Lab 2: gRPC

Course Title: Distributed and Cloud Computing

Course Number: COE892

Semester/Year: Winter 2024

Instructor: Dr. Muhammad Jaseemuddin

TA: Abdul Bhutta (Section 4)

Due date: March 10, 2024 @ 23:59

Authored by: Ibrahima Bah

Table of Contents

Introduction

The objective of this lab is to modify the work of the previous lab to implement gRPC and proto buffers to create a client-server architecture for the rover program. We will work to create a server (ground control) that retrieves the relevant data at the request of the clients (rovers). Most of the basic implementations were already created for the first lab so this lab will focus on the creation of the proto files and the implemented functions.

Part 1: Creating the Proto Buffer file

Basic Implementation

We create the services of the protobuf based on the following criterias:

1. Get map (e.g., “The 2D land array”)
2. Get a stream of commands (e.g., “RMLMMMMMDLMMRMD”)
3. Get a mine serial number
4. Let the server know if they have executed all commands successfully, or not (a mine might explode with a wrong series of commands)
5. Share a mine PIN with the server

```
service rovers {
    rpc getMap (mapRequest) returns (mapReply) {}
    rpc getCommandStream(commandStreamRequest) returns (commandStreamReply) {}
    rpc getMineSerial(serialNumRequest) returns (serialNumReply) {}
    rpc completedCommands(completedRequest) returns (completedReply) {}
    rpc sendPin(pinRequest) returns (pinReply) {}
}
```

Now we simply create the messages for both the requests and response (replies) of each service. We ensure to include the data to be passed through each service (rover numbers, serials, pins, commands, etc).

Get Map

A rover will send a request to the server containing its number as the ID. The server will retrieve the map data and then send a reply back to the rover with the map and the number of rows and columns.

```
message mapRequest {
    string id = 1;
}
message mapReply {
    repeated string map = 1;
    int32 rows = 2;
    int32 cols = 3;
}
```

Get Command Stream

A rover will send a request to the server containing its number as the ID. The server will retrieve the commands from the API and return the commands back to the rover.

```
message commandStreamRequest {
    string id = 1;
}
message commandStreamReply {
    string cmds = 1;
}
```

Get Mine Serial Number

A rover will send a request to the server containing its number as the ID along with its current position (only sent when on a mine). The server will retrieve the serial number of the mine the rover is currently on and send it over to the rover.

```
message serialNumRequest {
    string id = 1;
    int32 i = 2;
    int32 j = 3;
}
message serialNumReply {
    string serialNum = 1;
}
```

Rover Completion Alert

A rover will send an alert (request) to the server containing its number as the ID and either 0, 1, or 2 as the code (0 = completed map, 1 = did not dig bomb, 2 = failed to disarm bomb).

```
message completedRequest {
    string id = 1;
    int32 code = 2;
}
message completedReply {
    string ack = 1;
}
```

Rover Valid Pin Alert

A rover will send an alert (request) to the server containing its number as the ID along with the serial number of the bomb it is on and the pin found to deactivate it..

```
message pinRequest {
    string id = 1;
    string serialNum = 2;
    string pin = 3;
}
message pinReply {
    string ack = 1;
}
```

With the proto buffer file completed we can compile it to create three separate files that we will use in the next part as templates to begin the implementation of each service as a function.

Running the following command will compile the protobuf:

```
python -m grpc_tools.protoc -I=proto --python_out=.
--grpc_python_out=. --pyi_out=.  proto/rovers.proto
```

-I : Input folder where proto is found

--python_out : Output location of pb2.py file

--grpc_python_out : Output location of pb2_grcp.py file

--pyi_out : Output location of .pyi file

proto/rovers.proto : Location of proto file

Part 2: Creating the Server

As stated in the introduction, this lab builds upon the previous one so to implement each function we can take lines from separate python programs and insert them here to ensure the overall functionality of the rover program stays the same.

We first create a function to 'serve' the server. This function will initialize the grpc methods and set up the server information to be used and accessed by the clients.

```
def serve() -> int:
    logging.info("Ground Control\t: Starting server......")
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    rovers_pb2_grpc.add_roversServicer_to_server(groundController(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    logging.info(f"Ground Control\t: Server running")
    server.wait_for_termination()
    return 0
```

Now we can begin the implementation of each function as shown below:

getMap()

```
# Retrieves map data from map.txt file
def getMap(self, request, context):

    # Retrieve map info from map.txt
    with open("map.txt", "r") as file:
        rows, cols = map(int, file.readline().split())
        matrix = np.empty((rows, cols), dtype='U21')
        for i in range(rows):
            row = list(map(int, file.readline().split()))
            matrix[i] = row

    # Convert array to string stream (list)
    matrix_list = matrix.flatten().tolist()

    return rovers_pb2.mapReply(map=matrix_list, rows=rows, cols=cols)
```

getCommandStream()

```
# Retrieve commands string from API
def getCommandStream(self, request, context):

    # Get data from URL
    res = requests.get(f'https://coe892.reev.dev/lab1/rover/{request.id}')

    # Parse json data
    cmds = json.loads(f'{res.text}')["data"]["moves"]

    return rovers_pb2.commandStreamReply(cmds=cmds)
```

getMineSerial()

```
# Retrieves mines serial number at given coordinates (mines.txt)
def getMineSerial(self, request, context):
    # Initialize mines serial numbers
    with open("mines.txt", "r") as f:
        json_data : dict = json.load(f)

    for serial, location in json_data.items():
        if (location == [request.i, request.j]):
            return rovers_pb2.serialNumReply(serialNum=serial)
```

completedCommands()

```
# Rover completion alert
def completedCommands(self, request, context):
    if (request.code == 0):
        logging.info(f'Ground Control\t: rover {request.id} has
                     successfully executed all commands')
    elif (request.code == 1):
        logging.info(f'Ground Control\t: rover {request.id} failed to
                     defuse a mine')
    elif (request.code == 2):
        logging.info(f'Ground Control\t: rover {request.id} failed to
                     defuse a mine')
    return rovers_pb2.completedReply(ack="ACK")
```

sendPin()

```
# Rover valid pin found alert
    def sendPin(self, request, context):
        logging.info(f'Ground Control\t: rover {request.id} has found
                       valid pin ({request.pin}) for serial num ->
                       {request.serialNum}')
        return rovers_pb2.pinReply(ack="ACK")
```

Part 3: Creating the Client

Similar to part 2, we will be taking lines and sections of previous programs to implement the client program. Because of this, we will instead focus on the sections that require calling the server to retrieve or send data. But first we add a simple line to prompt users to enter the ID.

```
id = input("Enter rover number: ")
```

Get Map

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.getMap(rovers_pb2.mapRequest(id=id))
matrix = np.array(response.map).reshape(response.rows, response.cols)
rows = response.rows, cols = response.cols
```

Get Command Stream

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.getCommandStream(rovers_pb2.commandStreamRequest(
                                        id=id))
cmd = response.cmds
```

Get Mine Serial Number

```
with grpc.insecure_channel('localhost:50051') as channel:
   stub = rovers_pb2_grpc.roversStub(channel)
   response = stub.getMineSerial(rovers_pb2.serialNumRequest(
                                    id=id, i=cur_i, j=cur_j))
```

Completion Alert

Code (0)

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.completedCommands(rovers_pb2.completedRequest(
                                        id=id, code=0))
```

Code (1)

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.completedCommands(rovers_pb2.completedRequest(
                                        id=id, code=1))
```

Code (2)

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.completedCommands(rovers_pb2.completedRequest(
                                        id=id, code=2))
```

Valid Pin Alert

```
with grpc.insecure_channel('localhost:50051') as channel:
    stub = rovers_pb2_grpc.roversStub(channel)
    response = stub.completedCommands(rovers_pb2.completedRequest(
                                        id=id, code=1))
```

Results

Server Side

```
PS E:\Users\Ibrahima (Ibah)\Documents\TMU\year 5\sem 8\COE892\Labs\Lab2> python ./groundcontrol_server.py
18:09:15: Ground Control        : Starting server......
18:09:15: Ground Control        : Server running
18:09:19: Ground Control <- Rover 2     : Map Request...
18:09:19: Ground Control -> Rover 2     : Sending map=['0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0']
18:09:19: Ground Control <- Rover 2     : Command Request...
18:09:19: Ground Control -> Rover 2     : Sending CMD=MRDRMMLMLRDRLLMRLLMMRDLMLDMLDLDLDLRLMRLMMRMRLMLMLMDMRLRMMMMMMM
18:09:19: Ground Control <- Rover 2     : Serial Request - i=0, j=1
18:09:19: Ground Control -> Rover 2     : Sending serial=LC8ZOSFZ
18:09:20: Ground Control        : rover 2 has found valid pin (P08QI7) for serial num -> LC8ZOSFZ
18:09:20: Ground Control        : rover 2 has succesfully executed all commands
```

Client Side

```
PS E:\Users\Ibrahima (Ibah)\Documents\TMU\year 5\sem 8\COE892\Labs\Lab2> python ./rover_client.py
Enter rover number: 2
18:09:19: Rover 2       : Setting up rover...
18:09:19: Rover 2       : Retriving map...
18:09:19: Rover 2       : rows = 4, cols = 3
18:09:19: Rover 2       : map_list = ['0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0']
18:09:19: Rover 2       : Retriving commands...
```

```
18:09:19: Rover 2       : Rover is currently on a mine!
18:09:19: Rover 2       : Command (D): Dig
18:09:19: ValidPin      : Mine Serial Number = LC8ZOSFZ
18:09:20: ValidPin      : Valid Pin = P08QI7
18:09:20: Rover 2       : Successfully disarmed mine
18:09:20: Rover 2       : Command (M): Move forward
```

```
18:09:20: Rover 2       : Command (M): Move forward
18:09:20: Rover 2       : All Commands executed : closing
18:09:20: Rover 2       : closing rover...
PS E:\Users\Ibrahima (Ibah)\Documents\TMU\year 5\sem 8\COE892\Labs\Lab2>
```

Conclusion

Using gRPC we were able to create a client-server architecture for the rover pathing program we initially implemented in the first lab. The results of the paths are all the same as expected but now the program runs a single rover on the client side with the given rover number provided through CLI for the user to input.

Appendix A: Python Code

rovers.proto

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "io.grpc.examples.rovers";
option java_outer_classname = "RoversProto";
option objc_class_prefix = "RVRP";
service rovers {
	rpc getMap (mapRequest) returns (mapReply) {}
	rpc getCommandStream(commandStreamRequest) returns (commandStreamReply) {}
	rpc getMineSerial(serialNumRequest) returns (serialNumReply) {}
	rpc completedCommands(completedRequest) returns (completedReply) {}
	rpc sendPin(pinRequest) returns (pinReply) {}
}

// Get Map Messages
message mapRequest {
	string id = 1;
}
message mapReply {
	repeated string map = 1;
	int32 rows = 2;
	int32 cols = 3;
}

// Get Command Steam Messages
message commandStreamRequest {
	string id = 1;
}
```

```
message commandStreamReply {
        string cmds = 1;
}

// Get Mine Serial Messages
message serialNumRequest {
        string id = 1;
        int32 i = 2;
        int32 j = 3;
}
message serialNumReply {
        string serialNum = 1;
}

//
message completedRequest {
        string id = 1;
        int32 code = 2;
}
message completedReply {
        string ack = 1;
}
//
message pinRequest {
        string id = 1;
        string serialNum = 2;
        string pin = 3;
}
message pinReply {
        string ack = 1;
}
```

groundcontrol_server.py

```
import logging, sys, requests, json
import numpy as np
import grpc, rovers_pb2, rovers_pb2_grpc
from concurrent import futures

class groundController(rovers_pb2_grpc.roversServicer):
	# Retrieves  map data from map.txt file
	def getMap(self, request, context):
	logging.info(f'Ground Control <- Rover {request.id}\t: Map Request...')

	# Retrieve map info from map.txt
	with open("map.txt", "r") as file:
	rows, cols = map(int, file.readline().split())
	matrix = np.empty((rows, cols), dtype='U21')
	for i in range(rows):
		row = list(map(int, file.readline().split()))
		matrix[i] = row

	# Convert array to string stream (list)
	matrix_list = matrix.flatten().tolist()

	logging.info(f'Ground Control -> Rover {request.id}\t: Sending map={matrix_list}')
	return rovers_pb2.mapReply(map=matrix_list, rows=rows, cols=cols)

	# Retrieve commands string from API
	def getCommandStream(self, request, context):
	logging.info(f'Ground Control <- Rover {request.id}\t: Command Request...')

	# Get data from URL
```

```
        res = requests.get(f'https://coe892.reev.dev/lab1/rover/{request.id}')

        # Parse json data
        cmds = json.loads(f'{res.text}')["data"]["moves"]

        logging.info(f'Ground Control -> Rover {request.id}\t: Sending CMD={cmds}')
        return rovers_pb2.commandStreamReply(cmds=cmds)

        # Retrieves mines serial number at given coordinates (mines.txt)
        def getMineSerial(self, request, context):
        logging.info(f'Ground Control <- Rover {request.id}\t: Serial Request - i={request.i}, j={request.j}')

        # Initialize  mines serial numbers
        with open("mines.txt", "r") as f:
        json_data : dict = json.load(f)

        for serial, location in json_data.items():
        if (location == [request.i, request.j]):
                logging.info(f'Ground Control -> Rover {request.id}\t: Sending serial={serial}')
                return rovers_pb2.serialNumReply(serialNum=serial)

        # Rover completion alert
        def completedCommands(self, request, context):
        if (request.code == 0):
        logging.info(f'Ground Control\t: rover {request.id} has succesfully executed all commands')
        elif (request.code == 1):
        logging.info(f'Ground Control\t: rover {request.id} failed to dig a mine')
        elif (request.code == 2):
        logging.info(f'Ground Control\t: rover {request.id} failed to defuse a mine')
```

```
        return rovers_pb2.completedReply(ack="ACK")

        # Rover valid pin found alert
        def sendPin(self, request, context):
        logging.info(f'Ground Control\t: rover {request.id} has found valid pin ({request.pin}) for serial num -> {request.serialNum}')
        return rovers_pb2.pinReply(ack="ACK")


def serve() -> int:
        logging.info("Ground Control\t: Starting server.....")
        server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
        rovers_pb2_grpc.add_roversServicer_to_server(groundController(), server)
        server.add_insecure_port('[::]:50051')
        server.start()
        logging.info(f"Ground Control\t: Server running")
        server.wait_for_termination()
        return 0


if __name__ == '__main__':
        FORMAT = "%(asctime)s: %(message)s"
        logging.basicConfig(format=FORMAT, level=logging.INFO, datefmt="%H:%M:%S")
        logging.disable(logging.DEBUG)
        sys.exit(serve())
```

rover_client.py

```
import hashlib, logging, sys, requests, os, json, random, string
import numpy as np
import grpc, rovers_pb2, rovers_pb2_grpc
from concurrent import futures

global path
path = "./paths"

# Compute the hashed key using sha256
def key(pin: str, serial: str) -> str:

    # Concatenate strings (Temporary Key)
    tmpkey = pin + serial

    # Hash Temporary Key
    h = hashlib.sha256()
    h.update(tmpkey.encode())

    # Hashed value
    hashkey = h.hexdigest()

    return hashkey

# Brute Forces to find a valid pin for the given serial number
def validpin(serialnum, n=500000) -> bool:

    logging.info(f'ValidPin\t: Mine Serial Number = {serialnum}')

    # Brute force arbitrary pins
```

```
        for i in range(n):
        # Select a random 6 digit number
        pin = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(6))
        logging.debug(f'Temporary Key\t= {pin} {serialnum}')

        # Hash Temporary key
        val = key(pin, serialnum)
        logging.debug(f'Hashed Key[0:5]\t= {val[0:5]}')

        # Check if hash has at least five leading zeros
        if (val[0:4] == "0000"):
        logging.info(f'ValidPin\t: Valid Pin = {pin}')

        # Alert ground control of valid pin
        with grpc.insecure_channel('localhost:50051') as channel:
                stub = rovers_pb2_grpc.roversStub(channel)
                response = stub.sendPin(rovers_pb2.pinRequest(id=id, serialNum=serialnum,
pin=pin))

        return True

        logging.info(f'ValidPin\t: Valid Pin not found')
        return False

# Rover Pathing Program
def run(id : str, cmd: str, rows: int, cols: int, matrix):
        logging.info(f'Rover {id}\t: Running rover...')

        # Rover Variables
        onMine  : bool  = False
        facing  : str   = "DOWN"
```

```
        cur_i   : int   = 0
        cur_j   : int   = 0

        # Initialize rover position
        matrix[0][0] = id

        # Other Variables
        index   : int   = 0
        running : bool  = True

        # Initilize mines serial numbers
        with open("mines.txt", "r") as f:
        json_data : dict = json.load(f)

        while (running):

        # print(f"i = {cur_i}, j = {cur_j}, facing = {facing}")
        # for row in range(rows):
        #         print(matrix[row])
        # print('-----------------------')

        if (onMine == True and cmd[index] != "D"):
        logging.info(f'Rover {id}\t: Command ({cmd[index]}) -- Rover did not dig mine: closing')
        matrix[cur_i][cur_j] = "x"

        # Alert Ground control, failed to defuse
        with grpc.insecure_channel('localhost:50051') as channel:
                stub = rovers_pb2_grpc.roversStub(channel)
                response = stub.completedCommands(rovers_pb2.completedRequest(id=id, code=1))
```

```
running = False
else:
try:
        match cmd[index]:
        case 'R':
        # Turn rover to the right
        logging.info(f'Rover {id}\t: Command ({cmd[index]}) = Turn right')
        if (facing == "DOWN"):
                facing = "LEFT"
        elif (facing == "LEFT"):
                facing = "UP"
        elif (facing == "UP"):
                facing = "RIGHT"
        elif (facing == "RIGHT"):
                facing = "DOWN"
        case 'L':
        # Turn rover to the left
        logging.info(f'Rover {id}\t: Command ({cmd[index]}) = Turn left')
        if (facing == "DOWN"):
                facing = "RIGHT"
        elif (facing == "RIGHT"):
                facing = "UP"
        elif (facing == "UP"):
                facing = "LEFT"
        elif (facing == "LEFT"):
                facing = "DOWN"
        case 'M':
        # Move rover forward 1 space
        logging.info(f'Rover {id}\t: Command ({cmd[index]}): Move forward')
        matrix[cur_i][cur_j] = "*"  # Set trail on map
```

```
# Change Rover Position
if ((facing == "DOWN") and (0 <= cur_i + 1 < rows)):
        cur_i += 1
elif ((facing == "RIGHT") and (0 <= cur_j + 1 < cols)):
        cur_j += 1
elif ((facing == "UP") and (0 <= cur_i - 1 < rows)):
        cur_i -= 1
elif ((facing == "LEFT")  and (0 <= cur_j - 1 < cols)):
        cur_j -= 1

# Check if rover is on a mine (1)
if (matrix[cur_i][cur_j] == "1"):
        logging.info(f'Rover {id}\t: Rover is currently on a mine!')
        onMine = True
matrix[cur_i][cur_j] = id   # Set rover on map
case 'D':
logging.info(f'Rover {id}\t: Command ({cmd[index]}): Dig')
if (onMine == True):
        # Request mine serial number and disarm
        with grpc.insecure_channel('localhost:50051') as channel:
        stub = rovers_pb2_grpc.roversStub(channel)
        response = stub.getMineSerial(rovers_pb2.serialNumRequest(id=id,
i=cur_i, j=cur_j))
        if (validpin(response.serialNum)):
        logging.info(f'Rover {id}\t: Successfully disarmed mine')
        onMine = False
        else:
        logging.info(f'Rover {id}\t: Rover did not disarm mine: closing')
        matrix[cur_i][cur_j] = "X"
```

```
                    # Alert Ground control, failed to defuse
                    with grpc.insecure_channel('localhost:50051') as channel:
                    stub = rovers_pb2_grpc.roversStub(channel)
                    response =
stub.completedCommands(rovers_pb2.completedRequest(id=id, code=2))

                    running = False
                case _:
                pass
                index += 1
        except IndexError:
                logging.info(f'Rover {id}\t: All Commands executed : closing')

                # Alert Ground control, completed commands
                with grpc.insecure_channel('localhost:50051') as channel:
                stub = rovers_pb2_grpc.roversStub(channel)
                response = stub.completedCommands(rovers_pb2.completedRequest(id=id,
code=0))


                running = False

        # Save Rover Path
        np.savetxt(f'{path}/path_{id}.txt', matrix, fmt='%s')

def main() -> int:
        global id

        # CLI Rover Number
        id = input("Enter rover number: ")
```

```
logging.info(f'Rover {id}\t: Setting up rover...')

# Retrive Map
logging.info(f'Rover {id}\t: Retriving map...')

with grpc.insecure_channel('localhost:50051') as channel:
stub = rovers_pb2_grpc.roversStub(channel)
response = stub.getMap(rovers_pb2.mapRequest(id=id))
matrix = np.array(response.map, dtype='U21').reshape(response.rows, response.cols)
rows = response.rows
cols = response.cols

logging.info(f'Rover {id}\t: rows = {response.rows}, cols = {response.cols}')
logging.info(f'Rover {id}\t: map_list = {response.map}')



# Get Commands
logging.info(f'Rover {id}\t: Retriving commands...')

with grpc.insecure_channel('localhost:50051') as channel:
stub = rovers_pb2_grpc.roversStub(channel)
response = stub.getCommandStream(rovers_pb2.commandStreamRequest(id=id))
cmd = response.cmds

logging.debug(f'Rover {id}\t: cmds = {response.cmds}')

# Create paths subfolder if not exists
os.makedirs(path, exist_ok=True)

# Run Program
print("")
```

```
        run(id=id, cmd=cmd, rows=rows, cols=cols, matrix=matrix)

        logging.info(f'Rover {id}\t: closing rover...')
        return 0

if __name__ == '__main__':
        FORMAT = "%(asctime)s: %(message)s"
        logging.basicConfig(format=FORMAT, level=logging.DEBUG, datefmt="%H:%M:%S")
        logging.disable(logging.DEBUG)
        sys.exit(main())
```

Appendix B: Path Results

*: Rover's trail 0: Blank space x: Last position (mine blew up) 1: mine

Path_1.txt	Path_2.txt	Path_3.txt	Path_4.txt	Path_5.txt
* 1 0 * 0 0 x 0 0 0 0 0	2 * * * * * 1 0 0 0 0 0	* 1 0 * 0 0 x 0 0 0 0 0	* x 0 * 0 0 1 0 0 0 0 0	* x 0 * 0 0 1 0 0 0 0 0

Path_6.txt	Path_7.txt	Path_8.txt	Path_9.txt	Path_10.txt
* 1 0 * 0 0 x 0 0 0 0 0	* 1 0 * 0 0 x 0 0 0 0 0	* 1 0 * * * x 0 0 0 0 0	* x 0 0 0 0 1 0 0 0 0 0	* 1 0 * 0 0 x 0 0 0 0 0