StockSim Design Doc

Authors: Nick Zullo, Shane Riley
Written: 2020-09-29

Objective

To develop algorithms that can produce a profit in the stock market over a period of time. Also, to create an environment where multiple algorithms can be inserted into the harness at once and have access to the simulation environment.

Goals and Challenges

To accomplish these objectives, first the testing environment needs to be created. Live data needs to be streamed into the environment so that the algorithms can process it. Also, this data should be stored so that past data can be accessed so analysis can be run on the past data. Once the data is ready, it must be able to be accessed in a simple, streamlined way by the algorithms. The data must be accessible as quickly as possible without delay so the algorithms can make decisions using the most up-to-date prices. This is important if the application ever gets connected to a real account, so the data used to make the decision matches the live data in the market as closely as possible.

After the testing environment is made, we will develop algorithms which will determine which stocks to buy and when to sell them. This is the biggest challenge as if everyone could figure this out, multi-billion-dollar companies would not need to hire so many people to try and solve this problem. With this in mind, our goal is to make a profit on at least half the time, not as much as possible. With longer time and testing, we will improve the algorithms, but the initial goal is 51% of days will be a profit. The biggest challenge for the algorithms will be if they can make a profit on days the overall market is at a loss. It is much easier to trade on days when everything goes up, but a profitable algorithm must be able to achieve a profit against the direction of the market.

Design Ideas

To obtain live data, the TD Ameritrade developer API will be used. This API provides 2 API calls per second for free. This means live updates to 2 stocks each second, or if more stocks are going to be tracked, 10 stocks updating every 5 seconds. The frequency of updates will be tested and adjusted after creating algorithms.

The API returns the data in a JSON format. We will build a JSON parser that will separate the API response into key:value pairs and store them in a hashmap. A hashmap is the natural way to store key:value pairs and does so very efficiently. We are willing to use more memory in exchange for an O(1) access time to the data, so the size of the hashmap is not a concern.

The data should be stored in a database. Databases allow for quick appending of data when a price is updated and will track a history of prices naturally. The date of the price should be

stored along with the price, and other information such as volume, bid/ask, etc. A database can easily accommodate multiple data points for each entry as columns with each row as the data returned from the API call.

Each algorithm should run in its own thread so that multiple can be running at the same time, such as 1 for buying decisions and 1 for selling decisions. This means synchronization techniques should be used on any shared resources. The total cash available in the account should be a critical section. Also, the currently held positions should be a critical section. The cash in the account needs to be protected because an accurate level needs to be read by each algorithm. The positions need to be protected in case only selling part of the shares or adding more shares to an existing position. The updated value stored needs to be accurate and making it a critical section will ensure this. Semaphores should be able to provide the data protection needed. Deadlocks should follow the deadlock avoidance strategy. The database can be accessed asynchronously because it is read only by the algorithms.

Various methodologies will be used to test the trading algorithms. First, if it is consistently not making a profit, something is not right. Next, we will employ Monte Carlo testing to build an equity curve of the trades made and evaluate them. More testing strategies will be found and developed later. Developing an algorithm is the most abstract part, so it is hard to write down now.

Timeline

Due Date	Deliverable	Owner(s)
9/26	API Integration	Nick
10/4	JSON Parsing	Shane
10/4	Database Integration	Nick
10/11	Thread Safety	Nick
10/11	API Wrappers and Database Operation Wrappers	Shane
10/25	Buying Algorithm(s)	Nick and Shane
10/25	Selling Algorithm(s)	Nick and Shane
10/30	Analysis	Nick and Shane