Homework 1

Student ID: 1004875

Name: Xiang Siqi

Table of Content

Introduction

This homework contains two assignments, each focusing on specific concepts in distributed systems. This README will provide you with a clear understanding of this project implementation, how to compile and execute the assignments, the different types of parameters, and what each assignment entails. Each assignment's implementation will be explained in detail.

The structure of this homework is:

```
├─hw1
|   ├─readme.md
|   ├─logger
|   |   └logger.go
|   ├─assignment2
|   |   ├─main.go
|   |   ├─doc
|   |   |   └fig1_message.jpg
|   |   ├─bully
|   |   |   ├─message.go
|   |   |   ├─server.go
|   |   |   └util.go
|   ├─assignment1
|   |   ├─main.go
|   |   ├─vectorclock
|   |   |   ├─client.go
|   |   |   ├─message.go
|   |   |   └server.go
|   |   ├─lamportclock
|   |   |   ├─client.go
|   |   |   ├─message.go
|   |   |   └server.go
|   |   ├─doc
|   |   |   ├─fig1_lamport_architecture.jpg
|   |   |   └fig2_vector_architecture.jpg
```

Compilation and Execution

To run each assignment, following these steps:

Assignment 1

1. Navigate to the `hw1/assignment1` directory.
2. Open a terminal in this directory.
3. Run the following command to start: `go run main.go`.
4. Note that you can change the type of clock by setting `algorithm` to either `VECTOR_CLOCK` or `LAMPORT_CLOCK` in `main.go`.

Assignment 2

1. Navigate to the `hw1/assignment2` directory.
2. Open a terminal in this directory.
3. Run the following command to start: `go run main.go`.

In this project, all logs generated during the execution will be written and saved in a file called `assignment{1/2}.log` under `hw1` folder. There will be no log printed out in the command shell during execution.

Assignment 1

Objects

1. Simulate the behavior of both the server and the registered clients via GO routines.
2. Use Lamport's logical clock to determine a total order of all the messages received at all the registered clients. Subsequently, present (i.e., print) this order for all registered clients to know the order in which the messages should be read.
3. Use Vector clock to redo the assignment. Implement the detection of causality violation and print any such detected causality violation.

Implementations

The configuration of this assignment can be customized through the following parameters defined in `main.go` (line 26 - line 28):

- `numOfClients`: Specifies the number of clients participating in this simulation. Set to 10 by default.

- `timeInterval`: Determines the time interval for a client to send a message to the server, measured in second. The default value is 1.
- `algorithm`: You can choose the clock algorithm to use, either `LAMPORT_CLOCK`, or `VECTOR_CLOCK`. The default setting is `VECTOR_CLOCK`.

You can experiment with different parameter values to observe the simulation's behavior.

Lamport Clock

The architecture of Lamport Clock simulation is illustrated in Figure 1. The Server and Clients are simulated via GO routines. Note that in this implementation, Each of the server and clients has only one channel.

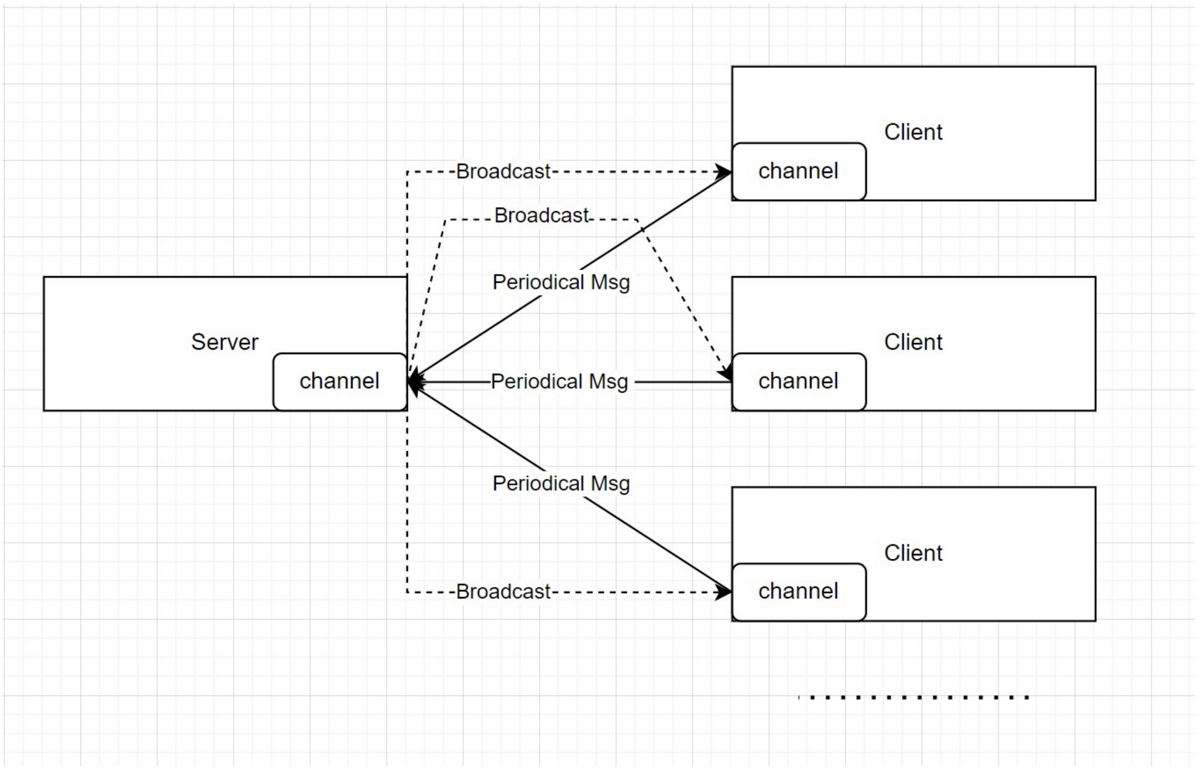


Figure 1: Lamport Clock Architecture

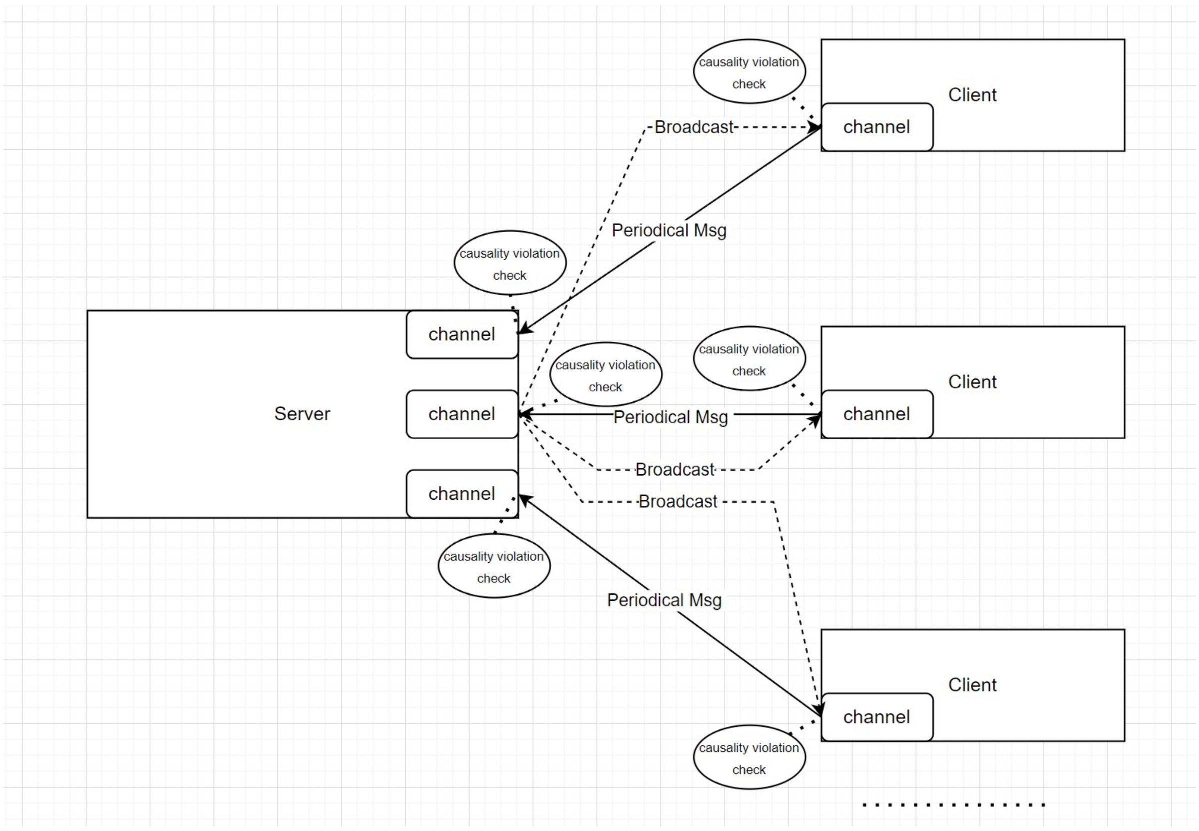
When you run `go run main.go` with the `LAMPORT_CLOCK` algorithm, the program will generate log outputs in `assignment_1.log`. Each log entry includes information about the printer, its current clock value, and the operation being performed. Here is a sample log.

```
assignment 1: 2023/10/18 15:13:49 [ Server ] -- Clock 0 -- Server starts
listening
assignment 1: 2023/10/18 15:13:49 [Client 0] -- Clock 0 -- client 0 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 1] -- Clock 0 -- client 1 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 2] -- Clock 0 -- client 2 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 3] -- Clock 0 -- client 3 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 4] -- Clock 0 -- client 4 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 5] -- Clock 0 -- client 5 starts
listening and sends periodical messages
```

```
assignment 1: 2023/10/18 15:13:49 [Client 6] -- Clock 0 -- client 6 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 7] -- Clock 0 -- client 7 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 8] -- Clock 0 -- client 8 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:49 [Client 9] -- Clock 0 -- client 9 starts
listening and sends periodical messages
assignment 1: 2023/10/18 15:13:50 [Client 9] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 3] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 2 -- receive message from
client 9
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 3 -- discard message from
client 9
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 4 -- receive message from
client 3
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 5 -- broadcast message from
client 3
assignment 1: 2023/10/18 15:13:50 [Client 5] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 8] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 7] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 6 -- receive message from
client 5
assignment 1: 2023/10/18 15:13:50 [Client 6] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 2] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 1] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 0] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 4] -- Clock 1 -- send message to server
assignment 1: 2023/10/18 15:13:50 [Client 9] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 0] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 1] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 2] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 4] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 5] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 6] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 7] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [Client 8] -- Clock 6 -- receive server's
broadcast message, originally from client 3
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 7 -- broadcast message from
client 5
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 8 -- receive message from
client 8
assignment 1: 2023/10/18 15:13:50 [ Server ] -- Clock 9 -- broadcast message from
client 8
assignment 1: 2023/10/18 15:13:50 [Client 3] -- Clock 8 -- receive server's
broadcast message, originally from client 5
```

Vector Clock

The architecture of Vector Clock simulation is illustrated in Figure 2. The Server and Clients are simulated via GO routines. Note that in this implementation, the server will create a channel for each client.



Server
channel
channel
channel
Client
channel
Client
channel
Client
channel
causality violation check
causality violation check
causality violation check
causality violation check
causality violation check
causality violation check
causality violation check
Broadcast
Broadcast
Broadcast
Periodical Msg
Periodical Msg
Periodical Msg
....

Figure 2: Vector Clock Architecture

When you run `go run main.go` with the `VECTOR_CLOCK` algorithm, the program will generate log outputs in `assignment_1.log`. Each log entry includes information about the printer, its current vector clock, and the operation being performed. Here is a sample log.

```
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  1 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
```

```
assignment 1: 2023/10/18 15:47:50 [Server Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- Server starts listening
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  2 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  3 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  4 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  5 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  6 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  7 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  8 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client  9 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:50 [Client Activate] -- Clock [0 0 0 0 0 0 0 0 0 0
0] -- client 10 starts listening and sends periodical messages
assignment 1: 2023/10/18 15:47:51 [Client  4] -- Clock [0 0 0 0 1 0 0 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  5] -- Clock [0 0 0 0 0 1 0 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  1] -- Clock [0 1 0 0 0 0 0 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  2] -- Clock [0 0 1 0 0 0 0 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  8] -- Clock [0 0 0 0 0 0 0 0 1 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client 10] -- Clock [0 0 0 0 0 0 0 0 0 0 1] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  9] -- Clock [0 0 0 0 0 0 0 0 0 1 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  6] -- Clock [0 0 0 0 0 0 1 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [Client  7] -- Clock [0 0 0 0 0 0 0 1 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [1 0 0 0 1 0 0 0 0 0 0] --
receive message from client  4
assignment 1: 2023/10/18 15:47:51 [Client  3] -- Clock [0 0 0 1 0 0 0 0 0 0 0] --
send message to server
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [2 0 0 0 1 0 0 0 0 0 0] --
discard message from client 4
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [3 0 0 0 1 1 0 0 0 0 0] --
receive message from client  5
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [4 0 0 0 1 1 0 0 0 0 0] --
broadcast message from client  5
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [5 1 0 0 1 1 0 0 0 0 0] --
receive message from client  1
assignment 1: 2023/10/18 15:47:51 [Client  7] -- Clock [4 0 0 0 1 1 0 2 0 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  2] -- Clock [4 0 2 0 1 1 0 0 0 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  6] -- Clock [4 0 0 0 1 1 2 0 0 0 0] --
receive  5's message
```

```
assignment 1: 2023/10/18 15:47:51 [Client  3] -- Clock [4 0 0 2 1 1 0 0 0 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  4] -- Clock [4 0 0 0 2 1 0 0 0 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  1] -- Clock [4 2 0 0 1 1 0 0 0 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  8] -- Clock [4 0 0 0 1 1 0 0 2 0 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [6 1 0 0 1 1 0 0 0 0 0] --
discard message from client 1
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [7 1 1 0 1 1 0 0 0 0 0] --
receive message from client  2
assignment 1: 2023/10/18 15:47:51 [ Server  ] -- Clock [8 1 1 0 1 1 0 0 0 0 0] --
broadcast message from client  2
assignment 1: 2023/10/18 15:47:51 [Client  8] -- Clock [8 1 1 0 1 1 0 0 3 0 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client  9] -- Clock [4 0 0 0 1 1 0 0 0 2 0] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client  9] -- Clock [8 1 1 0 1 1 0 0 0 3 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client  1] -- Clock [8 3 1 0 1 1 0 0 0 0 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client  3] -- Clock [8 1 1 3 1 1 0 0 0 0 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client  4] -- Clock [8 1 1 0 3 1 0 0 0 0 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client  5] -- Clock [8 1 1 0 1 2 0 0 0 0 0] --
receive  2's message
assignment 1: 2023/10/18 15:47:51 [Client 10] -- Clock [4 0 0 0 1 1 0 0 0 0 2] --
receive  5's message
assignment 1: 2023/10/18 15:47:51 [Client 10] -- Clock [8 1 1 0 1 1 0 0 0 0 3] --
receive  2's message
```

This implementation does not contain any causality violations by default. This is because The FIFO (First-In-First-Out) nature of GO channels. It ensures that messages are received in the order they are sent, thereby preventing any causality violations.

However, to introduce and demonstrate the detection of causality violations, I have implemented a `MadlyActive` method. This method instructs the client to intentionally send two messages with causality violations to the server. As a result, the server detects and handles these violations. Below, you can find the relevant code for this feature.

```go
c.incrementClock()
clockSmall := make([]int, len(c.vectorClock))
copy(clockSmall, c.vectorClock)
msgSmall := Message{senderId: c.Id, vectorClock: clockSmall} // construct a
message with a smaller vector clock

c.incrementClock()
clockLarge := make([]int, len(c.vectorClock))
copy(clockLarge, c.vectorClock)
msgLarge := Message{senderId: c.Id, vectorClock: clockLarge} // construct a
message with a larger vector clock
c.mu.Unlock()
```

```
// send the messages in a wrong order
c.sendMsg(msgLarge)
c.sendMsg(msgSmall)
```

There is an implement of "madly launching" in the `main.go`. You have to comment the normal launching code, and uncomment the madly launching code to see the simulation.

You will see something like:

```
assignment 1: 2023/10/18 16:04:25 [Potential Causality Violation Detected on
Server when receiving  2's message]
-- Vector Clock on Server -- [14 3 4 3 0 1 0 0 0 1 2]
-- Vector Clock from client  2-- [4 0 3 0 0 0 0 0 0 0 2]
```

This shows the causality violation detected by the server in vector clock algorithm.

Assignment 2

Objects

1. Use Bully algorithm to implement a working version of replica synchronization.
    - Each replica maintains some data structure, which are periodically synchronized with the coordinator.
    - The coordinator initiates the synchronization process by sending message to all other machines.
    - Upon receiving the message from the coordinator, each machine updates its local version of the data structure with the coordinator’s version.
    - The coordinator, being an arbitrary machine in the network, is subject to fault. Thus,
        - A new coordinator is chosen by the Bully algorithm.
        - You can assume a fixed timeout to simulate the behavior of detecting a fault. The objective is to have a consensus across all machines (simulated by GO routines) in terms of the newly elected coordinator.
2. Simulate the worst-case and the best-case situation as discussed in class.
3. Consider the case where a GO routine fails during the election, yet the routine was alive when the election started.
    - The newly elected coordinator fails while announcing that it has won the election to all nodes.
    - The failed node is not the newly elected coordinator.
4. Multiple GO routines start the election process simultaneously.
5. An arbitrary node, either the coordinator or non-coordinator silently leaves the network.

Implementation

Message

Messages play an important role in communication among servers (GO routines). There are six distinct message types in this implementation, each serving specific purposes in the synchronization and coordination processes.

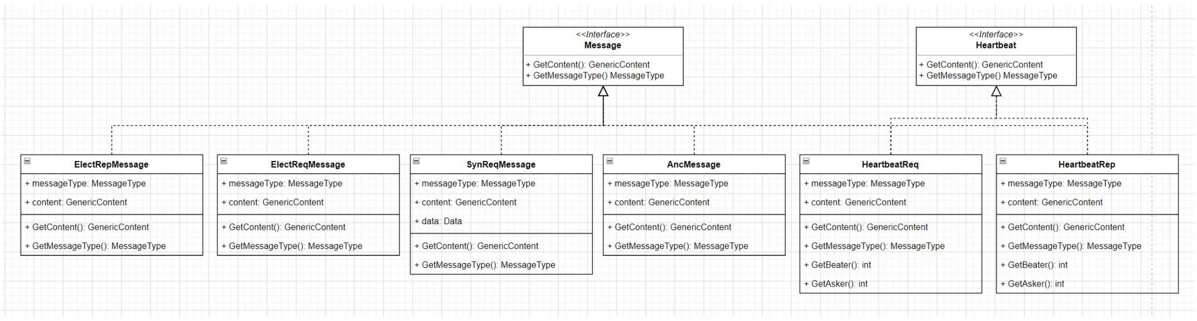

<<Interface>>
Message
+ GetContent(): GenericContent
+ GetMessageType() MessageType
<<Interface>>
Heartbeat
+ GetContent(): GenericContent
+ GetMessageType() MessageType
ElectRepMessage
+ messageType: MessageType
+ content: GenericContent
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
ElectReqMessage
+ messageType: MessageType
+ content: GenericContent
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
SynReqMessage
+ messageType: MessageType
+ content: GenericContent
+ data: Data
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
AncMessage
+ messageType: MessageType
+ content: GenericContent
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
HeartbeatReq
+ messageType: MessageType
+ content: GenericContent
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
+ GetBeater(): int
+ GetAsker(): int
HeartbeatRep
+ messageType: MessageType
+ content: GenericContent
+ GetContent(): GenericContent
+ GetMessageType(): MessageType
+ GetBeater(): int
+ GetAsker(): int

Figure 3: Message UML Diagram

Server

The central component in this simulation is the server. Each server contains the following fields.

Field	Type	Explanation
id	int	the server's unique identifier
role	Role	The server's role in the network (COORDINATOR or WORKER).
status	Status	The current status of the server, which can change due to failures (ALVE or DOWN)
Cluster	Cluster	The group of servers
msgChannel	chan Message	A channel used to receive various message types, including SynReqMessage, ElectReqMessage, ElectRepMessage, and AncMessage
heartbeatChannel	chan Message	A channel used to receiving heartbeat requests and replies for failure check
data	Data	The data stored in this server
election	Election	State information related to the election process
heartbeatFrequency	int	The frequency at which heartbeats are exchanged between the coordinator and other servers, measured in second
replyTimeout	int	The maximum time allowed for receiving replies, measured in second
electionTimeout	int	The timeout duration from election to announcement, measured in second

Field	Type	Explanation
syncFrequency	int	The frequency at which data synchronization from the coordinator occurs, measured in second
failWhileAnnounce	bool	A flag indicating whether this server will fail while announcing itself as the coordinator
failWhileElection	bool	A flag indicating whether this server will fail during the election process

A server is managed by three separate GO routines, each responsible for distinct tasks. These routines collectively handle message reception, the heartbeat mechanism, and local work, as described below:

- **Message Handling**: One GO routine listens to the `msgChannel` and processes received messages. This includes managing both election and synchronization messages and process.
- **Heartbeat Mechanism**: Another GO routine is dedicated to the heartbeat mechanism. In a normal server, it periodically sends heartbeat requests to the coordinator. The coordinator, if alive, responds with a heartbeat message. This mechanism ensures the continuous monitoring of the server's status within the cluster.
- **Local Work Simulation**: The third GO routine manages local work. This work is simulated by modifying a local time value, introducing a random element to add diversity among different servers.

To activate a server, the `Activate` function is employed:

```go
// Activate the server
func (s *Server) Activate() {
    go s.Listen()    // listen to the msgChannel
    go s.Heartbeat() // heartbeat
    go s.Work()      // update local data
}
```

For a deeper look into the details, feel free to check out the code; it's well annotated with comments.

Simulation

Same as assignment 1, you can initiate the simulation by running `go run main.go` command within the `hw1/assignment2` directory. A log file named assignment_2.log will be created under `hw1` folder.

To tailor the simulation, there are some configuration parameters that can be customized in the `main.go` file. These parameters include:

- `numOfServers`: Number of servers in the cluster. By default, it is set to 5.
- `heartbeatFrequency`: The frequency at which heartbeat requests are sent from a server to the coordinator, measured in second. The default value is 10 seconds.
- `electionTimeout`: The timeout duration from election to announcement, measured in second. By default, this is configured for an 8-second duration.

- `replyTimeout`: The maximum time allowed for receiving replies, measured in second. The default value is 4 seconds.
- `syncFrequency`: The frequency at which data synchronization from the coordinator occurs, measured in second. The default is set to 10 seconds.

These parameter options provide you with the capability to fine-tune the simulation as needed. Of course, except for nodes failures, which will be introduced later.

Simulation 1. Normal Run

Starting from `main.go` line 41, you can configure various simulation scenarios.

The first scenario involves the activities of five servers. In this setup, the servers initiate an election process to select a coordinator and subsequently synchronize their local data. This scenario represents the typical and expected behavior of the simulation.

```
// 1. Normal simulation
for _, server := range servers {
    server.Activate()
}
```

Here is a sample log output. It offers a comprehensive overview of the activities within the simulation. It covers the entire sequence of events, starting from server activation, moving through the election and announcement phases, and encompassing synchronization and heartbeat checks.

```
assignment 2: 2023/10/19 00:11:53 [Server Activate] Server 2 is activated
assignment 2: 2023/10/19 00:11:53 [Server Activate] Server 3 is activated
assignment 2: 2023/10/19 00:11:53 [Server Activate] Server 4 is activated
assignment 2: 2023/10/19 00:11:53 [Server Activate] Server 1 is activated
assignment 2: 2023/10/19 00:11:53 [Server Activate] Server 0 is activated
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 2, replying a
no message
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 4, stop electing 2
assignment 2: 2023/10/19 00:12:03 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 3, replying a
no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 2, replying a
no message
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 2
assignment 2: 2023/10/19 00:12:03 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 2, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 2] Election request from 1, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 2, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 3, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 4, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 2, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 3, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 4, stop electing 2
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 00:12:03 [Server 0] Sending election message to 1
assignment 2: 2023/10/19 00:12:03 [Server 0] Sending election message to 2
assignment 2: 2023/10/19 00:12:03 [Server 0] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 0] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 0, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 0] Reply from 4, stop electing 0
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 2
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 1] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 1, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 4, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 1] Election request from 0, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 0] Reply from 1, stop electing 0
assignment 2: 2023/10/19 00:12:03 [Server 2] Election request from 1, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 3, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 4] Election request from 2, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 00:12:03 [Server 2] Election request from 0, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 0] Reply from 2, stop electing 0
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 2, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 1, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 1] Reply from 3, stop electing 1
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 2, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 3] Election request from 0, replying a no message
assignment 2: 2023/10/19 00:12:03 [Server 0] Reply from 3, stop electing 0
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 4, stop electing 2
assignment 2: 2023/10/19 00:12:03 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 00:12:11 [Server 4 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 00:12:11 [Server 3] Announcement received from 4
assignment 2: 2023/10/19 00:12:11 [Server 3] Set coordinator to 4
assignment 2: 2023/10/19 00:12:11 [Server 0] Announcement received from 4
assignment 2: 2023/10/19 00:12:11 [Server 0] Set coordinator to 4
assignment 2: 2023/10/19 00:12:11 [Server 1] Announcement received from 4
assignment 2: 2023/10/19 00:12:11 [Server 1] Set coordinator to 4
assignment 2: 2023/10/19 00:12:11 [Server 2] Announcement received from 4
assignment 2: 2023/10/19 00:12:11 [Server 2] Set coordinator to 4
assignment 2: 2023/10/19 00:12:11 [Server 4 - Coordinator] Synchronize to value
3204322506, sending to 1
assignment 2: 2023/10/19 00:12:11 [Server 1] Synchronization request from 4,
current local time 3222928979, set localtime to 3204322506
assignment 2: 2023/10/19 00:12:11 [Server 4 - Coordinator] Synchronize to value
3204322506, sending to 0
assignment 2: 2023/10/19 00:12:11 [Server 0] Synchronization request from 4,
current local time 3217338271, set localtime to 3204322506
assignment 2: 2023/10/19 00:12:11 [Server 4 - Coordinator] Synchronize to value
3204322506, sending to 2
assignment 2: 2023/10/19 00:12:11 [Server 2] Synchronization request from 4,
current local time 3223042749, set localtime to 3204322506
assignment 2: 2023/10/19 00:12:11 [Server 4 - Coordinator] Synchronize to value
3204322506, sending to 3
assignment 2: 2023/10/19 00:12:11 [Server 3] Synchronization request from 4,
current local time 3223020483, set localtime to 3204322506
assignment 2: 2023/10/19 00:12:13 [Server 2] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:13 [Server 4 - Coordinator] Sending Heartbeat to 2
assignment 2: 2023/10/19 00:12:13 [Server 1] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:13 [Server 3] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:13 [Server 4 - Coordinator] Sending Heartbeat to 3
assignment 2: 2023/10/19 00:12:13 [Server 4 - Coordinator] Sending Heartbeat to 1
assignment 2: 2023/10/19 00:12:13 [Server 0] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:13 [Server 4 - Coordinator] Sending Heartbeat to 0
assignment 2: 2023/10/19 00:12:21 [Server 4 - Coordinator] Synchronize to value
4619700955, sending to 1
assignment 2: 2023/10/19 00:12:21 [Server 4 - Coordinator] Synchronize to value
4619700955, sending to 0
assignment 2: 2023/10/19 00:12:21 [Server 4 - Coordinator] Synchronize to value
4619700955, sending to 2
assignment 2: 2023/10/19 00:12:21 [Server 0] Synchronization request from 4,
current local time 4994117917, set localtime to 4619700955
assignment 2: 2023/10/19 00:12:21 [Server 2] Synchronization request from 4,
current local time 5004099827, set localtime to 4619700955
assignment 2: 2023/10/19 00:12:21 [Server 4 - Coordinator] Synchronize to value
4619700955, sending to 3
assignment 2: 2023/10/19 00:12:21 [Server 3] Synchronization request from 4,
current local time 4997404028, set localtime to 4619700955
assignment 2: 2023/10/19 00:12:21 [Server 1] Synchronization request from 4,
current local time 4985505345, set localtime to 4619700955
assignment 2: 2023/10/19 00:12:23 [Server 2] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:23 [Server 3] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:23 [Server 4 - Coordinator] Sending Heartbeat to 3
assignment 2: 2023/10/19 00:12:23 [Server 1] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:12:23 [Server 4 - Coordinator] Sending Heartbeat to 1
```

Simulation 2. Worst-case Election Scenario

In this scenario, I simulate the worst-case election situation, where the node with the smallest ID initiates the election process. To achieve this, I simulate it by activating server 1 after the first announcement, necessitating a new round of elections.

```
// 2.1 Simulate worst-case
for index, server := range servers {
    if index != 0 {
        server.Activate()
    }
}

for {
    if servers[1].Cluster.GetCoordinator() != nil {
        servers[0].PleaseIgnorePreviousAnnouncement()
        break
    }
}
```

I need to explain the `PleaseIgnorePreviousAnnouncement` method here. It instructs a server to disregard any previous announcements. This is not caused by any limitation in the system's ability to handle the addition of new servers at a later stage. Quite the opposite, when a server joins the network later, it automatically receives all previous messages that were sent to its designated channel (because of the channel buffer). If, for instance, a server with a larger Id previously sent an announcement, the new server simply accepts this announcement instead of initiating a fresh election. Furthermore, even in cases where the coordinator has become inactive, it can be detected through the ongoing heartbeat checks. That is why here, to ask server 0 to start a new election, it has to ignore the previous announcement and start a new round of election.

The critical section of the log is shown below, starting from the activation of server 0, until the announcement of server 4 (the server with the highest id).

```
assignment 2: 2023/10/19 00:30:20 [Server Activate] Server 0 is activated
assignment 2: 2023/10/19 00:30:20 [Server 0] Synchronization request from 4,
current local time 14980, set localtime to 2425050040
assignment 2: 2023/10/19 00:30:20 [Server 0] Sending election message to 1
assignment 2: 2023/10/19 00:30:20 [Server 0] Sending election message to 2
assignment 2: 2023/10/19 00:30:20 [Server 0] Sending election message to 3
assignment 2: 2023/10/19 00:30:20 [Server 0] Sending election message to 4
assignment 2: 2023/10/19 00:30:20 [Server 1] Sending election message to 2
assignment 2: 2023/10/19 00:30:20 [Server 1] Sending election message to 3
assignment 2: 2023/10/19 00:30:20 [Server 1] Sending election message to 4
assignment 2: 2023/10/19 00:30:20 [Server 4] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 1] Reply from 4, stop electing 1
assignment 2: 2023/10/19 00:30:20 [Server 4] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 3] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 1] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 00:30:20 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 00:30:20 [Server 4] Election request from 2, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 2] Reply from 4, stop electing 2
assignment 2: 2023/10/19 00:30:20 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 00:30:20 [Server 3] Election request from 2, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 00:30:20 [Server 2] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 1] Reply from 2, stop electing 1
assignment 2: 2023/10/19 00:30:20 [Server 3] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 1] Reply from 3, stop electing 1
assignment 2: 2023/10/19 00:30:20 [Server 2] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 0] Reply from 4, stop electing 0
assignment 2: 2023/10/19 00:30:20 [Server 0] Reply from 3, stop electing 0
assignment 2: 2023/10/19 00:30:20 [Server 0] Reply from 1, stop electing 0
assignment 2: 2023/10/19 00:30:20 [Server 0] Reply from 2, stop electing 0
assignment 2: 2023/10/19 00:30:20 [Server 4] Election request from 3, replying a
no message
assignment 2: 2023/10/19 00:30:20 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 00:30:22 [Server 1] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:30:22 [Server 2] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:30:22 [Server 3] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:30:28 [Server 4 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 00:30:28 [Server 0] Announcement received from 4
assignment 2: 2023/10/19 00:30:28 [Server 0] Set coordinator to 4
assignment 2: 2023/10/19 00:30:28 [Server 3] Announcement received from 4
assignment 2: 2023/10/19 00:30:28 [Server 3] Set coordinator to 4
assignment 2: 2023/10/19 00:30:28 [Server 1] Announcement received from 4
assignment 2: 2023/10/19 00:30:28 [Server 1] Set coordinator to 4
assignment 2: 2023/10/19 00:30:28 [Server 2] Announcement received from 4
assignment 2: 2023/10/19 00:30:28 [Server 2] Set coordinator to 4
```

Simulation 2. Best-case Election Scenario

In this scenario, I simulate the best-case election situation, where the node with the highest ID initiates the election process. To recreate this scenario, I activate the server with the highest ID after the initial announcement. This deliberate action triggers a new round of elections.

```go
// 2.2 Simulate best-case
for index, server := range servers {
    if index != len(servers)-1 {
        server.Activate()
    }
}

for {
    if servers[0].Cluster.GetCoordinator() != nil {
        servers[len(servers)-1].Activate()
        break
    }
}
```

It a very simple case. Here shows the log from the activation of server 4 to its declaration.

```
assignment 2: 2023/10/19 00:46:25 [Server Activate] Server 4 is activated
assignment 2: 2023/10/19 00:46:25 [Server 3 - Coordinator] Synchronize to value
2391288529, sending to 1
assignment 2: 2023/10/19 00:46:25 [Server 1] Synchronization request from 3,
current local time 2380090415, set localtime to 2391288529
assignment 2: 2023/10/19 00:46:25 [Server 3 - Coordinator] Synchronize to value
2391288529, sending to 0
assignment 2: 2023/10/19 00:46:25 [Server 0] Synchronization request from 3,
current local time 2369495018, set localtime to 2391288529
assignment 2: 2023/10/19 00:46:25 [Server 3 - Coordinator] Synchronize to value
2391288529, sending to 2
assignment 2: 2023/10/19 00:46:25 [Server 2] Synchronization request from 3,
current local time 2373144684, set localtime to 2391288529
assignment 2: 2023/10/19 00:46:25 [Server 3 - Coordinator] Synchronize to value
2391288529, sending to 4
assignment 2: 2023/10/19 00:46:25 [Server 4] Election is ongoing, data
synchronization request is not accepted
assignment 2: 2023/10/19 00:46:27 [Server 1] Ask Heartbeat from 3
assignment 2: 2023/10/19 00:46:27 [Server 3 - Coordinator] Sending Heartbeat to 1
assignment 2: 2023/10/19 00:46:27 [Server 2] Ask Heartbeat from 3
assignment 2: 2023/10/19 00:46:27 [Server 3 - Coordinator] Sending Heartbeat to 2
assignment 2: 2023/10/19 00:46:27 [Server 0] Ask Heartbeat from 3
assignment 2: 2023/10/19 00:46:27 [Server 3 - Coordinator] Sending Heartbeat to 0
assignment 2: 2023/10/19 00:46:33 [Server 4 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 00:46:33 [Server 3] Announcement received from 4
assignment 2: 2023/10/19 00:46:33 [Server 3] Set coordinator to 4
assignment 2: 2023/10/19 00:46:33 [Server 2] Announcement received from 4
assignment 2: 2023/10/19 00:46:33 [Server 2] Set coordinator to 4
assignment 2: 2023/10/19 00:46:33 [Server 1] Announcement received from 4
assignment 2: 2023/10/19 00:46:33 [Server 1] Set coordinator to 4
assignment 2: 2023/10/19 00:46:33 [Server 0] Announcement received from 4
assignment 2: 2023/10/19 00:46:33 [Server 0] Set coordinator to 4
assignment 2: 2023/10/19 00:46:35 [Server 4 - Coordinator] Synchronize to value
627354768, sending to 0
```

Simulation 3.A. Simulate an elected coordinator fails while announcing

This case could be well-handled by the embedded heartbeat mechanism in this implementation. To simulate this, activate a server, in this scenario specifically, the server with the highest ID, by invoking the `PleaseFailWhileAnnounce()` method.

```
// 3.a Simulate newly elected coordinator fails while announcing that it has won
election to all nodes
for index, server := range servers {
    if index < len(servers)-1 {
        server.Activate()
    } else {
        server.PleaseFailWhileAnnounce()
    }
}
```

The `PleaseFailWhileAnnounce()` method serves the purpose of setting the server's status to `DOWN` after sending its initial announcement to another server. As a result, only one server receives the announcement, while the remaining servers remain uninformed. In this case, the informed server will eventually detect the failure of the coordinator, and initiates a new round of election.

Here shows the log. Server 4 is down after announcing to Server 0. Note `[Server 0] Ask Heartbeat from 4` and `[Server 0] Fail to Ask Heartbeat from 4, restart election`.

```
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Opps.. I am DOWN
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Synchronize to value
1960012632, sending to 0
assignment 2: 2023/10/19 00:56:51 [Server 0] Announcement received from 4
assignment 2: 2023/10/19 00:56:51 [Server 0] Set coordinator to 4
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Synchronize to value
1960012632, sending to 1
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Synchronize to value
1960012632, sending to 2
assignment 2: 2023/10/19 00:56:51 [Server 4 - Coordinator] Synchronize to value
1960012632, sending to 3
assignment 2: 2023/10/19 00:56:51 [Server 3] Election is ongoing, data
synchronization request is not accepted
assignment 2: 2023/10/19 00:56:51 [Server 0] Synchronization request from 4,
current local time 1950445904, set localtime to 1960012632
assignment 2: 2023/10/19 00:56:51 [Server 1] Election is ongoing, data
synchronization request is not accepted
assignment 2: 2023/10/19 00:56:51 [Server 2] Election is ongoing, data
synchronization request is not accepted
assignment 2: 2023/10/19 00:56:53 [Server 2] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:56:53 [Server 3] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:56:53 [Server 1] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:56:53 [Server 0] Ask Heartbeat from 4
assignment 2: 2023/10/19 00:56:57 [Server 0] Fail to Ask Heartbeat from 4,
restart election
```

```
assignment 2: 2023/10/19 00:56:57 [Server 0] Sending election message to 1
assignment 2: 2023/10/19 00:56:57 [Server 0] Sending election message to 2
assignment 2: 2023/10/19 00:56:57 [Server 0] Sending election message to 3
assignment 2: 2023/10/19 00:56:57 [Server 0] Sending election message to 4
assignment 2: 2023/10/19 00:56:57 [Server 1] Sending election message to 2
assignment 2: 2023/10/19 00:56:57 [Server 1] Sending election message to 3
assignment 2: 2023/10/19 00:56:57 [Server 1] Sending election message to 4
assignment 2: 2023/10/19 00:56:57 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 00:56:57 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 00:56:57 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 00:56:57 [Server 2] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 3] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 0] Reply from 3, stop electing 0
assignment 2: 2023/10/19 00:56:57 [Server 2] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 0] Reply from 2, stop electing 0
assignment 2: 2023/10/19 00:56:57 [Server 3] Election request from 2, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 00:56:57 [Server 3] Election request from 1, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 1] Reply from 2, stop electing 1
assignment 2: 2023/10/19 00:56:57 [Server 1] Reply from 3, stop electing 1
assignment 2: 2023/10/19 00:56:57 [Server 1] Election request from 0, replying a
no message
assignment 2: 2023/10/19 00:56:57 [Server 0] Reply from 1, stop electing 0
assignment 2: 2023/10/19 00:57:03 [Server 3] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:57:03 [Server 2] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:57:03 [Server 0] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:57:03 [Server 1] Election is ongoing, heartbeat
checking is ceased
assignment 2: 2023/10/19 00:57:05 [Server 3 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 00:57:05 [Server 0] Announcement received from 3
assignment 2: 2023/10/19 00:57:05 [Server 0] Set coordinator to 3
assignment 2: 2023/10/19 00:57:05 [Server 1] Announcement received from 3
assignment 2: 2023/10/19 00:57:05 [Server 1] Set coordinator to 3
assignment 2: 2023/10/19 00:57:05 [Server 2] Announcement received from 3
assignment 2: 2023/10/19 00:57:05 [Server 2] Set coordinator to 3
assignment 2: 2023/10/19 00:57:05 [Server 3 - Coordinator] Synchronize to value
4408242483, sending to 1
assignment 2: 2023/10/19 00:57:05 [Server 1] Synchronization request from 3,
current local time 4418104966, set localtime to 4408242483
assignment 2: 2023/10/19 00:57:05 [Server 3 - Coordinator] Synchronize to value
4408242483, sending to 0
assignment 2: 2023/10/19 00:57:05 [Server 0] Synchronization request from 3,
current local time 4424529974, set localtime to 4408242483
assignment 2: 2023/10/19 00:57:05 [Server 3 - Coordinator] Synchronize to value
4408242483, sending to 2
assignment 2: 2023/10/19 00:57:05 [Server 2] Synchronization request from 3,
current local time 4406341958, set localtime to 4408242483
```

```
assignment 2: 2023/10/19 00:57:05 [Server 3 - Coordinator] Synchronize to value
4408242483, sending to 4
assignment 2: 2023/10/19 00:57:13 [Server 2] Ask Heartbeat from 3
assignment 2: 2023/10/19 00:57:13 [Server 3 - Coordinator] Sending Heartbeat to 2
```

Simulation 3.B. A node fails while election, the failed node is not the newly elected coordinator

This case is well-handled by the embedded message sending timeout mechanism in this implementation. To simulate this, activate a server, in this example, the server with the ID 1, by invoking the `PleaseFailWhileElection()` method.

```
for index, server := range servers {
    if index != 1 {
        server.Activate()
    } else {
        server.PleaseFailWhileElection()
    }
}
```

Within the server's implementation, this failure is determined based on the specified `replyTimeout`. If the time it takes to send a message exceeds the `replyTimeout` duration, the server considers the communication as "Failed". Notably, from the perspective of the Bully algorithm, this election process is not affected either.

Here shows the output log. It is worth noting that the message `[Server 1] Ops... I am DOWN` indicates a server failure. Remarkably, this failure does not disrupt the ongoing election process or the subsequent synchronization and heartbeat operations. The system continues to operate seamlessly.

```
assignment 2: 2023/10/19 01:10:06 [Server Activate] Server 0 is activated
assignment 2: 2023/10/19 01:10:06 [Server Activate] Server 1 is activated
assignment 2: 2023/10/19 01:10:06 [Server Activate] Server 4 is activated
assignment 2: 2023/10/19 01:10:06 [Server Activate] Server 2 is activated
assignment 2: 2023/10/19 01:10:06 [Server Activate] Server 3 is activated
assignment 2: 2023/10/19 01:10:16 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 01:10:16 [Server 0] Sending election message to 1
assignment 2: 2023/10/19 01:10:16 [Server 0] Sending election message to 2
assignment 2: 2023/10/19 01:10:16 [Server 1] Election request from 0, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 0] Reply from 1, stop electing 0
assignment 2: 2023/10/19 01:10:16 [Server 2] Election request from 0, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 0] Reply from 2, stop electing 0
assignment 2: 2023/10/19 01:10:16 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 01:10:16 [Server 2] Sending election message to 4
assignment 2: 2023/10/19 01:10:16 [Server 4] Election request from 3, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 3] Election request from 2, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 01:10:16 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 01:10:16 [Server 1] Sending election message to 2
assignment 2: 2023/10/19 01:10:16 [Server 1] Ops... I am DOWN
assignment 2: 2023/10/19 01:10:16 [Server 4] Election request from 2, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 2] Reply from 4, stop electing 2
assignment 2: 2023/10/19 01:10:16 [Server 0] Sending election message to 3
assignment 2: 2023/10/19 01:10:16 [Server 2] Election request from 1, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 2] Sending election message to 3
assignment 2: 2023/10/19 01:10:16 [Server 3] Election request from 2, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 2] Reply from 3, stop electing 2
assignment 2: 2023/10/19 01:10:16 [Server 3] Sending election message to 4
assignment 2: 2023/10/19 01:10:16 [Server 4] Election request from 3, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 3] Reply from 4, stop electing 3
assignment 2: 2023/10/19 01:10:16 [Server 3] Election request from 0, replying a
no message
assignment 2: 2023/10/19 01:10:16 [Server 0] Reply from 3, stop electing 0
assignment 2: 2023/10/19 01:10:24 [Server 4 - Coordinator] Announcing Coordinator
assignment 2: 2023/10/19 01:10:24 [Server 3] Announcement received from 4
assignment 2: 2023/10/19 01:10:24 [Server 3] Set coordinator to 4
assignment 2: 2023/10/19 01:10:24 [Server 0] Announcement received from 4
assignment 2: 2023/10/19 01:10:24 [Server 0] Set coordinator to 4
assignment 2: 2023/10/19 01:10:24 [Server 2] Announcement received from 4
assignment 2: 2023/10/19 01:10:24 [Server 2] Set coordinator to 4
assignment 2: 2023/10/19 01:10:24 [Server 4 - Coordinator] Synchronize to value
2197493958, sending to 1
assignment 2: 2023/10/19 01:10:24 [Server 4 - Coordinator] Synchronize to value
2197493958, sending to 0
assignment 2: 2023/10/19 01:10:24 [Server 0] Synchronization request from 4,
current local time 2218308531, set localtime to 2197493958
assignment 2: 2023/10/19 01:10:24 [Server 4 - Coordinator] Synchronize to value
2197493958, sending to 2
assignment 2: 2023/10/19 01:10:24 [Server 2] Synchronization request from 4,
current local time 2207598084, set localtime to 2197493958
assignment 2: 2023/10/19 01:10:24 [Server 4 - Coordinator] Synchronize to value
2197493958, sending to 3
assignment 2: 2023/10/19 01:10:24 [Server 3] Synchronization request from 4,
current local time 2218337233, set localtime to 2197493958
assignment 2: 2023/10/19 01:10:26 [Server 0] Ask Heartbeat from 4
assignment 2: 2023/10/19 01:10:26 [Server 4 - Coordinator] Sending Heartbeat to 0
```

Simulation 4. Multiple GO routines start the election process simultaneously

In this implementation, the nature of the system inherently involves the simultaneous initiation of multiple election processes. That is to say, in any cases, all GO routines start the election process simultaneously. To observe this behavior, you can refer to **Simulation 1**, or any other simulations.

Simulation 5. Node silently leaves the network

This scenario, where a node silently leaves the network, closely mirrors **Simulation 3: A Node Fails**. In both cases, the method to simulate this is identical: setting the server's status to DOWN. Consequently, the entire process, the handling mechanism, and the resulting outcomes are consistent with those in **Simulation 3**. I believe it is reasonable to conclude that **Simulation 3** effectively demonstrates this scenario.

Conclusion

In assignment 1, I simulated the behavior of distributed systems and explored two clock synchronization mechanisms: Lamport's Logical Clock and Vector Clock. The implementation demonstrated the robustness and reliability of these synchronization methods, highlighting their vector clock's to maintain global order and detect potential causality violations.

Assignment 2 is an implementation of replica synchronization and coordinator election using bully algorithm. By simulating various situations, such as best-case and worst-case elections and server failures, the robustness of the system and bully algorithm is proved. Moreover, it's essential to highlight that the implemented model has the capacity to simulate an even broader range of cases. The system's adaptability and robustness can withstand rigorous testing scenarios, including situations where messages arrive out of order.