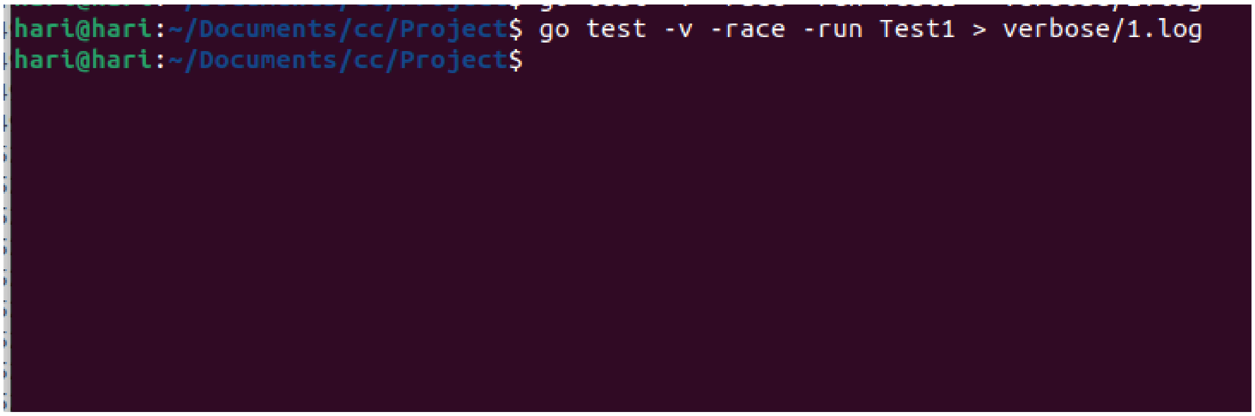
Log replication: Once a leader is elected, it is responsible for replicating the system's state across all nodes in the system. This is done through a process called log replication, where the leader appends new entries to its log and sends those entries to the followers. The followers then append those entries to their logs, and acknowledge that they have replicated them.

Safety: The Raft algorithm ensures safety by requiring that the majority of nodes agree on the new state before it can be committed. This ensures that even if some nodes fail or become disconnected, the system can continue to make progress as long as the majority of nodes are still available.

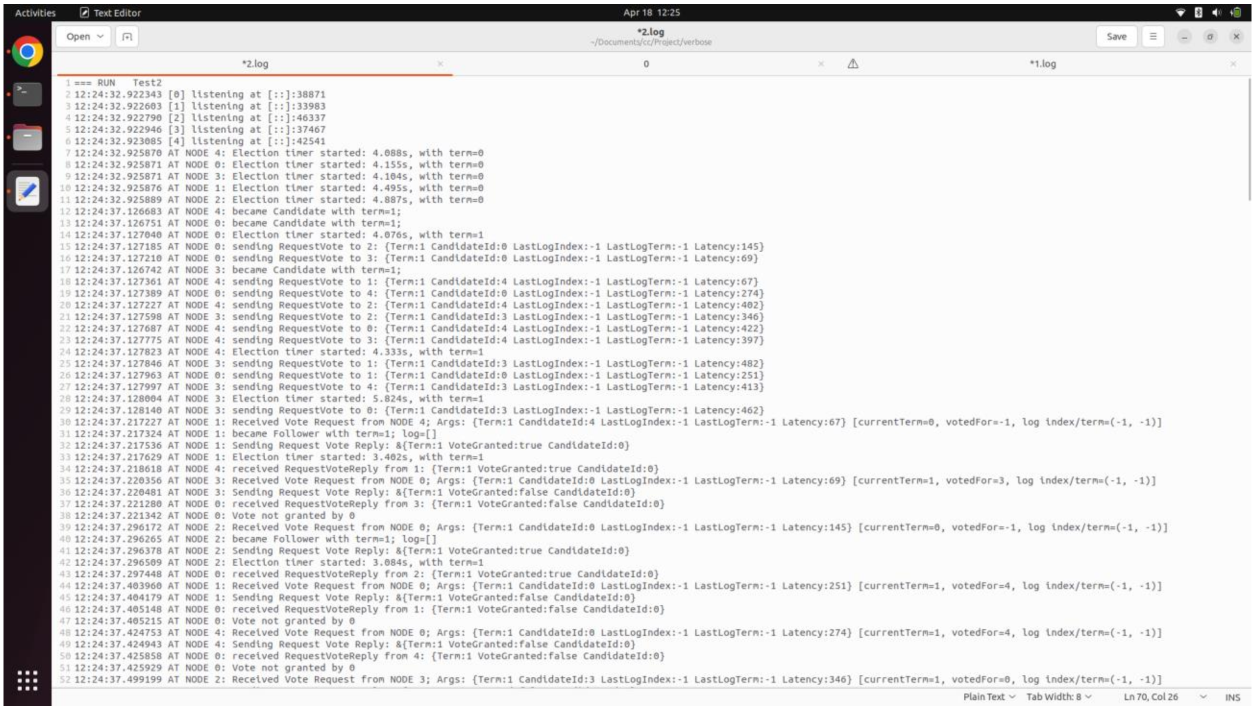
Testing



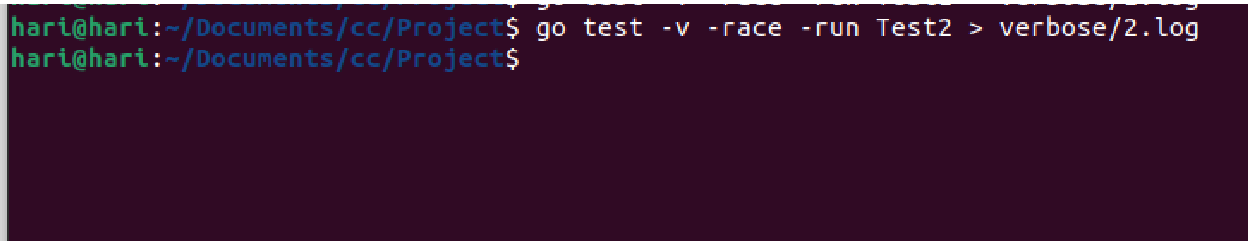
```
S=== RUN   Test1
12:21:49.124709 [0] listening at [::]:34409
12:21:49.124958 [1] listening at [::]:41039
12:21:49.125134 [2] listening at [::]:40413
12:21:49.125278 [3] listening at [::]:41677
12:21:49.125415 [4] listening at [::]:42201
12:21:49.128197 AT NODE 2: Election timer started: 4.593s, with term=0
12:21:49.128199 AT NODE 4: Election timer started: 5.207s, with term=0
12:21:49.128207 AT NODE 3: Election timer started: 3.348s, with term=0
12:21:49.128225 AT NODE 1: Election timer started: 3.666s, with term=0
12:21:49.128259 AT NODE 0: Election timer started: 4.59s, with term=0
12:21:52.529157 AT NODE 3: became Candidate with term=1;
12:21:52.529538 AT NODE 3: sending RequestVote to 4: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:55}
12:21:52.529638 AT NODE 3: sending RequestVote to 1: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:273}
12:21:52.529758 AT NODE 3: sending RequestVote to 0: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:106}
12:21:52.529645 AT NODE 3: sending RequestVote to 2: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:223}
12:21:52.529558 AT NODE 3: Election timer started: 4.463s, with term=1
12:21:52.607340 AT NODE 4: Received Vote Request from NODE 3; Args: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:55} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:21:52.607415 AT NODE 4: became Follower with term=1; log=[]
12:21:52.607627 AT NODE 4: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:21:52.607834 AT NODE 4: Election timer started: 5.33s, with term=1
12:21:52.608756 AT NODE 3: received RequestVoteReply from 4: {Term:1 VoteGranted:true CandidateId:0}
12:21:52.659005 AT NODE 0: Received Vote Request from NODE 3; Args: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:106} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:21:52.659088 AT NODE 0: became Follower with term=1; log=[]
12:21:52.659284 AT NODE 0: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:21:52.659370 AT NODE 0: Election timer started: 3.957s, with term=1
12:21:52.660253 AT NODE 3: received RequestVoteReply from 0: {Term:1 VoteGranted:true CandidateId:0}
12:21:52.660410 AT NODE 3: became Leader; term=1, nextIndex=map[0:0 1:0 2:0 4:0], matchIndex=map[0:-1 1:-1 2:-1 4:-1]; log=[]
12:21:52.775939 AT NODE 2: Received Vote Request from NODE 3; Args: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:223} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:21:52.775980 AT NODE 2: became Follower with term=1; log=[]
12:21:52.776051 AT NODE 2: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:21:52.776118 AT NODE 2: Election timer started: 4.45s, with term=1
12:21:52.776553 AT NODE 3: received RequestVoteReply from 2: {Term:1 VoteGranted:true CandidateId:0}
12:21:52.776580 AT NODE 3: State changed from Candidate to Leader
12:21:52.825892 AT NODE 1: Received Vote Request from NODE 3; Args: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:273} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:21:52.825935 AT NODE 1: became Follower with term=1; log=[]
12:21:52.826009 AT NODE 1: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:21:52.826108 AT NODE 1: Election timer started: 3.733s, with term=1
12:21:52.826575 AT NODE 3: received RequestVoteReply from 1: {Term:1 VoteGranted:true CandidateId:0}
12:21:52.826648 AT NODE 3: State changed from Candidate to Leader
12:21:52.879145 [ACTION] Disconnecting 3
12:21:56.627115 AT NODE 1: became Candidate with term=2;
12:21:56.627372 AT NODE 1: sending RequestVote to 4: {Term:2 CandidateId:1 LastLogIndex:-1 LastLogTerm:-1 Latency:435}
12:21:56.627437 AT NODE 1: Election timer started: 4.181s, with term=2
12:21:56.627457 AT NODE 1: sending RequestVote to 2: {Term:2 CandidateId:1 LastLogIndex:-1 LastLogTerm:-1 Latency:358}
12:21:56.627377 AT NODE 1: sending RequestVote to 3: {Term:2 CandidateId:1 LastLogIndex:-1 LastLogTerm:-1 Latency:74}
12:21:56.627392 AT NODE 1: sending RequestVote to 0: {Term:2 CandidateId:1 LastLogIndex:-1 LastLogTerm:-1 Latency:366}
12:21:56.859838 AT NODE 0: became Candidate with term=2;
12:21:56.859986 AT NODE 0: Election timer started: 3.355s, with term=2
12:21:56.860128 AT NODE 0: sending RequestVote to 4: {Term:2 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:363}
12:21:56.860137 AT NODE 0: sending RequestVote to 3: {Term:2 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:490}
12:21:56.860130 AT NODE 0: sending RequestVote to 1: {Term:2 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:135}
```



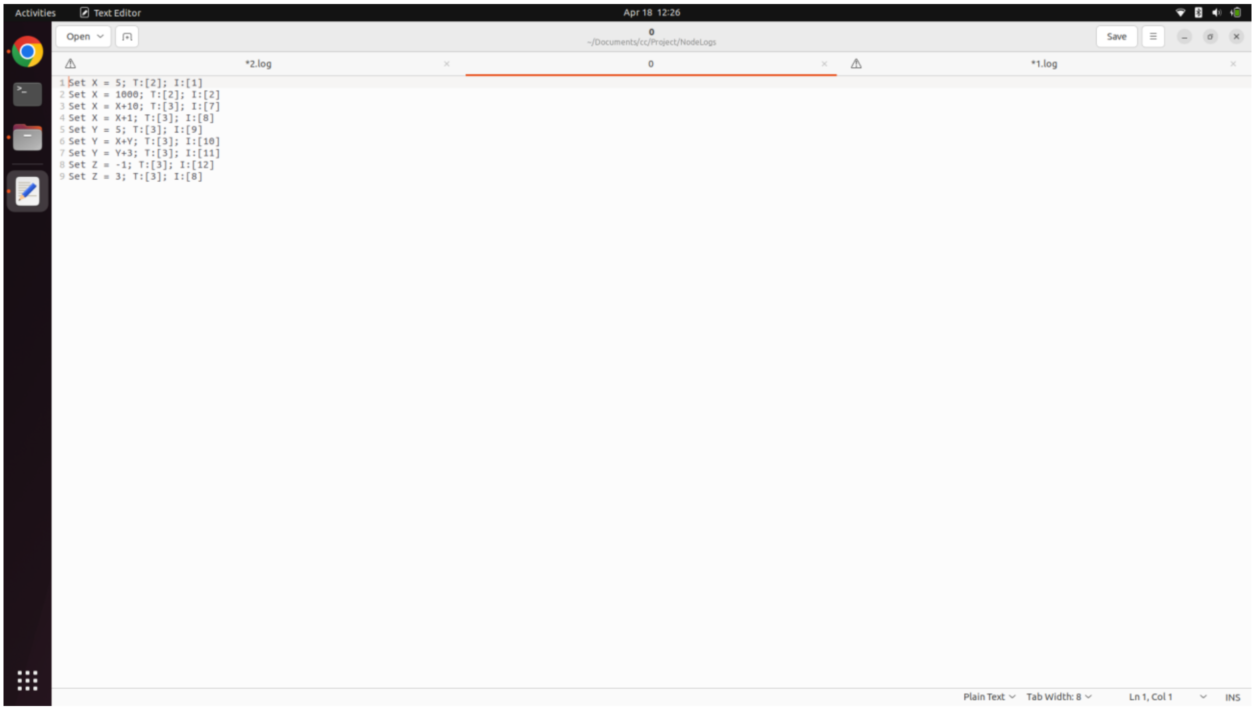
```
hari@hari:~/Documents/cc/Project$ go test -v -race -run Test1 > verbose/1.log
hari@hari:~/Documents/cc/Project$
```



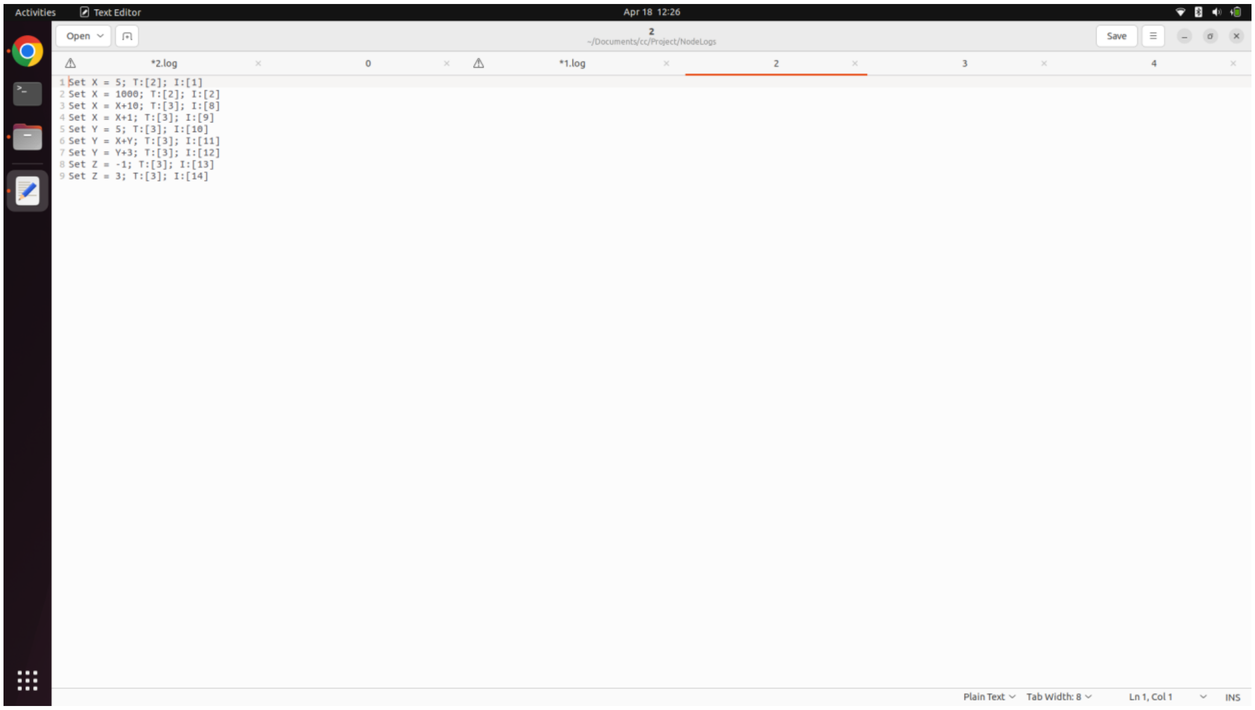
```
=== RUN   Test2
12:24:32.922343 [0] listening at [::]:38871
12:24:32.922603 [1] listening at [::]:33983
12:24:32.922790 [2] listening at [::]:46337
12:24:32.922946 [3] listening at [::]:37467
12:24:32.923085 [4] listening at [::]:42541
12:24:32.925870 AT NODE 4: Election timer started: 4.088s, with term=0
12:24:32.925871 AT NODE 0: Election timer started: 4.155s, with term=0
12:24:32.925871 AT NODE 3: Election timer started: 4.104s, with term=0
12:24:32.925876 AT NODE 1: Election timer started: 4.495s, with term=0
12:24:32.925889 AT NODE 2: Election timer started: 4.887s, with term=0
12:24:37.126683 AT NODE 4: became Candidate with term=1;
12:24:37.126751 AT NODE 0: became Candidate with term=1;
12:24:37.127040 AT NODE 0: Election timer started: 4.076s, with term=1
12:24:37.127185 AT NODE 0: sending RequestVote to 2: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:145}
12:24:37.127210 AT NODE 0: sending RequestVote to 3: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:69}
12:24:37.126742 AT NODE 3: became Candidate with term=1;
12:24:37.127361 AT NODE 4: sending RequestVote to 1: {Term:1 CandidateId:4 LastLogIndex:-1 LastLogTerm:-1 Latency:67}
12:24:37.127389 AT NODE 0: sending RequestVote to 4: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:274}
12:24:37.127227 AT NODE 4: sending RequestVote to 2: {Term:1 CandidateId:4 LastLogIndex:-1 LastLogTerm:-1 Latency:402}
12:24:37.127598 AT NODE 3: sending RequestVote to 2: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:346}
12:24:37.127687 AT NODE 4: sending RequestVote to 0: {Term:1 CandidateId:4 LastLogIndex:-1 LastLogTerm:-1 Latency:422}
12:24:37.127775 AT NODE 4: sending RequestVote to 3: {Term:1 CandidateId:4 LastLogIndex:-1 LastLogTerm:-1 Latency:397}
12:24:37.127823 AT NODE 4: Election timer started: 4.333s, with term=1
12:24:37.127846 AT NODE 3: sending RequestVote to 1: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:482}
12:24:37.127963 AT NODE 0: sending RequestVote to 1: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:251}
12:24:37.127997 AT NODE 3: sending RequestVote to 4: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:413}
12:24:37.128004 AT NODE 3: Election timer started: 5.824s, with term=1
12:24:37.128140 AT NODE 3: sending RequestVote to 0: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:462}
12:24:37.217227 AT NODE 1: Received Vote Request from NODE 4; Args: {Term:1 CandidateId:4 LastLogIndex:-1 LastLogTerm:-1 Latency:67} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:24:37.217324 AT NODE 1: became Follower with term=1; log=[]
12:24:37.217536 AT NODE 1: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:24:37.217629 AT NODE 1: Election timer started: 3.402s, with term=1
12:24:37.218618 AT NODE 4: received RequestVoteReply from 1: {Term:1 VoteGranted:true CandidateId:0}
12:24:37.220356 AT NODE 3: Received Vote Request from NODE 0; Args: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:69} [currentTerm=1, votedFor=3, log index/term=(-1, -1)]
12:24:37.220481 AT NODE 3: Sending Request Vote Reply: &{Term:1 VoteGranted:false CandidateId:0}
12:24:37.221280 AT NODE 0: received RequestVoteReply from 3: {Term:1 VoteGranted:false CandidateId:0}
12:24:37.221342 AT NODE 0: Vote not granted by 0
12:24:37.296172 AT NODE 2: Received Vote Request from NODE 0; Args: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:145} [currentTerm=0, votedFor=-1, log index/term=(-1, -1)]
12:24:37.296265 AT NODE 2: became Follower with term=1; log=[]
12:24:37.296378 AT NODE 2: Sending Request Vote Reply: &{Term:1 VoteGranted:true CandidateId:0}
12:24:37.296509 AT NODE 2: Election timer started: 3.084s, with term=1
12:24:37.297448 AT NODE 0: received RequestVoteReply from 2: {Term:1 VoteGranted:true CandidateId:0}
12:24:37.403960 AT NODE 1: Received Vote Request from NODE 0; Args: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:251} [currentTerm=1, votedFor=4, log index/term=(-1, -1)]
12:24:37.404179 AT NODE 1: Sending Request Vote Reply: &{Term:1 VoteGranted:false CandidateId:0}
12:24:37.405148 AT NODE 0: received RequestVoteReply from 1: {Term:1 VoteGranted:false CandidateId:0}
12:24:37.405215 AT NODE 0: Vote not granted by 0
12:24:37.424753 AT NODE 4: Received Vote Request from NODE 0; Args: {Term:1 CandidateId:0 LastLogIndex:-1 LastLogTerm:-1 Latency:274} [currentTerm=1, votedFor=4, log index/term=(-1, -1)]
12:24:37.424943 AT NODE 4: Sending Request Vote Reply: &{Term:1 VoteGranted:false CandidateId:0}
12:24:37.425858 AT NODE 0: received RequestVoteReply from 4: {Term:1 VoteGranted:false CandidateId:0}
12:24:37.425929 AT NODE 0: Vote not granted by 0
12:24:37.499199 AT NODE 2: Received Vote Request from NODE 3; Args: {Term:1 CandidateId:3 LastLogIndex:-1 LastLogTerm:-1 Latency:346} [currentTerm=1, votedFor=0, log index/term=(-1, -1)]
```



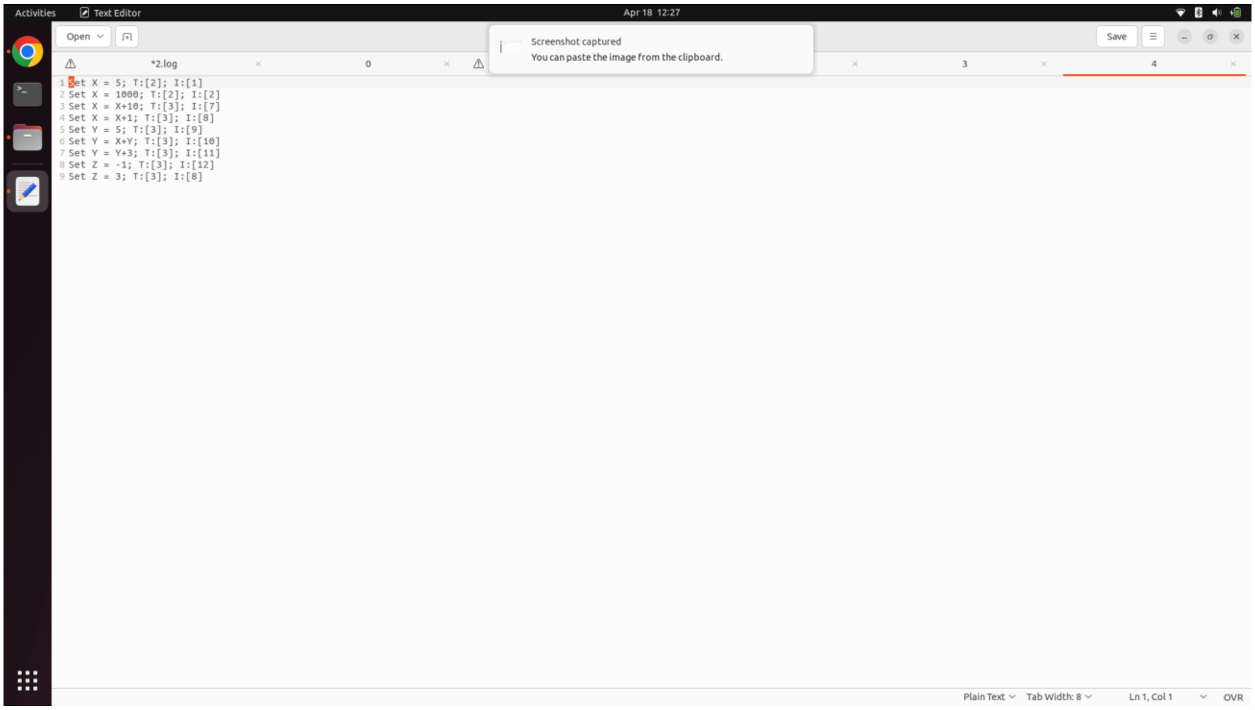
```
hari@hari:~/Documents/cc/Project$ go test -v -race -run Test2 > verbose/2.log
hari@hari:~/Documents/cc/Project$
```



```
Set X = 5; T:[2]; I:[1]
Set X = 1000; T:[2]; I:[2]
Set X = X+10; T:[3]; I:[7]
Set X = X+1; T:[3]; I:[8]
Set Y = 5; T:[3]; I:[9]
Set Y = X+Y; T:[3]; I:[10]
Set Y = Y+3; T:[3]; I:[11]
Set Z = -1; T:[3]; I:[12]
Set Z = 3; T:[3]; I:[8]
```

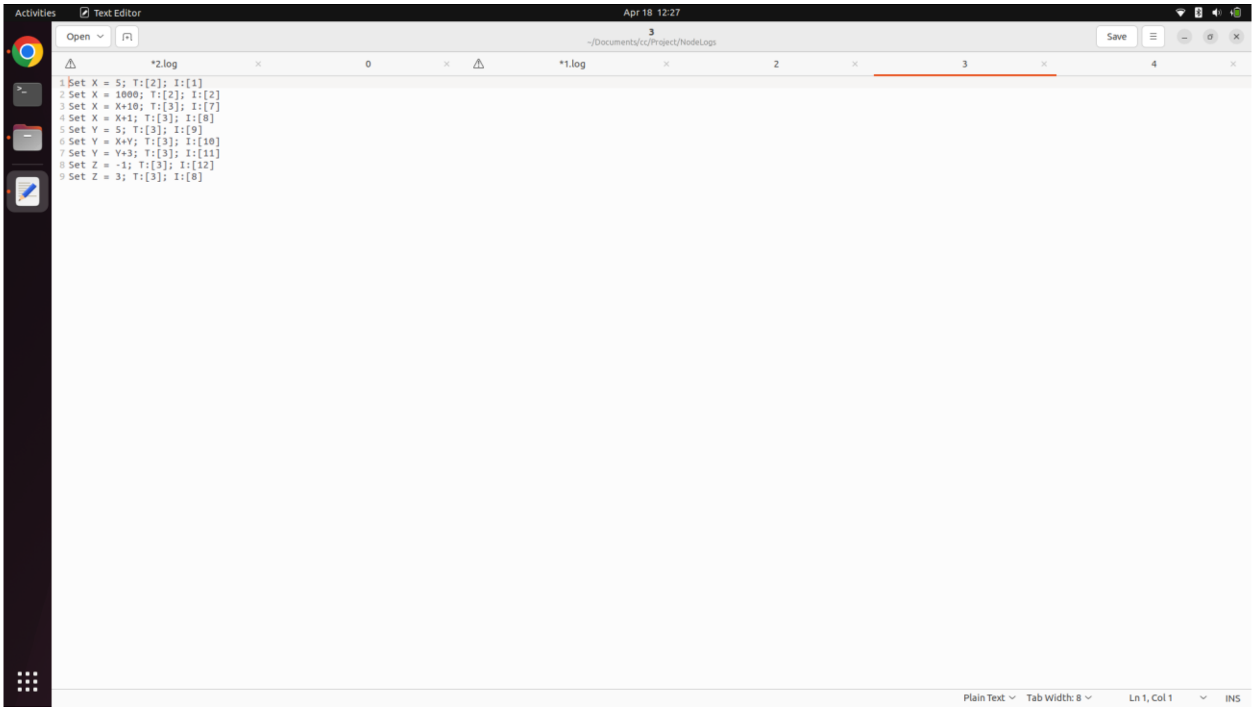


```
Set X = 5; T:[2]; I:[1]
Set X = 1000; T:[2]; I:[2]
Set X = X+10; T:[3]; I:[8]
Set X = X+1; T:[3]; I:[9]
Set Y = 5; T:[3]; I:[10]
Set Y = X+Y; T:[3]; I:[11]
Set Y = Y+3; T:[3]; I:[12]
Set Z = -1; T:[3]; I:[13]
Set Z = 3; T:[3]; I:[14]
```



Screenshot captured
You can paste the image from the clipboard.

```
Set X = 5; T:[2]; I:[1]
Set X = 1000; T:[2]; I:[2]
Set X = X+10; T:[3]; I:[7]
Set X = X+1; T:[3]; I:[8]
Set Y = 5; T:[3]; I:[9]
Set Y = X+Y; T:[3]; I:[10]
Set Y = Y+3; T:[3]; I:[11]
Set Z = -1; T:[3]; I:[12]
Set Z = 3; T:[3]; I:[8]
```



```
Set X = 5; T:[2]; I:[1]
Set X = 1000; T:[2]; I:[2]
Set X = X+10; T:[3]; I:[7]
Set X = X+1; T:[3]; I:[8]
Set Y = 5; T:[3]; I:[9]
Set Y = X+Y; T:[3]; I:[10]
Set Y = Y+3; T:[3]; I:[11]
Set Z = -1; T:[3]; I:[12]
Set Z = 3; T:[3]; I:[8]
```

Results and Conclusions

The results and conclusion of the project would depend on the success of the implementation and the goals set for the project. Here are some potential results and conclusions:

Results:

Successful implementation of the Raft algorithm in Go

Comprehensive test suite to validate the implementation

Documentation explaining the design and implementation of the Raft algorithm in Go

Gained understanding of distributed systems and Go programming

Conclusion:

The Raft algorithm is a complex distributed consensus algorithm that requires careful implementation and testing.

Go is a well-suited programming language for developing distributed systems due to its support for concurrency and its efficient garbage collection.

Developing an implementation of the Raft algorithm in Go provides valuable experience in designing and implementing distributed systems, as well as developing critical thinking and problem-solving skills.

The implementation of the Raft algorithm in Go provides a solid foundation for building a distributed system that requires consensus.