**Lab 1-Asterix Stock Bazaar Part 2 Design document**

* **Overview:**

This is a stock bazaar implementation to demonstrate grpc protocol. Protobuf is used to define and create services such as lookup, trade and update along with the message structure required to generate stubs of request and response messages. Client uses grpc stubs to invoke remote object calls from the server.

* **Architectue:**

The model is designed in a single server multiple client manner. A single server instance is running, to which rpc calls are made by multiple client processes. At startup client program requires the hostname and port number of machine on which server process is running. For server program, the port number to listen for calls is need to be specified, along with maximum threads for threadpool and maximum threshold of trading for each stock are required at startup.

* **Inbuilt Threadpool:**

Python’s ***concurrent.futures.ThreadPoolExecutor*** is used to create a threadpool at server side to handle incoming requests. The ***max\_workers v***ariable used to specify the maximum number of concurrent threads in the threadpool is set at startup as per user input. The thread pool executor has an inbuilt queue that stores incoming requests and assigns them to idle workers in the threadpool.

* **Code Design:**

Code consists of three parts: Client, Server, and stub files generated from proto file. The proto file consists of *StockBazaarService* definition which has three methods: lookup, trade and update. Each method has its own request message and response message definition containing different arguments.

Client folder has different client programs, one for updating stock prices, one program each to issue lookup and trade calls, and one client to issue both lookup and trade calls. In every client, arguments such as stock name, price, trade volume is randomly selected and calls are made using stubs. Some invalid stock names are also included to demonstrate invalid condition The connection is set up using grpc module’s “secure\_insecure channel”. The client requires hostname and port to connect to server. Both hostname and server are accepted from user at client program startup.

At Server side, stock catalog is implemented as a dictionary. Each stock has a maximum tradable volume threshold which is accepted from user at startup. Also, the program requires port to be entered at startup.

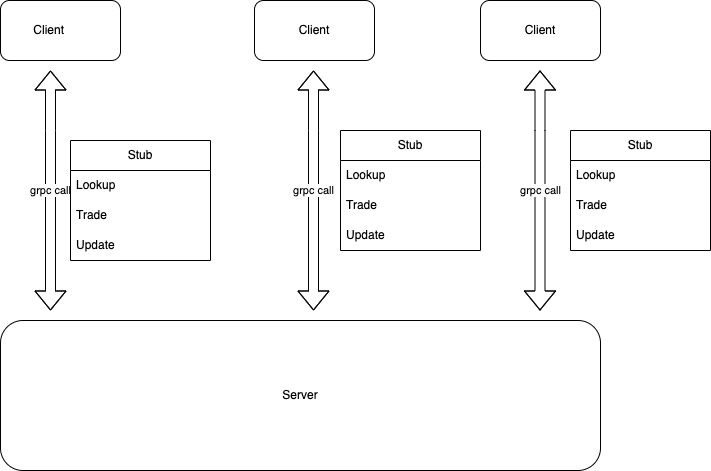
Lookup method accepts 1 argument: stock name, and returns price and traded volume of the stock name specified. It returns -1 if invalid name is specified and 0 if trading is suspended for that stock. Trade method accepts stock name, stock volume and transaction type as arguments, and returns 1 if trade is successful, 0 if trading is suspended for the particular stock and -1 if invalid stock name is specified. Update method accepts stock name and price as arguments, and returns 1 if stock price update was succesful, -2 if invalid price (negative value) is entered and -1 if stock name is invalid.   
 A threadpool is used to handle incoming calls to the implemented lookup, trade and update methods. The threadpool requires max workers to be set, which is accepted at startup. Locks are used in trade and update methods to access and change shared variables of stock catalog. Official documentation of grpc, threadpoolexecutor and protobuf was used to refer while developing the code.

* **Latency Measurement:**

Measurement of latency done for following steps:

1. Create a channel (grpc.insecure\_channel)
2. create a stub (rpc\_pb2.grpc.StockBazaarStub)
3. Make server call (stub.Lookup/stub.Trade/stub.Update)

Latency is measured for lookup and trade calls with update client running in background.



* References:
  + Wikipedia : <https://en.wikipedia.org/wiki/GRPC>
  + GRPC Python Documentation : <https://grpc.io/docs/languages/python/basics/>
  + Protobuf Documentation: <https://protobuf.dev/overview/>
  + Threadpoolexecutor : <https://docs.python.org/3/library/concurrent.futures.html>