COE 892 - Distributed Cloud Computing

Winter 2025

Dr. Khalid A. Hafeez

Lab 2 - gRPC

Submission Date: February 22, 2025

Eric Muzzo

501019745

Introduction

This lab builds on lab 1 by introducing the concept of gRPC, a modern open source Remote Procedure Call framework created by google. Lab 1 is redesigned to now include a ground control station, which facilitates communication with each of the rover nodes. The ground control is a gRPC server that has methods for carrying out the same set of tasks as done in Lab 1. Refer to the Lab 1 report for a breakdown on the main objectives of each rover and how they work. To avoid redundancy, only the changes to the first iteration of the application will be described.

Program Design & Overview

Proto Buffer

First, the .proto file was created which defines the services, methods and messages that the client and server will use to communicate with each other. A single service, "GroundControl" is defined with the 5 RPC methods specified in the lab manual. These are summarized in the list below. The ReportStatus and ShareMinPin methods do not have a return message since they are only status indicators from the Rovers and do not need a response. Thus, these methods use a generic empty message which is imported. I defined both request and response message structures for methods where a request and response was necessary.

Remote Procedure Calls:

- **GetMap:** a rover requests the map data
    - **Messages Used:**
        - *MapRequest*: blank
        - *MapRow*: a row in the map (used to implement 2D array data structure in MapResponse.
        - *MapResponse*: the 2D array of cells and the dimensions of the map
- **GetCommands:** a rover requests its commands
    - **Messages Used:**
        - *CommandRequest*: requires the rover_id
        - *CommandResponse*: a string of commands
- **GetMineSerial:** a rover requests the serial number of a mine at a given location
    - **Messages Used:**
        - *SerialNumRequest:* the x and y coordinates of the mine
        - *SerialNumResponse:* the serial number as a string
- **ReportStatus:** a rover reports its status to the server
    - **Messages Used:**

        - *ExecutionStatus*: provide the rover_id, a boolean flag indicating success/failure, a string message
- **ShareMinePin:** a rover shares the decoded pin of a min with the server
    - **Messages Used:**
        - MinPin: the rover_id and the pin as a string

I then used the grpcio-tools library in python to compile and generate the client and server interfaces from the proto buffer definition using the following command:

```
python -m grpc_tools.protoc --proto_path=./rpc --python_out=./rpc --grpc_python_out=./rpc
rpc/ground_control.proto
```

Note that I manually modified one of the output files, “ground_control_pb2_grpc”, in order to resolve the relative import issue due to my project directory structure.

Server

The GroundControlService class is implemented by inheriting the generated GroundControlServicer and implementing all of the remote procedure calls defined in the proto buffer file. Most of the functionality seen here is very similar to my implementation in Lab 1. I first initialize the ServerMap object from the map.txt file into a custom data structure that I defined. The ServerMap class inherits the Map data structure that I defined in the last lab (see Lab 1 Report for reference). This is done for two reasons; the amount of redundant code is reduced and I can instantiate a Map object from the server using file paths and from the client using a 2D array. The server then stores this in working memory so that it can quickly access properties when a client requests them. This class is added to the gRPC servicer within the serve() function and begins to listen for client requests.

Client

The client program begins by prompting the user to enter a rover number to execute with. Once this process begins, I first perform a few setup and initialization functions before actually running the rover. A remote procedure call is made to the server for the map using the GetMap service. The response is processed into the Map parent class. Expanding on the reason for two Map classes as described in the Server section above, having a separate constructor for the client to use allows me to instantiate the mine cells without knowing their serial numbers. The client then makes another call to the server to get the command sequence for the rover. Once the map and command data is retrieved and processed, a Rover object is created and ready to run.

The run sequence for the Rovers is almost identical to Lab 1. One difference is that when a mine is hit and the Rover is about to execute the dig command, it makes a remote procedure call for the server using the GetMineSerial method and passes along the mine coordinates in the request. Once it gets the serial number of the mine, it continues with the mine() function of trying to find a valid pin. In Lab 1, the serial number was known already since it was part of the map preprocessing. Another difference is that after the program returns from the mine() function with the pin, another RPC is made to the ShareMinePin method to send the decoded pin to the ground control server. The final change is seen within the run() method of the rover class. Here, if the Rover explodes due to not digging when it hits a mine, there are success flags that are set to False, which indicates a failure. After the run() method has been through all of the commands or the Rover has exploded, the Rover uses the ReportStatus RPC to indicate its status to the server.

Conclusion

Google's Remote Procedure Call framework provides an interesting open source and platform independent solution to communication in distributed computer systems. As someone who has only ever worked with RESTful frameworks and had not heard of gRPC, I found it to be a very useful and uniform approach to message communication. To conclude, I was able to successfully execute both the server and client sides of my application to achieve the same base functionality as in Lab 1 but with a central ground station master node.