Design Doc

In our service we have 2 micro services in the backend namely Catalog and Order. We have one micro service in the front end and a client process to test these services by sending a sequence of requests as a session to the front end.

Backend

Catalog Micro service :

We are using HTTP REST protocol to serve our incoming requests from the front end service. We are maintaining a JSON file containing four stock company information. Whenever we start the catalog micro service, we are reading and loading the json file to in-memory data structure. We are initiating an HTTP server and all the incoming requests are handled by ThreadPool. Since it is a HTTP REST API server, we are serving our requests by exposing our resources using a URL path with GET method to front end service and PUT method to order service. GET method is nothing but a lookup method to lookup the stock by its name from the in-memory data structure and return its information with a structured JSON format. Since the in-memory data structure is shared by multiple threads, we are using read lock to prevent race-in conditions such that concurrency is achieved.

Similarly PUT method is nothing but a function, it accepts request body as JSON and from the JSON it reads the stock name to check if a stock name is valid or not, if it is valid depending on the transaction type, it checks if the requested quantity is less than the available quantity such that it increments or decrements the quantity based on sell or buy in in-memory data structure and also it increments trading volume. We are sending updated stock information to order service as a response. As multiple threads may write on the same shared data structure, we are using write lock here to prevent race-in conditions.

And also to persist our data in the disk, we are updating the db.json file with in-memory data periodically for every constant time. We have also managed to persist our data even if the catalog service abruptly stopped using keyboard interrupt.

Order Micro Service :

We are using HTTP REST protocol to serve our incoming requests from the front end service. We are maintaining a JSON file containing transaction history information (log file). Whenever we start the order micro service, we are reading and loading the log json file to in-memory data structure to track the transaction number. All the incoming trade requests from the front end service will be handled by a threadpool. Since it is a HTTP REST API server, we are serving our requests by exposing our resources using a URL path with POST method to front end service. POST method is nothing but a trade request, it accepts request body as json and communicates with the catalog microservice's PUT method passing the received payload from front end service. Based on the response received from catalog micro service, if it is a success, we are maintaining the transaction in in-memory data structure and incrementing the transaction number and we are constructing a JSON with transaction information and sending it to the front end service as a response. As multiple threads may write on the same shared data structure, we are using write lock here to prevent race-in conditions. And also to persist our data in the disk, we are updating the log.json file with in-memory data periodically for every constant time.

We have also managed to persist our data even if the order service abruptly stopped using keyboard interrupt.

Frontend

We are using HTTP REST protocol to serve our incoming requests from the client. SInce the client is trying to send the sequence of requests through the session, we handled it using a thread-per-session model. Since it is a HTTP REST API server, we are serving our requests by exposing our resources using a URL path with GET and POST methods to the client. If the front end service receives a lookup request as a GET request, then it communicates with the catalog micro-service whereas if it receives a trade request as a POST request, then it communicates with the order micro-service. The received response from the backend micro services is sent back to the client with a top level object of either “data” or “error” based on the type of response received. Initially to validate our thread-per-session model, when we sent a sequence of requests from multiple clients and observed that all the requests are being assigned to a single thread as the computation time is very small, so the thread is becoming idle very fast and incoming requests are assigned back to the idle thread. So to validate the thread-per-session, we have induced delay using sleep method in front end service which is making the computation time a little considerable making the other requests to be served by the new threads. Now we observe that all the requests from a single client session are assigned and handled only by a single thread until the session is closed from the client.

Client:

The client will send a sequence of requests as a session to the front end service. It first initiates the HTTP connection with the front end service and maintains the session context. Firstly the client will lookup for a stockname and based on the quantity of the stock in the response, it then sends the trade request based on the probability logic. If we keep P = 1, then for every lookup request, it almost sends an equal number of order requests to the front end service. The lookup request is a GET request and the order request is a POST request. For a lookup request of a stock, if the quantity in response is 0, then the session gets closed and creates a new session as it is in an infinite requests while loop.