Assignment 6
Distributed Systems

Version: April 15, 2024

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% completed, but working program that compiles than try to implement everything and nothing really works.
Please watch the videos supplied with this assignment to get some more insight. I think they will help to see what is going on since you get a lot of code already.

What you definitely need:

1. Structure: you will have to create one program. Does not need a sub directory - just your build.gradle file, README and of course your project sources that we need to run things. 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 correctly start up your system
   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 100 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 (100 points)

Background

Use the starter code provided, videos walk you through it in case you need help figuring things out. The code already includes some example services. The layout is having a Registry, and Nodes (with services) can register on this Registry. Clients can then ask the Registry about services and call specific services on Nodes. The Registry can also be turned on and a Client can communicate with a Node directly.
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, see the videos if you need help.
For Task 1 you can start the Node and Client with default values, which will be without a Registry (the Registry is not needed for this task). So only run "gradle runNode" and "gradle runClient" in separate terminals.
Your task will be to enhance the Node 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. 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 Nodes.

Task 1: Starting your services locally (60 points)

First analyze, understand and run the given code (see the provided video).
In the given code you will see some more .proto files with defined services. The headers in the .proto files include information about the services.
Your task is now to create a client server application where the server implements at least 2 of the services not yet included. Choose 2 services of the 4 given services to implement: flowers, qanda, follow, or weightTracker.
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 runClient" using the correct default values to connect to the started service node!!!! If this does not work we will not run things, this is already given so just do not change it.

2. (20 points each service) Implement 2 from the 4 services that are given in the .proto files, **choose from these 4 protos: flowers, qanda, follow, or weightTracker. Read through the Protobuf files for more details on how the services are supposed to work.** You can talk to your friends to coordinate which services you all want implement, so you can test each others nodes with your own clients. BUT do not exchange code!

3. (8 points) Your client should let the user decide what they want to do with some nice terminal input that is 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(s) the service needs. You should have a 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 hard code 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. **As another option instead of doing this you can write Unit Test similar to what you did in assignment 3 Activity 1 (also see example tests provided), where your Unit Test Class is basically your Client and calls requests on the server and then asserts the response.**

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. You are only allowed to design the proto file 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. Please put the name of the person(s) you worked together with at the top of your proto file (in the case where you did work together with someone).
The service should be something small and fun (or big and fun) which fulfill 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 (or similarly simple) 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 little network

Task 3.1: Register things locally (10 points)

The given code gives you a Registry where servers can register. You can run this through "gradle runRegistryServer" (which will run it on localhost). See the video for some more details on the Registry.
Do the following:

1. MUST: Create a new version of your Client, call this client "Client2.java". You should be able to run it using the command "gradle runClient2" using the same parameters as the task that was already in the given Gradle file for running the original Client. This client (Client2) should have the regOn flag set to true so it can connect to the Registry.

2. Test this: Run your Registry, run your Node with the regOn flag set to true. 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). See the video, I show how this can be done.

3. Now, you should run your new client (Client2) and see if it will find the registered services correctly.

4. If all this works, adapt your Client2 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) Client is showed all registered services it received from the Registry in the terminal and the client can choose one (preferably through numbering, you can filter by only the services that you actually implemented yourself)
   c) (You should basically have this already) Based on what service the client chooses the terminal should ask for input
   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 service the user has chosen 2) the returned Server should then be used (so the ip and port), 3) create a new channel for that ip, port and adjust the blockingstub so you can send your request to this Node, 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.

Task 3.2: Putting your node online (5 points extra)

We are not going to use the registry but for these points, put your running node with your services online. Post the command for Gradle how the client should be started to reach your node (so include the IP and port).
The first thing the client should do after connecting is to ask the node you are connecting to which services are implemented on this particular node. Your client should then match if it supports these services (you only implemented 2 so you only need to support those two services).
Test at least one other node that is posted online and provide feedback to that person.
If you are nice, then mention in your Slack post which services you implemented so others will know if their client works with your server.

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. As always, also submit a zip on Canvas which includes your project.