CS417 Project 1:
A Bulletin Board Service using gRPC and Google Protocol Buffers

Rutgers University

Due: October 17, 11:55pm

Introduction

In this project we will be exploring Remote Procedure Calls (RPCs). A remote procedure call is a when a program invokes a procedure to run in different address space as if it were a local procedure call. In this project we will implementing a Bulletin Board Service where a client will invoke RPCs to store, fetch, and delete bulletin board posts stored on another machine.

1 Intro to gRPC and Google Protocol Buffers

Google Protocol Buffers and gRPC are modern frameworks that originated at Google and were developed to build performant services. Google Protocol Buffers is a data serialization mechanism that has been proven to serialize and deserialize data between services faster than other comparable data formats such as JSON or XML. GRPC is an RPC framework that abstracts networks communication for developers. It has been developed to take advantage of the newer HTTP/2 protocol as well as use Google Protocol Buffers to efficiently connect services to each other and support more advanced communication methods such as bi-directional streaming.

To help developers take advantage of these underlying mechanisms within the framework, Google developed a compiler within Google Protocol Buffers that will generate source code in different languages based on an interface description using an interface description language (IDL). For these frameworks, an interface is described in a "Proto" file with a .proto extension. To describe the interface, developers must use the Proto IDL syntax. Here developers will define any messages that will be passed between services as well as any RPC services that will need to be supported. The Proto compiler will then compile the Proto file and generate any source code. In the case of Java, the compiler will generate java classes to represent messages that will be serialized using Google Protocol Buffers, and generate gRPC base classes that can be easily extended and implemented. For this project, we'll being doing just that, writing our own interface description and using the Proto compiler to build our own bulletin board service.

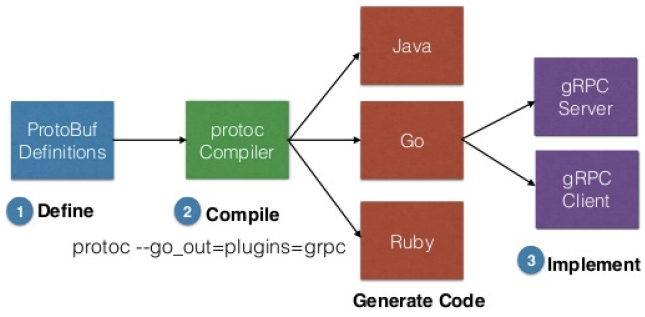

Figure 1: gRPC & Google Protocol Buffers Workflow

2 Project Stages

To implement a bulletin board service using gRPC and Google Protocol Buffers, we will break down this project into four stages:

- Stage 1 – Generating Interface Classes using gRPC and Protocol Buffers
- Stage 2 – Implementing the Bulletin Board Service
- Stage 3 – Implementing the Bulletin Board Server
- Stage 4 – Implementing the Bulletin Board Client

3 Stage 1: Defining the Interface and generating the service classes

Before we can implement anything, we need to define the interface for our service within a .proto file. We have to consider the messages our client and server will be passing back to each other. We will need a message to pass a post consisting of title and body. We will also need another message to represent a list of posts that the Server will send back to the client when the client asks for the list of posts. We will also need three RPCs: One for posting a post, one for getting a post, and one for listing the posts. To generate the service classes, follow the instructions to build your project in section 8.

In general, for stage 1 you will need to:

- Define messages within the proto file
- Define RPC services within the proto file
- Build the project to generate gRPC classes

Note: Make sure to read the entire project description, especially the section describing the client requirements as you will need to to get an idea of what RPCs should be defined in your Proto file.

4 Stage 2: Implementing the Bulletin Board Service

The Google Protocol Compiler will generate the base classes your service will use. This will include classes implementation of the messages you defined and a gRPC class that implements the base of your service. In this case, if generated properly, there would be a BulletinBoardGrpc class that would have been generated. Within this class there will be a static class called BulletinBoardImplBase. This class will have the RPC methods you have defined in the proto file. Your task is to extend this class with BulletinBoardService.java and override the RPC methods.

In general, for stage 2 you will need to:

- Extend the generated BulletinBoardServiceImplBase Class with BulletinBoardService.java
- Override all generated RPC methods
- Implement a system to handle posts from clients

5 Stage 3: Implementing the Bulletin Board Server

In order to use the service, you will have to create a server. The server will use the gRPC framework to instantiate a new Bulletin Board Service and bind it to a port. Your task will be to implement this within BulletinBoardServer.java

In general, for stage 3 you will need to:

- Figure out how to initialize and start a new RPC service

6 Stage 4: Implementing the Bulletin Board Client

In order to interact with the Bulletin Board Service, you will need to create a client. Your task will be to implement this within BulletinBoardServer.java. The client will have make a stub object and connect to the service. The client will be able to invoke RPCs using the stub object.

The client should then take in the following commands:

- post <Title> <Body> - sends a post to our Bulletin Board Server with a title and body
- get <Title> - gets the post and prints out the body
- delete <Title> - deletes the post of the given title
- list – lists all the post titles on the bulletin board

In general, for stage 4, you will need to:

- Figure out how to use the gRPC generated code to connect to the RPC Service
- Figure out how to take in user input and invoke the proper RPCs
- Format client output in a way that is readable and understandable

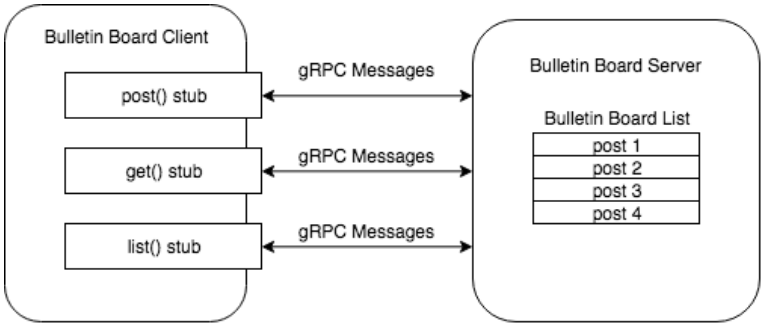

Figure 2: Simple Bulletin Board Service Overview

7 Getting Started

The following sections describe how to get started on the project.

7.1 Requirements

In order to complete this project you will need the following:

1. Java JDK
2. gRPC-BulletinBoard project template
3. Eclipse IDE (optional)

7.2 Downloading the Project Template

If you do not already have the Project Template ZIP provided, you can download the project template two other ways:

- Download directly from Github:
  https://github.com/DaveedDomingo/gRPC-BulletinBoard-Project/archive/master.zip
- Clone the project template using GIT:
  git clone https://github.com/DaveedDomingo/gRPC-BulletinBoard-Project.git

7.3 Importing the Project into Eclipse

To import the project into Eclipse, carry out the following steps:

1. In Eclipse Go to "File" > "Import"
2. Under the Maven folder, select “Existing Maven Projects”. Click Next
3. In “Root Directory”, browse to the gRPC-BulletinBoard-Project folder. If done correctly, there will be an entry under project window called /pom.xml
4. Ensure /pom.xml is selected.
5. Click Finish.

Notice: Although it is recommended, **it is not necessary to use the Eclipse IDE**. We are using Eclipse because it already comes prepackaged with Apache Maven. Maven is a tool that is used to help automate the building of a Java project. If you decide you do not want to use Eclipse you will still need Maven to build and compile the project. You can look up look up instructions on how to install the Maven tool on your particular operating system. We will mention how to use Maven in both Eclipse and Command Line in the following sections.

7.4 Project Template Overview

The following is the bare structure of the project template:

```
gRPC-BulletinBoard-Project
├── src
│   └── main
│       ├── java
│       │   └── com
│       │       └── bulletinboard
│       │           ├── BulletinBoardService.java
│       │           ├── BulletinBoardServer.java
│       │           └── BulletinBoardClient.java
│       └── proto
│           └── BulletinBoard.proto
└── pom.xml
```

Notice: If you imported the project into Ecllipse, additional folders may be present. This is due to Eclipse realizing that it is a Maven project, autogenerating extra folders that it may use.

Within the /src/main/proto directory there is one file:

- ***BulletinBoard.proto*** - Here is where you will define the messages and services for the bulletin board service using google protocol buffer’s proto3 syntax

BulletinBoard.proto

```
1  syntax = "proto3";
2  package com.bulletinboard;
3  option java_multiple_files = true;
4
5  // Implement Proto File Here
```

Notice that this file comes with a few lines already present to help get you started. Line 2 has been added to ensure that generated gRPC source classes will already be automatically added to your build path, reliving the need to explicitly import them within your implementation classes. You will still however need to figure out which other gRPC packages to import to use the generated code. Line 3 has been added to ensure the Google Protocol Buffer compiler will generate code into multiple files. Without this, It will place all generated classes sources in a single file containing multiple classes. It's recommended not to change these lines

Within the /src/main/java/com/bulletinboard directory there are three files:

- ***BulletinBoardService.java*** - This is where you will implement the RPC methods
- ***BulletinBoardServer.java*** - This is where you will implement the server application
- ***BulletinBoardClient.java*** - This is where you will implement the client application

You will notice at the root of the project there is a file called ***pom.xml***. This file is for Maven. Like mentioned in the previous section notice, Maven is a build tool used for Java projects and uses the pom.xml file to know how to build the Java project as well as know what dependencies need to downloaded before hand. **DO NOT EDIT THIS FILE**. The pom.xml file has been preconfigured to include the gRPC and google protocol buffer dependencies as well as preconfigured to generate gRPC code and Runnable JARs every time the project is compiled *for your convenience*.

8 Building the Project

To build your project, you will need to use Maven and run "goals". Goals are procedures that carry out actions related to the project lifecycle. There are three Maven goals you will need will to familiarize yourself with in order to build and compile your project:

- install - this goal downloads any dependencies for your project
- package - this goal compiles the project and packages artifacts.
- clean - this goal cleans the project of any artifacts and code generated by the project

8.1 Install

The install goal downloads any dependencies and plugins you will need for your project. You will need to run this goal when you first import your project so the nessesary gRPC and Google Protocol Buffer libraries can be downloaded. If you don't do this, your IDE may give you errors or you may get errors when you try and build the project.

- To do this in Eclipse:
  1. In Eclipse, select the project folder within the Package Explorer window.
  2. Go to "Run" > "Run As" > "Maven Install"
- To do this in Command Line:
  1. Navigate to the root of the project directory.
  2. Run the following: "mvn install"

8.2 Package

The package goal will carry out the Maven build process: compiling project code, packaging artifacts, as well as carry out any additional tasks defined within the pom.xml file. You will need to run this goal for two reasons. The first is to generate gRPC Java code. Within the pom.xml there is a task that invokes the Google Protocol Buffer compiler to take the proto file you defined and generate Java classes. These classes are what you will use to implement your service. The second reason to run the package goal is to compile your server and client code so that you can run them.

- To do this in Eclipse:
    1. In Eclipse, select the project folder within the Package Explorer window.
    2. Go to "Run" > "Run As" > "Maven build..."
    3. Within the Goals text box, type in "package"
    4. Click "Run"
- To do this in Command Line:
    1. Navigate to the root of the project directory.
    2. Run the following: "mvn package"

8.3 Clean

The clean goal will clean the project directory of any generated project artifacts and source code. Project artifacts and generated code are usually placed within a folder named "target". You will need to run the clean goal before you build your project with the package goal to ensure any files from your old code is gone. You can run the package goal without running the clean goal as it will just overwrite the existing files, but there may be some files that were not overwritten still lying around.

- To do this in Eclipse:
    1. In Eclipse, select the project folder within the Package Explorer window.
    2. Go to "Run" > "Run As" > "Maven clean"
- To do this in Command Line:
    1. Navigate to the root of the project directory.
    2. Run the following: "mvn clean"

Note: Sometimes when running the previous goals in Eclipse, it may seem like nothing changed within the project structure. An example would be the target folder not showing up after you built your project. This is because Eclipse may have failed to update the window to reflect the new files. If this happens, try refreshing by right-clicking the project folder and selecting refresh.

9 Running Your Project

While you are implementing your service, you may want to test your server and client applications.

- To do this in Eclipse:
    1. In Eclipse, right-click the BulletinBoardServer.java within the Package Explorer window.
    2. Run the server by selecting "Run As" > "Java Application"
    3. Repeat steps 1 and 2 but for the client class instead
- To do this in Command Line:
    1. Navigate to the root of the project directory.
    2. Navigate to the target folder where the runnable server and client jars should have been placed
    3. Run the server by running: java -jar BulletinBoardServer.jar
    4. Repeat step 4 but for the client jar instead.

Sample execution

The following is a sample of what the client output should look like:

Command Line

```
$ java -jar BulletinBoardClient.jar
Client started. Connected to service at port 5501.
Input a command: post  ‘‘Hello’’ ‘‘This is a message to the world’’
Input a command: list
Posts:
     1. Hello
Input a command: get ‘‘Hello’’
This is a message to the world
Input a command: delete "Hello"
Input a command: list
There are no posts available.
....
```

The following is a sample of what the server output should look like:

Command Line

```
$ java -jar BulletinBoardServer.jar
Server started. Listening at port 5501.
Posting post with title "Hello" and body "This is a message to the world"
Deleting post "Hello"
....
```

You will not have have to follow this format exactly. This is just an example. You can have the input and output be as formatted as much as you would like, given that it is simple and intuitive. What this means is that I should not need to ask questions or read any documentation to carry out the main functionalities the client should have.

10 Submission

To submit your project, have ONE of the group members submit two things:

1. A zip file containing your project
2. A pdf document consisting of the following:
    - List of group members and NetIDs
    - A short paragraph or two describing what you accomplished including details on how to use the client if you deviated from the original design
    - A short paragraph or two describing any issues you may have encountered and how you think you could have solved them. If you didn't have any simply state that you didn't have any issues.
    - A few sentences to answer each the following two questions:
        (a) In this project, we implemented a simple RPC service but we didn't have to explicitly implement mechanisms to handle multiple clients. Instead we just used the mechanisms the gRPC framework provides. How does gRPC handle multiple clients?
        (b) With the current implementation, if we add the requirement stating that each post title must be unique, we could end up with more than one post with the same title if two clients post at times close enough to each other. How can we modify our implementation to ensure this edge case won't occur?

11 Additional Hints

Tutorials are you friend: Follow gRPC tutorials to understand the syntax, design patterns, and development process when using generated gRPC code

Easy Method Overriding in Eclipse: After extending the base service class, you can easily override the methods in Eclipse by going to "Source" > "Override/Implement Methods...". This will show all superclasses and the inherited methods you can override. You can have eclipse automatically override any of the methods by selecting them and clicking 'Ok".

Combining Maven Goals: You may find yourself running Maven goals frequently to build the project. You might find yourself running them so frequently it may just interfere with your precious development time. Luckily you can group goals together. For example, since it is good practice run the clean goal before the package goal, we should group them. In Eclipse this is done by running a Maven build like we are running the package goal, but instead of writing "package" in the Goals text box, we write "clean package". In Command Line, this is done by navigating the root of the project directory and running "mvn clean package". These will both run the clean goal first then the package goal.

When in doubt, Maven clean install package: Maven projects are complicated. If you're wondering why your code may not be running or compiling as intended, re-run the Maven goals to make sure the proper dependencies are installed and you're building in a fresh workspace. This way, if it still doesn't work, you can be sure it was because of your own doing.

12 Useful Links and Resources

- **What is gRPC?** - https://grpc.io/docs/guides/
- **gRPC Concepts** - https://grpc.io/docs/guides/concepts.html
- **gRPC Java Quickstart** - https://grpc.io/docs/quickstart/java.html
- **gRPC Java Tutorial** - https://grpc.io/docs/tutorials/basic/java.html
- **gRPC Live Coding** - https://youtu.be/sdsabcI02LE?t=19m40s
- **Google Protocol Buffer Proto3 Syntax** - https://developers.google.com/protocol-buffers/docs/proto3

13 Additional Questions

If you have any questions about the project or are having any issues, email me at David.Domingo@rutgers.edu