Christopher Shi
33061961
cshi@umass.edu

Lab 1 Design Document
CS677 Distributed and Operating Systems

2023-02-27

1 Part 1 Implementation:

1.1 Configuring IP and Port:

- Executing the server.py file first will print an ip address that can be used in the client files to ensure server connection.

1.2 Thread Pool Configuration:

- The user is asked to input the number of desired threads in the pool at program start.

1.3 Server Implementation:

1.3.1 Thread Pool Naive Implementation:

- The thread pool was naively implemented through the use of a Queue data structure, the size of which is equal to the number of desired threads in the pool. The manner in which new requests are handled by the server is to add the accepted connection and address as a tuple (connection,address) to the queue. When the threadpool is initialized, the queue is passed to n number of workers where n is equal to the number of threads in the threadpool.

1.3.2 Worker Implementation:

- Workers are initialized as a thread from the threading package. In their run() subfunction, the worker takes a (connection, address) tuple from the front of the queue. The worker than executes a try/except/-finally decision. In the try portion, while the connection exists, the worker receives the message from the connection and decodes the request. Given that the requests are in the format of a string as follows, ”RequestType Arguement”, the worker splits the request string s.t. it can extract which request type is called as well as the argument. The worker then evaluates the argument on the given request type. (In this case the request type will always be ”Lookup”) If the returned value is -1, the worker encodes and sends a message stating that the worker could not find the price for the given company. For all other values, the worker encodes and sends a message stating the price of the stock for the given company. This continues until an exception is thrown, which occurs when the connection no longer has requests to send. Finally, the worker closes its connection with the current connection.

1.3.3 Datastructure Implementation:

- The prices of stocks are stored in a dictionary data structure. The names of companies are mapped to their respective stock price. The stock prices are configurable within the code.

1.3.4 Lookup Implementation:

- The lookup function simply calls and returns the .get(arg, -1) function on the dictionary storing the prices of the stocks. If the stock is not found within the dictionary, this allows for -1 to be returned.

1.3.5 Server Initialization:

- In the main function, a socket is created and bound to a unique port and IP address. A thread pool with a configurable number of threads is then created. The socket then listens for up to 10 clients. While listening, the socket stores the connection and address of any client during acceptance and calls the handle request function of the thread pool. (In order to store the (connection,address) tuple in the Queue. i.e. add the request to the thread pool).

1.4 Client Implementation:

- A list of stock names is first initialized containing [’GameStart’,’FishCo’,’Apple’] which represents two real stocks, and one stock to test for when an unknown stock is fed to the server.

- Each client then sends 1000 requests to the server in the following manner:
  - A stock is randomly chosen from the list of stock names to be the stock utilized for the current request.
  - A socket is created, and connected to the same port and IP address as defined by the server.
  - The request is first created as a string, by appending the work ”Lookup ” with the chosen stock name.
  - This request is then encoded and sent to the server.
  - The reply from the server is then decoded and printed on the client.
  - The socket is then closed.

1.5 Thread per Request:

Given each client repeats the following 1000 times:

1. Create socket
2. Send ONE request
3. Receive response
4. Close Socket

and that the request Queue is a queue of (connection,address) tuples and that the workers/threads each process one (connection,address) tuple at a time, we can see that the server follows a thread per request architecture.

2 Part 2 Implementation:

2.1 Configuring IP address and Port:

- Executing the server.py file first will yield a string in the format of ”ip:port”. Replace the value in grpc.insecure_channel() in client.py and updateClient.py with this string to ensure server connection.

2.2 Max Volume Configuration:

- The user is asked to input the max volume desired for each stock at program start time.

2.3 .proto Implementation:

2.3.1 Lookup

- Lookup takes in string stockname as input, and returns float stockPrice and int32 stockVolume.

2.3.2 Trade

- Trade takes in string stockname, int32 stockquantity, string tradetype as input, and returns int32 traderesult.

2.3.3 Update

- Update takes in string stockname, float stockprice, and returns int32 updateresult.

```
service StockBazaar{
rpc Lookup(lookupStockName) returns (lookupResult) {}
rpc Trade(tradeStockName) returns (tradeResult) {}
rpc Update(updateStockName) returns (updateResult) {}
}
message lookupStockName {
    optional string stockName = 1;
}
message lookupResult {
    optional float stockPrice = 1;
    optional int32 tradingVolume = 2;
}
message tradeStockName {
    optional string stockName = 1;
    optional int32 stockQuantity = 2;
    optional string tradeType = 3;
}
message tradeResult {
    optional int32 resultTrade = 1;
}
message updateStockName {
    optional string stockName = 1;
    optional float stockPrice = 2;
}
message updateResult {
    optional int32 resultUpdate = 1;
}
```

2.3.4 Datastructure Implementation:

- The stock prices, volume, and trading limit are stored in three separate dictionaries, respectively. The dictionaries map the stock names to their respective values. The starting values are configurable within the code as follows:

```
#Dict mapping company names to their respective stock price
stockPrices = {}
stockPrices["GameStart"] = 15.99
stockPrices["FishCo"] = 25.47
stockPrices["BoarCo"] = 5.28
stockPrices["MenhirCo"] = 22.58
#Dict mapping company names to the trading volume of their respective stock
stockVolume = {}
stockVolume["GameStart"] = 0
stockVolume["FishCo"] = 0
stockVolume["BoarCo"] = 0
stockVolume["MenhirCo"] = 0
#Dict mapping company names to the maximum trade volume of their respective stock
stockMaxVolume = {}
stockMaxVolume["GameStart"] = 20000
stockMaxVolume["FishCo"] = 20000
stockMaxVolume["BoarCo"] = 20000
stockMaxVolume["MenhirCo"] = 20000
```

2.4 Server Implementation:

2.4.1 Servicer:

1. **Lookup:**

   - With the thread lock, lookup first utilizes the .get(arg, -1) function on the dictionary containing the stock prices. As such if the stock is not found, a -1 will be returned. If the stock price is -1, return a stock price value of -1 and a stock volume value of -1. For any other value, return the stock price and the stock volume queried from the stock price and stock volume dictionaries respectively.

```
        #Takes the string stockName and returns the price of the stock and the trading
            volume so far.
        def Lookup(self, request, context):
            global stockPrices
            global stockVolume
            #Enable read-lock
            with lock:
                #Get stock price for the given stock name
                stockPrice = stockPrices.get(request.stockName, -1)
                #If the name cannot be found in dict, return -1/-1
                if stockPrice == -1:
                    return stockbazaar_pb2.lookupResult(stockPrice=-1,tradingVolume=-1)
                #If found, return stock price and stock volume
                else:
                    return stockbazaar_pb2.lookupResult(stockPrice=stockPrice,
                        tradingVolume=stockVolume[request.stockName])
```

2. **Trade:**

   - With the thread lock, extract the stock name, trade type, and stock quantity from the request. Query the current volume of the stock with .get(arg, -1). If the current volume is -1, return a value of -1 for the result of the trade. If the current volume is larger than the max trade volume as queried from the max volume dictionary, return a value of 0 for the result of the trade. If the trade type is Buy, increment the volume of the stock in the stock volume dictionary. If the trade type is Sell, increment the volume of the stock in the stock volume dictionary. If the trade did not exit earlier with -1 or 0, return a value of 1.

```
        #Buys or sells N items of the stock and increments the trading volume of that
            item by N.
        def Trade(self, request, context):
            global stockPrices
            global stockVolume
            global stockMaxVolume
            #Enable write-lock
            with lock:
                tradeType = request.tradeType
                stockName = request.stockName
                stockQuantity = request.stockQuantity
                currVolume = stockVolume.get(stockName,-1)
                #If stock cannot be found, return -1
                if currVolume == -1:
                    return stockbazaar_pb2.tradeResult(resultTrade = -1)
                #If the stock is suspended, return 0
                elif currVolume >= stockMaxVolume[stockName]:
                    return stockbazaar_pb2.tradeResult(resultTrade = 0)
                #If the trade type is Buy, increment stock volume
                elif tradeType == "Buy":
                    stockVolume[stockName] += stockQuantity
                #If the trade type is Sell, increment stock volume
                elif tradeType == "Sell":
                    stockVolume[stockName] += stockQuantity
                #Successfully buy/sell returns 1
                return stockbazaar_pb2.tradeResult(resultTrade = 1)
```

3. **Update:**

   - With the thread lock, extract the stock name and stock price from the request. Query the dictionary of stock volumes with .get(arg, -1). If the current volume is -1, return a value of -1 for the result of the update. If the stock price is less than 0, return a value of -2 for the result of the update. For all other values, change the value in the stock price dictionary for the current stock to the stock price extracted from the request. Return a value of 1 for the result of the update.

```
        #Updates the stock price
        def Update(self, request, context):
            global stockPrices
            global stockVolume
            #Enable write-lock
            with lock:
                stockName = request.stockName
                stockPrice = request.stockPrice
                currVolume = stockVolume.get(stockName,-1)
```

```
                #If stock cannot be found, return -1
                if currVolume == -1:
                    return stockbazaar_pb2.updateResult(resultUpdate = -1)
                #If the updated stock price is invalid, return -2
                elif stockPrice <= 0:
                    return stockbazaar_pb2.updateResult(resultUpdate= -2)
                #Update stock price and return 1
                else:
                    stockPrices[stockName] = stockPrice
                    return stockbazaar_pb2.updateResult(resultUpdate = 1)
```

2.4.2 Serve Implementation:

- A grpc server is initialized with a threadpool size of 5. An insecure port is added with a value of '10.0.0.246:5555' corresponding to a unique IP address and port, respectively.

2.5 Lookup/Trade Client Implementation:

- A list of stock names, ['GameStart','FishCo','BoarCo','MenhirCo','Apple'], which represents 4 real stocks, and one stock to test for when an unknown stock is fed to the server.
- The following is repeated 1000 times:
  - A random number from the integers [0,1]. 0 will correspond to a Lookup operation and 1 will correspond to a Trade operation.
  - The grpc channel is connected to, and a new stub is created.
  - A stock is randomly chosen from the list of stock names to be the stock utilized for the current request.
  - **Lookup:**
    - A lookup request is passed to the server with the selected stock name.
    - If the returned values are both equal to -1, we know that the stock is not in the database. We thus print "StockName not found in lookup."
    - For all other values, we print "Price of stockName: stockPrice. Volume of stockName: stockVolume"
  - **Trade:**
    - A list containing ["Buy", "Sell"] is initialized.
    - An item is randomly chosen from the list to represent the trade type.
    - A random integer is chosen from the range [1,100] to represent the quantity of stock to perform the prior operation.
    - A trade request is passed to the server with the selected stock name, trade type, and quantity.
    - If the returned value is equal to 1, we know that the trade was successful. We print a success message corresponding to the trade type. "Successfully bought/sold stockQuantity stocks."
    - If the returned value is equal to -1, we know that the stock was not found. We thus print "StockName not found in lookup."
    - If the returned value is equal to 0, we know that trade is suspended for the current stock. We thus print "Trade suspended for stockName."

2.6 Update Client Implementation:

- A list of stock names, ['GameStart','FishCo','BoarCo','MenhirCo','Apple'], which represents 4 real stocks, and one stock to test for when an unknown stock is fed to the server.
- The following is repeated 1000 times:
  - The grpc channel is connected to, and a new stub is created.
  - A stock is randomly chosen from the list of stock names to be the stock utilized for the current request.
  - A stock price is chosen from the range (-10.0,100.0) rounded to 2 decimal places. Negative values are included to test for invalid stock price changes.
  - A update request is passed to the server with the selected stock name, and stock price.
  - If the returned value is equal to 1, we know that the update was successful and print "Successfully updated stockName price to stockPrice"

- If the returned value is equal to -2, we know that the stock price was invalid and print ”Unsucessfully updated stockName price to stockPrice. Reason: Invalid stock price.”
- If the returned value is equal to -1, we know that the stock was not found. We thus print ”StockName not found in lookup.”
- A random value is chosen from the range (0.0,3.0) to represent how long to sleep.
- The client then sleeps for the chosen amount of time. s.t. the client updates stocks at random times.