Assignment 6
Distributed Systems

Version: June 28, 2021

Prerequisites

1. The assigned readings in module 6 on Canvas
2. Lecture videos from Canvas
3. Running and understanding the examples listed on the Canvas Module 6 page

Learning outcomes of this assignment are:

1. Understand basics of Distributed System
2. Understand gRPC in detail
3. Understand how to create a node for a distributed system

Preliminary things

I strongly advise you to work on Git and GitHub, to version control and also to practice. If you work on GitHub make sure your repository is private.
As always rather have a 50% full working version that compiles than try to implement everything and nothing really works.
Please watch the videos supplied with this assignment to get some more insight.

What you definitely need:

1. Structure: you will have to create one program. Does not need a subdirectory just your build.gradle file, README and of course your project sources. You should make sure to clone your repo after uploading and test things so you can be sure you pushed everything that is needed. Your project needs a README.md and a build.gradle file.
2. A README.md
   a) A description of your project and a detailed description of what it does and which requirements it fulfills
   b) An explanation of how we can run the program (hopefully as we want it to run), paste the command we should use into your readme so we can just copy paste to make sure to use it correctly
   c) Explain how to "work" with your program, what inputs does it expect etc.
   d) Design your calls and user interaction in a way that they are easy. Remember we have over 60 assignments to grade, design it so it is easy for you.

e) More details for the Readme will follow in the activities directly.

f) As always a screencast showing your program in action and showing what you accomplished

Activity: Distributed System gRPC (105 points)

Background

For this activity we will use sample code provided on Canvas, please watch the given video. The code shows you a server (Node.java) that provides 2 services already: echo and returning jokes (and adding a new joke). The EchoClient calls gRPC services from this server.
The server will automatically register with the Registry Server (see video and upcoming tasks), the EchoClient will also call the Registry Server and ask for services.
Before getting started: I would advise you to run the Registry, Node and Client on your system locally, so you see how it actually works together before starting to make changes.
For Task 1 I would advise you to comment out the Registry parts in the server and client so you can develop a simple client server application first. The registration comes later!
Your task will be to enhance this server with more services and also adjust the client so that the user can choose which services to use. You can change the code in any way you want or start from scratch if you prefer. You can leave the joke and echo service in there or remove them. That is up to you.
You do need to use the Protobuf files and the given protocol/services so we have compatibility between the clients and servers.

Task 1: Starting your services locally (50 points)

First analyze, understand and run the given code (see the provided video). You should see that there are already some simple services included to show you how to integrate different services into a server node with gRPC.
In the given code you will also see some more .proto files with defined services. Please read through the headers of these to understand what they are supposed to do. You are not allowed to change the .proto files! Your server/client needs to implement these services as is.
Your task is now to create a client server application where the server implements at least 2 of the services not yet included. You can choose between (tips or story) and (calc or sort), e.g. implement story and sort.
Constraints

1. (3 points) Must have: We need to be able to run the service node through "gradle runNode" which should use default arguments, and the client through "gradle runClientJava" using the correct default values to connect to the started service node!!!! If this does not work we will not run things locally on our end.

2. (15 points each service) Implement 2 from the 4 services that are given in the .proto files. Implement either calc or sort AND either tips or story. **Read through the Protobuf files and Readme for more details on how the services are supposed to work.**

3. (8 points) Your client should let the user decide what they want to do with some nice terminal input easy to understand, e.g. first showing all the available services, then asking the user which service they want to use, then asking for the input the service needs. Good overall client that does not crash.

4. (4 points) Give the option that we can run "gradle runClient -Phost=host -Pport=port -Pauto=1" which will run all requests on its own with input data you hardcode and give good output results and also of course shows what was called. This will call the server directly without using any registry. So basically shows your test cases running successfully. See video about Task 1 for more details.

5. (5 points) Server and Client should be robust and not crash.

Task 2: Inventing your own service (30 points)

Now it is time to create your own proto file and come up with a new service. For this you can work together with 1-3 of your peers to design the new proto file and service. You are only allowed to design the proto file with its service together not the implementation in your client/server. But if you design a protocol together then you can use each others service, which might be fun.
The service should be something small and fun (or big and fun) that fulfills at least 3 out of the following requirements:

- Service allows at least 2 different requests
- Each request needs at least 1 input
- Response returns different data for different requests
- Response returns a repeated field
- Data is held persistent on the server

Do not just do an add/sub but come up with something yourself. You can pitch ideas on Slack as well if you like.
Then of course implement your service into your client and server as another option.
10 points protocol design, 10 points client, 10 points server (all this robust, working and well described in your Readme and shown in your screencast).

Task 3: Building a network together (25)

Now we want to create a network of nodes and register them so others can access your services.

Task 3.1: Register things locally

The given code gives you a Registry where servers can register. You can run this through "gradle runRegistry" (which will run it on localhost). See the video for some more details on the Registry. In the end it provides 3 ports for 3 different protocols that it can handle, we will only use the grpc port!!!
Do the following:

1. MUST: Create a new version of your Node.java ==> NodeService.java and your EchoClient.java ==> Client.java. You should be able to call them through "gradle registerServiceNode" and "gradle runClient2" asking for the same parameters as the calls that were already in the given Gradle file for the original files. This server and client should use the Registry again, so in case you commented the Registry parts of the code in the previous tasks, include this code again in NodeService.java and Client.java.

   This call will use the Registry again, so is not the same as runClient from the previous task. These two gradle tasks should use default values so that they connect correctly!

2. Test this: Run your Registry, run your NodeService (you need to provide the correct host and port of course) – you should set this as default values for us. You should see a println on the Registry side that the service is registered. If you do not, try to figure out what happened (or did not happen).

3. Now, you should run your Client and check if it will find the registered services correctly.

4. (15 points) If all this works, adapt your client so it does not just call the service on the node you provide directly as was done in Task 1 but that the client can choose between all services registered on the Registry (in this case locally it will still just be your services. For testing purposes you can run a couple server nodes and register all of them to your local registry. You do not hard code which server to talk to anymore but use the following workflow:
   a) Client contacts Registry to check for available services
   b) List all registered services in the terminal and the client can choose one (preferably through numbering)
   c) (You should basically have this already) Based on what the client chooses the terminal should ask for input, eg. a new sentence, a sorting array or whatever the request needs
   d) The request should be sent to one of the available service nodes with the following workflow: 1) client should call the registry again and ask for a Server providing the chosen service 2) the returned server should then be used, 3) should send the request to the server that was returned, 4) return the response in a good way to the client
   e) Make sure that your Client does not crash in case the Server did not respond or crashed. Make it as robust as possible.

If you feel like it you can share the .proto file with the others on Slack in case they want to implement it on their client and or server!

Task 3.2: Registering your node online (7 points)

We want to build a service network together. This is for fun and we hope that our Registry can handle the traffic. It did handle it fairly well last time.

We have a Registry node running on ser321test.duckdns.org port 8080. You should register your Node on this server (only if your Node works and was tested in the previous section). You should keep your Node running and up so others can use it. The name of your node needs to be your asurite to get credit for this and your node needs to run until we are done grading so we can actually test it.
When you start your client now and use the above host and port for the Registry you should be able to see all registered services on our Register node (you should be able to do that anyway, even if your Node does not work).

Readme

You should have a good documentation in your Readme about what you accomplished and what you did not! This will be graded as well. Include also how to call each program, your screencast and all we need to see you actually did the work.

Submission

As always push your code to GitHub and provide the link to your repository in your Canvas submission.