Project Documentation

Callum Thomas – Max Gunton – James Ryan – Shawn Dhaliwal

University of Victoria

SENG 468

## Group Workload

### James

James built the initial blueprint for the transaction server, which was then primarily developed by Max. James also wrote the initial project documentation for the first project deliverable, which is the Project Plan word document included in the documentation folder of the repository. James acted as a consultant for much of the project, advising on design decisions or changes to the architecture. The overall project documentation was created by James, with assistance from each of the other group members.

### Shawn

Handled the set up of Docker and Docker-Compose, along with briefing the other group members. Shawn did the same for the use of HAProxy as the load balancer between clients and the transaction servers. Shawn and Callum both codeveloped the front-end system of our platform including the web server and web clients. Shawn also acted as a consultant during much of the project.

### Max

Max performed continual development on the transaction server throughout the project. The databases were also built by Max. Most of the transaction server functionality was built by Max, with some help from all other members of the group. Additionally, a significant amount of our performance improvement was the result of Max’s considerable amount of work on the transaction servers and databases.

### Callum

Callum and Shawn both codeveloped the front-end system of our platform including the web server and web clients. Callum also acted as a consultant for the duration of the project, assisting with making decisions on how to approach obstacles during the project.

## Architecture

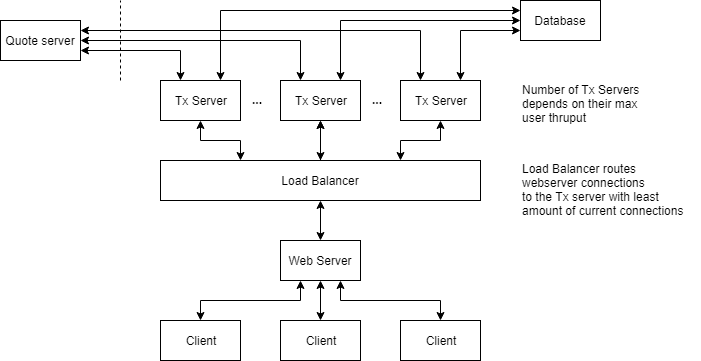
During development of stock trading platform, the architecture changed numerous times. The final product was derived from the original but is different as seen by comparing **Figure 1** to **Figure 2**.

Figure : Initial System Architecture

### Transaction Servers

The transaction servers handle the backend logic of our stock trading application developed in Python 3.7 and receives requests from the clients. When a client opens a TCP connection to the transaction server, a new thread is created dedicated to that client. When the transaction server receives a request from a client, it will process the request as needed and then send a response back to the client when finished.

When a transaction server needs to quote a stock, it will first check its local cache to see if it has a valid quote for the stock. If it does, it uses that stock for the transaction. If it doesn’t, it will then send a request to a stock cache server which all transaction servers can communicate with. If the stock cache server has a valid quote, the transaction server will use that for the current transaction request. If cache server does not have a valid quote for the requested stock, the transaction server will then request a quote from the legacy quote server, using that quote for the transaction and sending the quote to the cache server so all other transaction servers can access it.

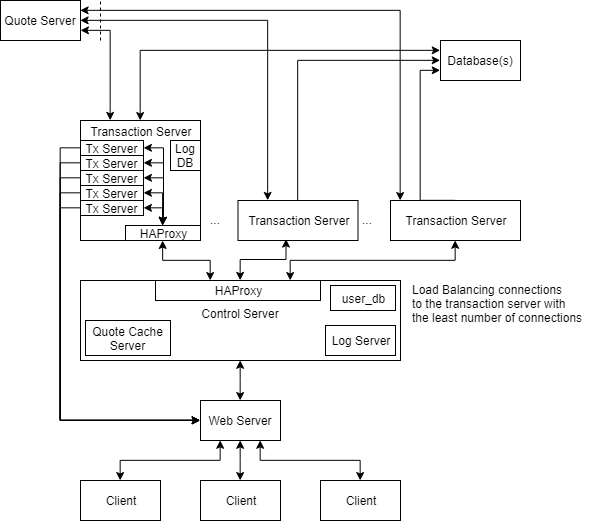
On disconnection caused by the clients, such as closing their web browsing or logging out of the website, the transaction server will close the thread that was opened for that client. If the connection is closed by the transaction server, for any reason including a server crash, the client will detect that disconnection and automatically open a new connection using HAProxy with a different transaction server. The transaction servers have been designed in such as way that we can scale up or down the number of servers on a machine and the number of machines running in our system based on the needs of the system. We were not able to implement by the end of this project; however future development would allow us to scale up or down on the fly dependent on current usage of the trading platform or on predicted usage based on previous usage statistics. For example, we would scale down the system outside of market trading hours, as demand for the system would drop outside those hours. We would keep it running at the capabilities needed, however usage of the system would not warrant as many servers as peak would. This technique would also be beneficial if this system were deployed on cloud servers, as this would reduce our daily cost overhead compared to running all the servers required at peak times at all hours of the day.

Figure : Final System Architecture

## Control Server

The control server contains a few services which are used by the individual transaction servers. These services include a quote cache server accessible by all transaction server, a log server and a database containing user information. In future iterations of this system, the control server could be scaled up vertically and horizontally if need were to arise based on usage of our platform, however that functionality is not built into the system at this time.

### Database

From our original design, we had split the database into three different types databases which would store different sets of information which are necessary for the functionality of our trading system. The databases are user information, stock information, and logging.

#### User Information Database

On the control server, there is a database server which hosts user information for our application. This database also holds the users stock information, including their amounts, sells, buys and triggers.

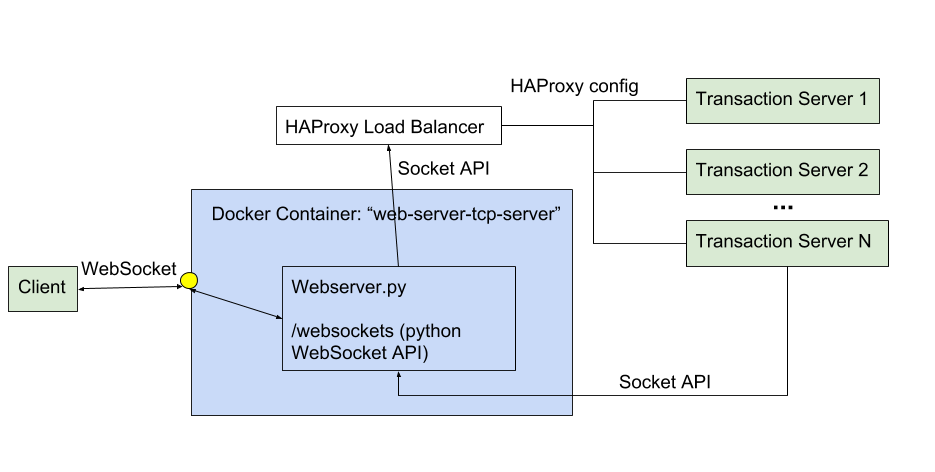
#### Logging Database

For logging events within our system, we decided to log to file. We made this design decision based off the sheer number of log events that needed to be store that it was much faster to log to a simple text file. Additionally, because logging is a write biased action with minimal reads it made sense to log to file. In future development, the plan would be to have an actual database which would handle storing all the log events which could then be sorted and queried for any reason such as auditing or analytics. This database would be updated once daily, ideally outside trading hours when the markets are closed which would reduce strain on the system. After offloading the logs to the database, the file copies of the logs would be cleared on the transaction servers to clear space for the next day’s logs.

#### Other Databases

Our architecture design allows for the ability to modify our system to include more databases. A future improvement we might implement is a decentralized database system that acts as a backup and auditing system. This server would receive information from the current servers and back them up on a periodic basis. In theory, these backups could then also be used to audit current databases to ensure proper operation.

### Web Server



### HAProxy Load Balancing

We use HAProxy to handle load balancing of customers between transaction servers which will handle their session. We have configured HAProxy balance connections to transaction servers based on the least amount of active connections to the transaction server. HAProxy also provides the ability to route new connections away from a particular transaction server in case of failure.