Project 02 - RPCs

Due: Monday Apr 13 11:59pm
(NOTE: I am unlikely to answer any questions between Apr 10 and 12)

In this project each group will be making the central server for a store's inventory system. You will need to write a server that supports XML-RPC and gRPC. You will have to also write demonstration clients that interface with the server and use a command line interface. You must write your code to reuse as much code as possible between the different parts of the system.

I will not be specifying the RPCs that must be made available. Instead you must come up with the APIs yourself and document them. The two APIs will likely be similar but not exactly the same. Documentation for each is:

- XML-RPC: write psuedo-code function definitions including names, types, and descriptions
- gRPC: use the proto file as your documentation - just make sure to add lots of comments

There will be lots of decisions to make about the API. For example, when updating the information about a product, do you have to send an entire new product, including fields that are not changing? Will you have a separate function for each field you could update? Special values that indicate a field is not updating? When getting multiple products, do you stream them (only available in gRPC), return them as an array, or paginate them?

Your inventory system should not lose its information when the server shuts down. The easiest solution to this, if you are using a pickle-able object (dict, list, or even a user-defined class) to represent your inventory then you can pickle it and save it to a file. On start-up it would unpickle the file. Just make sure that you catch the KeyboardInterrupt exception in your server to give yourself a chance to save your database.

Both of your servers run from a single file, it just exposes both servers. On the other hand, you will make two separate clients. These clients will have massive use of argparse (that code should be shared between the two different clients).

NOTE: It is completely acceptable for one person to work mostly on XML-RPC and one person to work mostly on gRPC, but you will need to coordinate just to make sure both servers can run at the same time from the same file, you aren't duplicating code, etc.

See the final page for details on what services must be available and the command line options must be implemented.

You can test your client and servers over the internet using ngrok (demonstrated in class). Eventually we will move to AWS, but that will come one week into the assignment.

Once you complete the client and server and you have it running on AWS you will compare the speed of the two different RPC methods. Do this by making a program that goes through many of the RPC functions and timing how long they take. Repeat for the other RPC method. Note that you must make sure that both start from the same “state” and since you should have some RPCs that add, delete, or modify data you will need a way to restore the database back for the next test. Your database should be quite full when running this (100s of products). Each timing run should take at least 15 seconds and you should run each from the same source (so one person in the group runs it). Run each of XML-RPC and gRPC 3 times and take the median.

Write a brief (½ single page) analysis considering the difficulty of implementing each of XML-RPC and gRPC servers and clients and the speed that each of them offers. Which would you choose if making an actual commercial application? Why? Submit this as a PDF with the timing results.

Rubric

XML-RPC + gRPC Server	20
XML-RPC API Documentation	10
gRPC API Documentation	10
XML-RPC Client	15
gRPC Client	15
Running on AWS	10
Timing results and analysis	10
Code Style	10

Inventory System

The inventory system must be able to keep track of products, the current stock, and the orders.

Each product has at least:

- ID
- Name
- Description
- Manufacturer
- Wholesale cost
- Sale cost

Products can be looked up by id or name. All products from a particular manufacturer can be retrieved. New products can be added (but they cannot have the same name as a previous product and the ID should be assigned by server). Old products can have all attributes updated except the id and name.

The current stock is the current amount of each product immediately available and shippable. This isn't necessarily a separate data structure from the products themselves. The client must be able to get a list of all in-stock items.

Each order has:

- ID
- Destination
- Date
- List of products and their counts
- If it is paid
- It it is shipped

The client must be able to get a particular order, create an order (ID should be assigned by server), amend an order (by adding/removing products, adjusting destination, changing the date, marking it as paid or shipped). You cannot add more of a product than is currently in stock. You must be able to query for a list of all unshipped and/or unpaid orders.