Distributed Retail Trading: System Design for Retail Traders

Kieyn Parks (kieynp2), Sarang Deotale(deotale2), Thomas Wang(mengran2)

ABSTRACT

Distributed retail trading is an approach using multiple computers to trade on the stock market in tandem using algorithms that autonomously executes the buying and selling of equities on the NASDAQ or NYSE stock exchange. This technique will enable the retail day trader to mask the trades or execute on both sides of the market simultaneously an action that is typically available to the market makers or large institutional traders. Though there are many types of stock market traders this paper will focus on the day/scalp trader. We will use the term day trader to cover both day trading and scalp trading activities.

1 INTRODUCTION

Computers have changed the way equities are bought and sold on the stock exchanges. With computer trade executions can now be handled with blinding speeds in the nanoseconds. With the increased speed of trading came increased volume and increased profits for the big players. Computers have also made it easier for retail traders to execute larger numbers of trades in the privacy of their homes; the socalled day trader.

Day trading involves buying and selling stocks multiple times a day; it can be from once a day to thousands of times a day with no trade held overnite. Day trades can be executed by individuals or automatically by computer algorithms known as algorithmic trading. Algorithmic trading is typically associated with large institutions while individual trading is associated with individuals [1].

1.1 INSTITUTIONAL TRADERS

The institutional traders are considered professional traders. Institutional traders participate in simple as well as highly complex trades, on behalf of their clients, which can be executed by a person or by a sophisticated computer algorithm; these algorithmic trades give those that use them a very distinct advantage over human executed trades in speed and frequency of execution [3] . Institutional traders also have the advantage of having their trades executed through "dark pools" [4] unlike the public trade executions of the retail trader.

1.2 RETAIL TRADERS

Retail traders are often referred to as individual traders that buy or sell stocks on the open/public markets. To gain access to the public stock market retail traders must go through a retail broker who can provide direct access, for a fee, or free access with terms and conditions to the stock exchanges. Free access to the public exchanges is the most inefficient way to day trade and it puts the trader at a disadvantage. Individual traders do not typically have access to sophisticated computer software to autonomously execute their trades. According to eToro, a retail broker, 80% of day traders lose all their money in the fist 1

year of trading with the median loss being -36% [2].

2 BACKGROUND

Retail day traders are at very distinct disadvantages on many fronts, as it is a contest between human executions vs computerised executions of big institutions and high frequency traders that can partition the market quotes making different prices appear to different entities at the same time this is called “quote stuffing”[8] .Also, high-frequency traders acquire the trading information of Retail day traders in order mount a strategy against them this is referred as “order flow”[10]. As far as we can tell there is no distributed system tailored for retail traders. Some retail trading platforms do offer algorithmic trading for their retail client but it is only an extension of the services they already offer which is to simply place buy and sell orders in an automated fashion when some condition is met. In our approach we are leveraging the distributed architecture to execute trades the moment a profit can be made which is the classic momentum strategy[11].

2.2 SOLUTION

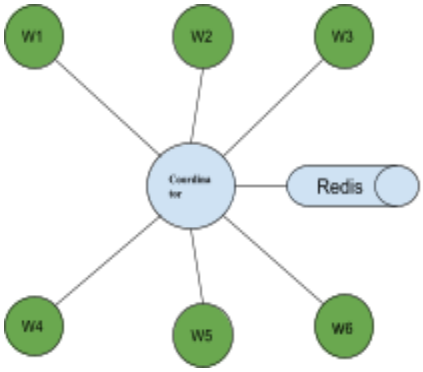
We propose to equalize the playing field by building a system of computers that can compete with the high frequency and institutional. This new paradigm will be called Distributed Retail trading. Distributed retail trading will consist of requisitioning available computers to form a computer cluster. This pool of computers will then be configured and tasked with the job of executing trades and trading in tandem autonomously using algorithmic trading.

The ability to trade in tandem will give the retail access to both sides of the margin (buy and sell simultaneously), an ability retail traders do not currently possess (*Figure1*). The system will also allow the retail trader to spread out the trades among many users (worker computer), that way their trades will most likely go under the radar of the high frequency traders who actually have the ability to attack trades using techniques such as **laddering** [7] which attempts to force the hand of the retail day trader. High frequency traders have many tricks they use to force out retail day traders [3]. Lastly the trader will have access to high speed distributed algorithmic trading that can open the door to many new and innovative trading paradigms.

Figure 1. Thinkorswim active trading trading margin (TD Ameritrade).

image of TD Ameritrade thinkorswim trading platform.
Traders can place their buy or sell orders on this screen.
Only one type of order can be placed, buy or sell in succession
but not simultaneously. The highlighted numbers is the last trade price
Of the underlying equity.

Figure 2- system architecture



Central coordination assigns workers and strategy for trading. When the trade is executed by the workers the workers inform the coordinator which keeps a record of the transaction in a redis database.

3 SYSTEM MODEL AND ALLOCATION

A coordinator will be given the number of workers to configure and trading objectives by the human user. The coordinator will configure the given number of workers , pass the trading algorithms and account information to the workers. The system will use the producer and subscriber model of Kafka for managing the data message flow. The workers will login to their individual accounts. The coordinate manages any faults that occur during setup and will switch out any worker that does not respond within a given amount of time. All records are kept in a key-value store database Redis and is managed by the coordinator. Redis is a high speed in memory database. The system architecture in Figure 2. The pseudocode is available in Appendix A.

4 FAULT MANAGEMENT AND PERFORMANCE OF SYSTEM

The system needs to be reliable in terms of guaranteed performance, and fault tolerance. To achieve desired performance level horizontal scalability aspect needs to be brought while designing each component. Below are high level design aspects

1. Database level - Sharding and replications. Consistent hashing mechanism can be used to come up with dynamic sharding of the database, and 3 replicas of each shard to achieve fault tolerance. The replication and read/write operations can be done via quorum based approach
2. Caching - most frequently used data, upto 98% would be kept in cache to give high throughput, cache also need to be replicated with 3 replicas to achieve fault tolerance.
3. Microsevice level replications and load balancing will solve the problem of throughput when multiple requests need to be addressed, the load balancer routing could be round robin or load aware.
4. Latency addressing - with above strategy we have addressed application level performance however network delays could be an issue in this application considering the sensitivity of time bound operations, this will be addressed via dedicated private network tunnel between inter datacenter traffic. The high priority for this application network traffic can be given via defining appropriate class of servier.

5 CHALLENGES

Since the system will rely on many computers coordination trades timing and latency will be of ultimate importance. Latency is endemic to all networked devices

and is an ongoing issue for the internet. Most applications running over the internet are latency tolerant due to the OSI type architecture and TCP/IP collision detection and the ability to resend data packets. High-frequency traders can trade at 64 millionth of a second [5] and any delay on the retail side can result in failed execution and even losses. We may solve the latency issue with consensu techniques and block chaining. Another challenge will be tuning the trading algorithms manually; this would require spending a lot of time in a trial and error mode first in a simulator and then on the live platform which will incur losses. Also, it will be cost prohibitive to execute this system in a live environment.

6 GOALS

- Successfully configure cluster
- Successfully Trade in Tandem
- Successfully hot-swap failed machines

7 INITIAL EXPERIMENT DATA

The initial test showed communication between the workers and coordinator are sub 1 ms which will be suitable for our application. The largest file size transmissions happen during the setup phase and do not exceed 5Mb . During the actual trading the data size will be a simple text of 100Kb or less. There are no delays or latency observed with data size of 30 requests per second with 8 actors and 1 coordinator experiment.

8 Workflow Demo

The demo system we built completes the following workflow. The controller program inserts sharelist.txt, which contains buying or selling commands, into Redis, the database we use to log command history. Then the controller program reads commands one by one from Redis and publishes each command to the Kafka topic, which consumer program will be listening on. Here is a screenshot of executing the controller program.

```
ITUB    buy     40      ALL
BKRKF   sell    50      ALL
TSLA    buy     60      ALL
FCEL    sell    70      ALL
CLF     buy     80      ALL
T       sell    100     ALL
PBR     buy     10      ALL
MSFT    sell    20      ALL
PLUG    buy     30      ALL
INTC    sell    40      ALL
X       buy     50      ALL
OXY     sell    60      ALL
AAL     buy     70      ALL
All Commands stored into Redis
Reading Commands From Redis and Publish to Kafka Topic:
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
Message Sent
```

Then the consumer program, named actors.py, will receive each command from the controller by listening on the Kafka topic. The consumer will execute each command(just printing the command) and log it in Redis. Here is a screenshot of executing the consumer program.

```
Read and Execute the Commands Published by Coordinator:
value=NIO       buy     10      ALL
['NIO', 'buy', '10', 'ALL']
Command Logged
value=AMC       sell    20      ALL
['AMC', 'sell', '20', 'ALL']
Command Logged
value=AAPL      buy     30      ALL
['AAPL', 'buy', '30', 'ALL']
Command Logged
value=F sell    40      ALL
['F', 'sell', '40', 'ALL']
Command Logged
value=TME       buy     50      ALL
['TME', 'buy', '50', 'ALL']
Command Logged
value=BAC       sell    60      ALL
['BAC', 'sell', '60', 'ALL']
Command Logged
value=GE        buy     70      ALL
['GE', 'buy', '70', 'ALL']
Command Logged
```

9 TRADING PLATFORM

The trading platform utilized is quantconnect lean which is a simulated open source trading environment real with historical data.

10 Solution Evaluation and Future Work

The experiments we conduct show little delay, which is fast enough for trading. However, the commands we passed to the machines are generated by hand. In a real world setting, asking a human user to compile a long list of commands could be a speed bottleneck. Although the system could distribute and execute a lot of commands very fast, it would take human lots of time to generate those commands. Therefore, in the future, we need to incorporate sophisticated trading algorithms into our system. The trading algorithms could be computationally heavy. Therefore we might need to reconfigure resources of our system to fit the trading algorithms.

Reference

1. Kristina Zucchi (2020) Institutional Traders vs. Retail Traders: What's the difference? retrieved on 2/28/2021 from. (www.investopedia.com)

2. Mark Lyck (2020), Why 80% Of Day Traders Lose Money. Retrieved on 2/28/2021 from (https://marklyck.medium.com/why-80-of-day-traders-lose-money-78d51b10fe25#:~:text=According%20to%20the%20stock%20platform,quitting%20within%20just%20two%20years)

3. Prableen Bajpai (2020), Strategies And Secrets Of High Frequency Trading (HFT) Firms retrieved on 2/28/2021 from (www.investopedia.com)

4. James Chen (2020) Dark Pools, retrieved on 2/28/2021 from (https://www.investopedia.com/terms/d/dark-pool.asp)

5. Eric Reed (2021), What Is High-Frequency Trading. Retrieved on 2/28/2021 from (https://smartasset.com/investing/high-frequency-trading#:~:text=High%2Dfrequency%20traders%20can%20conduct,than%20any%20human%20possibly%20could)
6. Andrew Bloomingtal (2020) Market Maker, retrieved on 2/28/2021 from (https://www.investopedia.com/terms/m/marketmaker.asp)

7. Jason Fernando (2020) Laddering, retrieved on 2/28/2021 from (https://www.investopedia.com/terms/l/laddering.asp)

8. Wikipedia, quotes stuffing, retrieved on 2/28/2021 from (https://en.wikipedia.org/wiki/Quote_stuffing)

9. Wikipedia, Carrier-sense multiple access with collision detection, retrieved on 2/28/2021 from (https://en.wikipedia.org/wiki/Carrier-sense_multiple_access_with_collision_detection)

10. Trade-dale (2020), Beginners Guide To Order flow, retrieved on 2/28/2021 from (https://www.trader-dale.com/beginners-guide-to-order-flow-part-1-what-is-order-flow/)
11. Adam Barone (2019), Introduction to Momentum Trading, retrieved on. 5/09/2021 from (https://www.investopedia.com/trading/introduction-to-momentum-trading/)

Appendix A

Pseudocode - Python

```
Class Setup:
# coordinator: configure workers
x = Get user input number of works
y = Get user input trading strategy (1-N)
Ticker = Get user input stock ticker symbol
strategy = Get user input strategy


setWorker(x)
        workerCount = 0
        FOR i in range(x):
                If good communication
Worker[i]
                        Configure Worker1
                        workerCount++
                        Set flag on Worker1
to 1
                        If workerCount == x
                                BREAK loop
                else:
                        Continue
        Return workerCount

# ensue that we have an even number of
workers
checkWorker():
        If workerCount is even:
        else:
                Remove 1 worker from pool
        return

# workers: worker registration
regWorker():
        Registar worker with redis database
        return

# coordinator : send trading strategy to
registered devices
sendStrat(y):
        SEND: trading strategy to registered
devices
        CONFIRM: receipt of strategy.
        Return

# coordinator : Get stock information
Class Trade:

Static : Price
Static: ticker
Static : quantity

setTicker():
        ticker = Get user input ticker symbol
from user
        quantity = Get user input quantity of
shares to buy
        Return [ticker, quantity]

# coordinator: trade execution
setExe():
        tradeType = Get user input for buy /
sell trade
        tradeInfo = getTicker()
        Price = Get user input price

# coordinator: get trade info and send to
workers
getTradeInfo():
        Return [tradeType, price, ticker,
quantity/numberWorkers]

# coordinator:
sendTradeInfo():
        getTradeInfo()
        SEND: tradeInfo to workers

# worker -TBD-: execute trade
Class Execute:
```

Business Plan: Distributed Retail Trading

Introduction
Distributed retail trading platform is a stock trading platform designed for retail traders. One disadvantage that retail traders face is that ability to trade on both sides of the buy/sell margin simultaneously. Another disadvantage retail traders face is that their trades and other personal information is sold to high frequency traders who use the information to work against the individual day trader.

The Proposal
We propose a system which will use distributed computing to autonomously place trades in tandem or to simply synchronise the trades for the retail trader. This new paradigm will give the retail trader super powers to compete with the high-frequency traders or institutional traders.

The Technology
As far as we know there is no one offering this type of service. Individual Retail traders typically use a retail broker that allows unary type trades: Either buying or selling stock but not both buying and selling at the same time manually or autonomously: Algorithmic trades.

What will we need to begin with?
Access to 6 - 10 networking ready computers running linux or window operating systems loaded with python or c++ IDE with intel or AMD cores; number of cores is not important for this application. The machines can be hosted in the cloud or locally.

Revenu
This service can be sold to novice or professional individual day traders who will be set up with the basic platform on a monthly fee basis. Additional revenu can come from leasing out highly sophisticated algorithms on a per transaction basis similar to serverless technology for high volume traders. Monthly fee can depend on service levels.
Fee structure:

- Monthly level fees: $150- $400, Serverless fees: $0.10 - $0.30 per transaction

Main Points
The system will address some disadvantages that the individual day trader encounters. By giving the day trader the ability to spread out the trades over a number of computers may result in the trades going under the radar from the prying eyes of the high-frequency traders. The system will also give the day trader the ability to appear on both sides of the buy sell margin going both long or short the market depending on market momentum. Lastly the trader will have access to high speed distributed algorithmic trading that can open the door to many trading paradigms.