Design Document

By: Aishwarya Malgonde, Sahil Jindal
Course: CS 677 | Professor Prashant Shenoy

Lab 1: Asterix and the Stock Bazaar

February 24, 2023

System Overview

We have implemented a over-simplified stock market back-end with a basic client-server application with two different approaches -

1. **Part1:** Has a network socket connection between client-server and a handwritten threadpool and queue implementation on the server side

2. **Part2:** Has a gRPC connection between client-server with a built-in dynamic threadpool

We have used the Python programming language in all parts of the project and for simplicity all the databases are in-memory. For both parts, we have a **setup.py** configuration file, where the user can configure system parameters before running it.

Design Considerations

1. **Code modularity -**
   We have used the following file system for easy debugging and understanding -
   a. **setup.py** - User can configure system settings before running
   b. **client.py** - Client part where the requests are running in a for-loop
   c. **server.py** - Server side part where the client requests are received and added to the queue
   d. **process.py** - Has the threadpool, queue and Lookup implementation. This script is accessed by server.py
   e. **client_update.py** (part 2) - Client part where the update price requests are running in a for-loop every 1 second

2. **Threadpool**
   a. **Part1** - All the threads in the threadpool are in ready-to-run state, in order to avoid the overheads of creating and stopping threads after each request. These threads are waiting in the get method of the Queue implementation
   b. **Part2** - For the threadpool, we have kept the maximum size of the queue fixed. The maximum concurrent rpc count which is set to 20 in our code defines that. So more than 20 requests are rejected and not put in the queue.

3. **Queue** (part1)- The queue has the producer-consumer concept implementation with a conditional lock on both sides. Whenever the put (producer) method receives an item, it notifies the get (consumer) method using the conditional lock

4. **Json serialization** (part1) - We are sending the buffer message as a json serialized string to keep the code coherent

5. **Database lock** - We are locking the database access for read and update operations in order to make it a thread-safe access and avoid race conditions

6. **Logging** - We have created a custom logger in setup.py which logs the time, script name, thread name and the message. Using these logs on the server side it is easy to see concurrency happening on the server side (attached the output image in the output doc)

Part 1

Server

On the server side we have the threadpool implementation, where we can configure a static number of threads at the start time. These threads start up and wait for requests on a request queue. This request queue is created by us and has a locking synchronization for any update made to the queue. The main server thread accepts requests over a socket connection and inserts them into the request queue, which causes one of the idle threads waiting on the conditional lock to be notified to process the request. We are making sure that the Lookup operation performed on the shared stock database (dictionary in our case) is thread-safe with the help of a global lock.

Client

The client component runs in a continuous for-loop sending sequential requests to the server. It connects to the server using a socket connection for every new request. It constructs a buffer message specifying the method name and stock name to be sent to the server and closes the socket connection after a response for every request is received. Closing the connection frees up the thread for the next request in the queue. Here the latency accounts for the time taken to send the message and receive the response from the server.

Part 2

Server

The server uses dynamic threadpool to **dynamically** resize the number of threads during runtime. We configure the maximum number of allowable threads which can run before the start of the server. The threads wait for the requests to come and every request is added to a queue before the threads

can process it. We made a global lock for the trade and update functions. This global lock is needed to get the requests from the queue and to update the share database.

It allows three types of requests Lookup, Trade and Update:

Lookup: It checks whether the stock is present in the database. If the stock is present it returns the price of the stock.
Trade: This type of request makes the trade of stocks. It updates the database by incrementing the trading volume. It returns the success or failure response depending on the request.
Update: This request updates the price of the stock and sends a success response if the name and price of the stock is valid.

Client

The client component is divided into two parts:

Trade and Lookup Client: One client trades and does lookup calls. It makes 100 queries for 6 stocks everytime it makes a call to the trade function. Two of these stocks are not listed in the database. The requests are sequentially sent to the server.

Update Client: The other client only updates the prices of the stocks every 1 second. It updates the price of each stock every second. To do this it first has to acquire the lock.

Part 3

Performance measurement

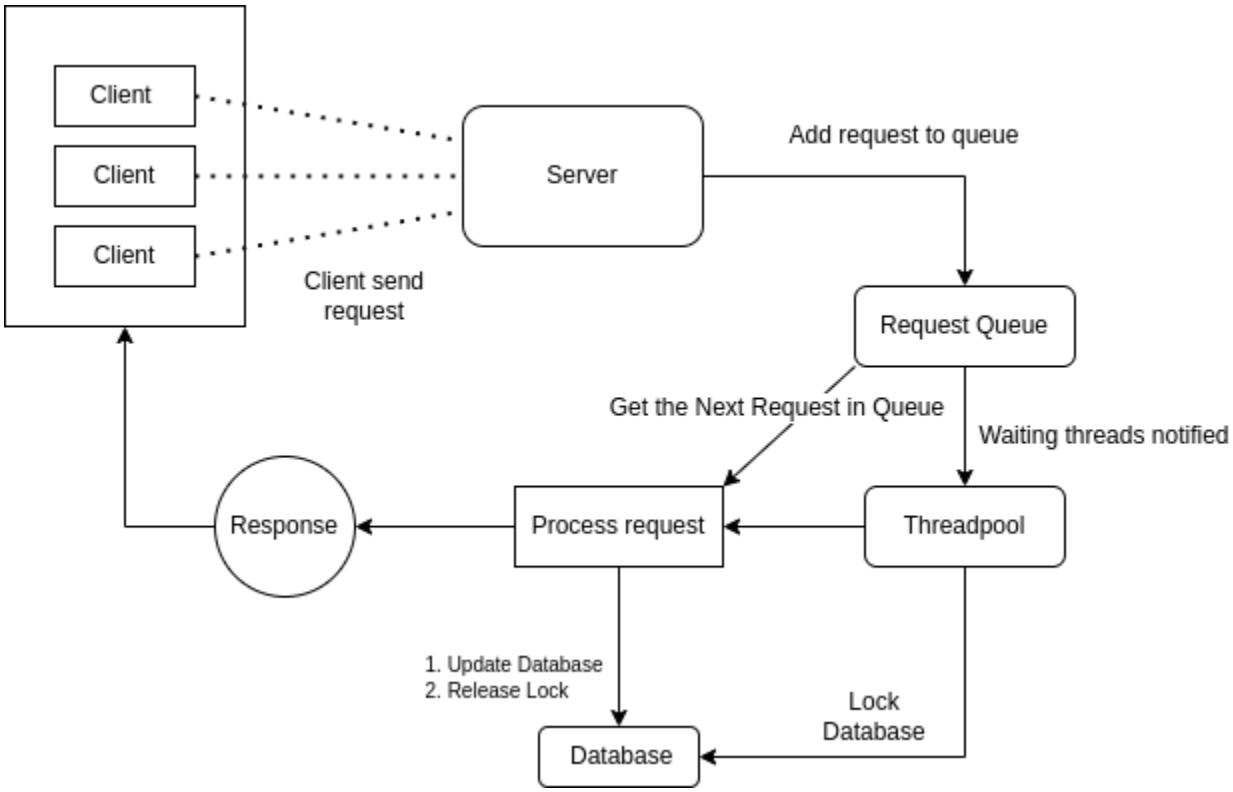
For both the parts we hosted the server on Edlab and spun multiple clients (varied the number of clients from 1 to 5) on our local machines using a bash script. The timestamps were logged in a log file and latency was calculated using them. We measured latency from client side as -

client latency = network latency + server latency

Where network latency accounts for sending and receiving messages, and server latency is the request processing time. We have varied the number of threads and number of clients from 1 to 5, and calculated latency for all the 25 scenarios by averaging between 500-600 requests in each scenario.

Design Diagram

Following is a high-level design diagram of our architecture for both parts. It conveys the relationship between the different software components, i.e, the client and server, and demonstrates how the server is processing requests.



References

Part 1 -

1. https://realpython.com/python-sockets/
2. https://realpython.com/intro-to-python-threading/
3. https://docs.python.org/3/library/queue.html
4. https://docs.python.org/3/library/threading.html#condition-objects
5. https://docs.python.org/3/library/threading.html#lock-objects

Part 2 -

1. https://www.velotio.com/engineering-blog/grpc-implementation-using-python