ProtoBot

Crypto Pricing Resource

29

Team Leader: Peter Parianos
Team Members: Azamkhuja Ashrabkhujaev, Tony DeProspo, Brian Salvoro, Eric Shells
Faculty Advisor: Dr. Haodong Wang
Electrical Engineering and Computer Science, Cleveland State University, Cleveland, OH 44108
p.parianos@vikes.csuohio.edu

Abstract

The goal of this project was to develop a client-server model where the server took information from an Application Programming Interface (API), specifically Binance, which would return a message in JavaScript Object Notation (JSON) of the current buy and or sell price at the top of the order book for a specific cryptocurrency. The server then processed the raw data and calculated two Simple Moving Averages (SMAs). Data based on the SMA crossover analysis was used to ascertain whether to buy or sell. After that, the server then sent a message to the client. The client provided end users with informative notifications based on that message from the server.

Introduction and Background

- **Financial Performance Indicators**
- Cryptocurrency values have had extreme ups and downs, with these variances, someone who can predict this market stands to make a healthy profit.
- Investor Attention and Market Momentum are the 2 biggest indicators, ProtoBot strictly focuses on Market Momentum.
- A moving average is a measurement of Market Momentum. Using a crossover analysis you can compare the two moving values to derive an indicator.

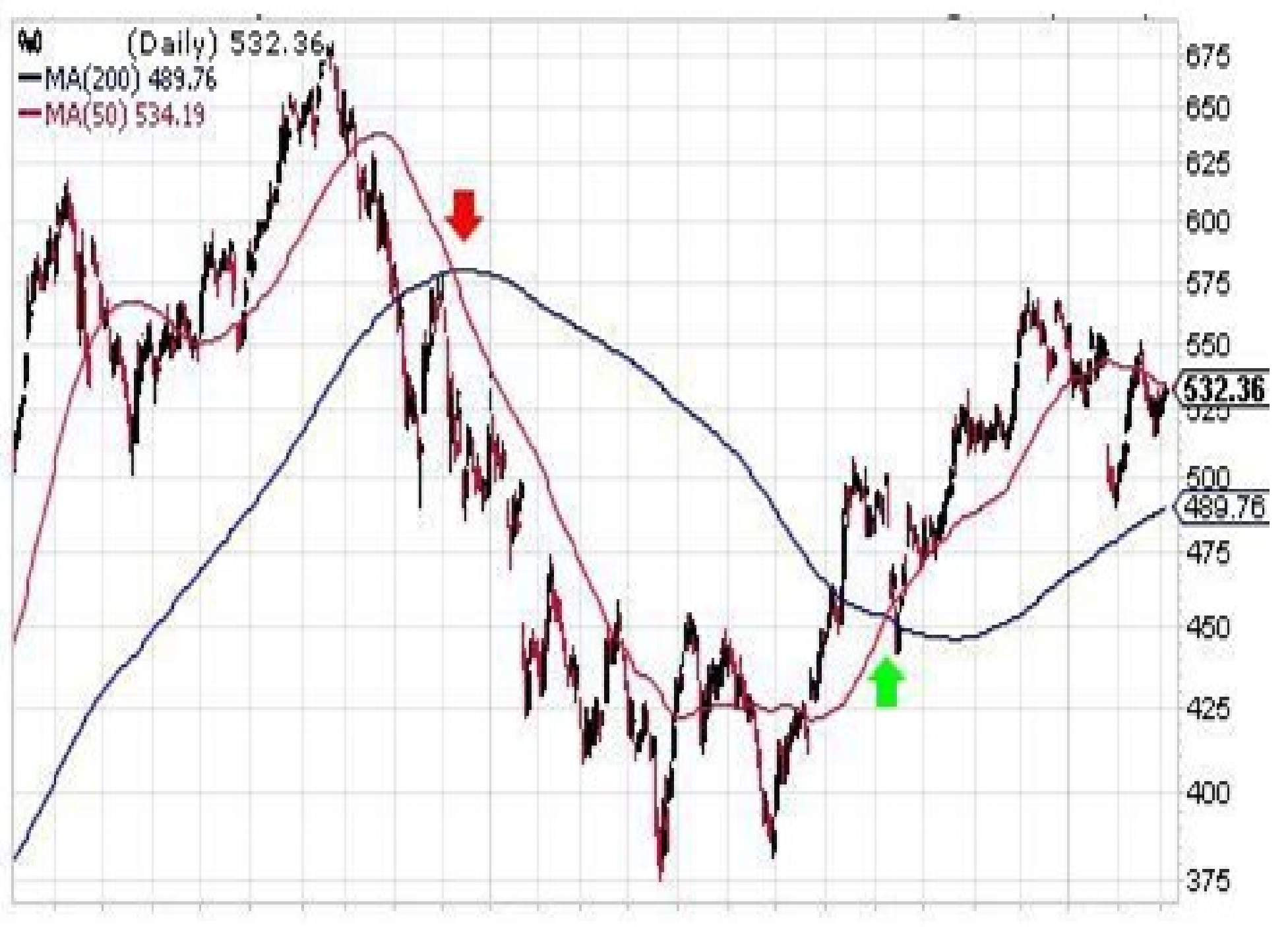

Figure 1: SMA Crossover Graph

- **Simple Moving Averages**

```
length = len(rawtable)
  j = length / 2
    for entry at index i in rawtable:
      total += float(entry['Close_Price'])
        if j == (i + 1):
          averageMov_ten = total/j
      averageMov_twenty = total/length
```

- ProtoBot uses 10-SMA and 20-SMA values. If 10-SMA crosses over 20-SMA it's an uptrend, meaning buy. If 20-SMA is greater than 10-SMA it's a downtrend, indicating to sell.

System Design and Functions

- **System Overview**
- Follow a server-client model to relay messages and data.
  - Server and client developed using Python programming language.
    - Server performs calls to Binance Exchange API for data acquisition.
    - Client sends GET requests to server for updates.
- Network communication established via Transmission Control Protocol (TCP) through Flask API.
- Server and Client applications are hosted on Azure Cloud Services.
- SQLite Database is used to manage and store users account information.
  - Account security managed with *bcrypt* library to securely hash passwords.
- Azure Web Application used to display fetched data and handle account authentications for mobile and desktop.
- **Program Functions**
- *rawtab()*:
  - Method that is run within a currency thread.
  - Maintains our data structure (list of dictionaries) in plain old memory.
- *calcMovAvg()*:
  - Calculates two moving averages, 10-SMA, 20-SMA.
- *get_historical()*:
  - Function can either get from Binance API or local Comma Separated Value (CSV) file.
- *convert_format()*:
  - convert from our CSV format on disk to custom format (list of dictionaries).
  - converts from Binance kline format(via API) to dictionary format.
- authentication - *createUser()* and *verify()*
  - get username and password then uses SQLAlchemy API to add to database.
  - check if hash equals the one in the database.

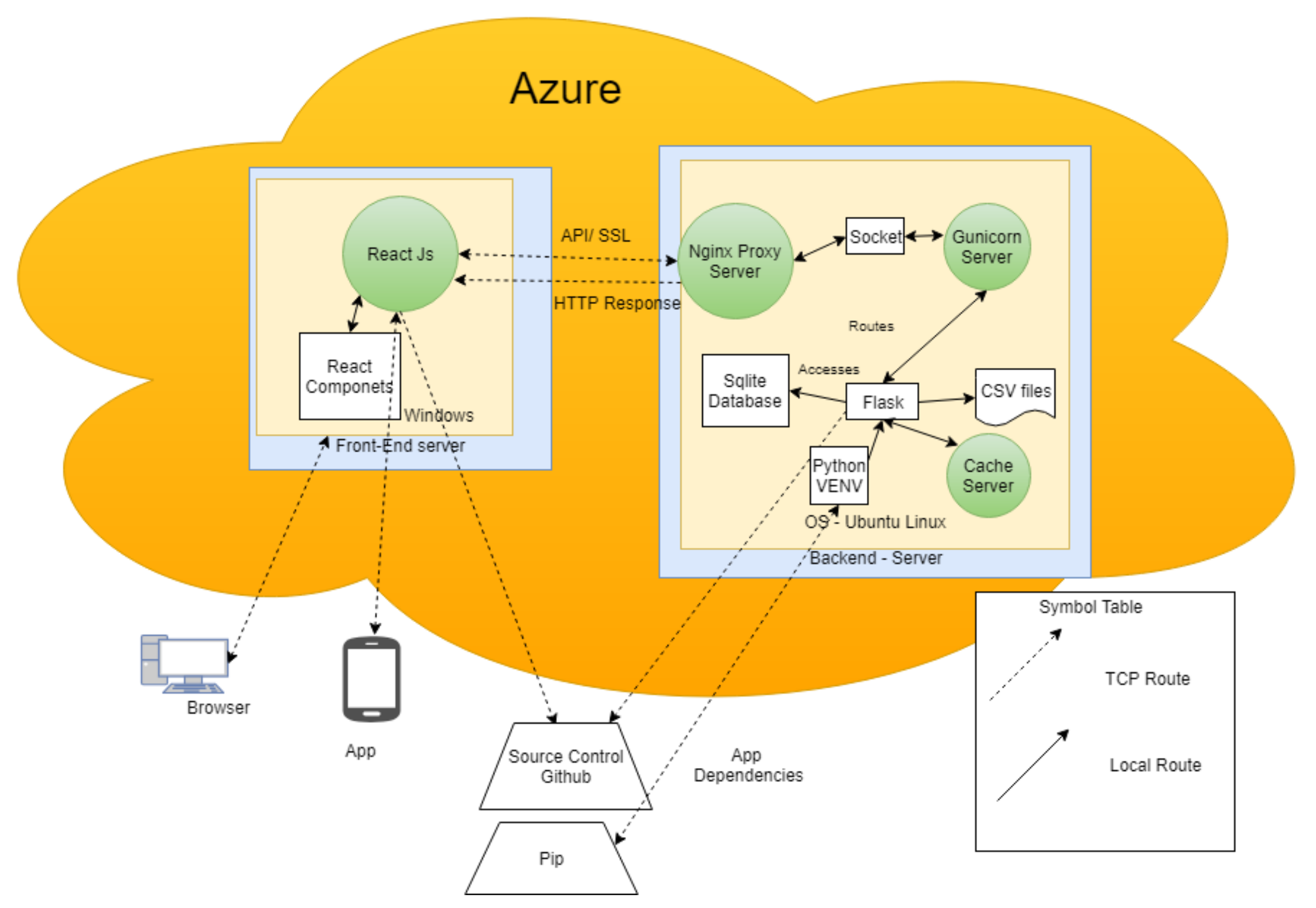

Figure 2: System Architecture Diagram

Experimental Results

- **Accuracy of Indicator**
- Financial calculations consist of Simple Moving Averages: 10-SMA, 20-SMA.
  - Both stored in a separate list to be used to determine the difference between each time interval.
  - ProtoBot predictive indicator has a success rate of 52% derived from a total of 2 years worth of financial testing.
- **Up-to-date and Relevant Price Information**
- Financial data is received through the use of an online cryptocurrency exchange
  - ProtoBot interacts exclusively with Binance Exchange via its REST API.
  - ProtoBot utilizes the *klines()* function, which returns *opening price*, *closing price*, *volume for a time period*, *number of trades*, as well as other financial data.
- Requests to the API can be made at varying time intervals (minutes, seconds).
- Improved fetch results by storing all historical data locally in a CSV, rather than pulling it from API. Performance was also increased by preprocessing defined time intervals in separate CSVs.
- **Account Security**
- Account credentials are stored in a database using SQLite.
- Passwords are salted and hashed in memory, salt and hash are stored.
- Authentication is managed through token value on server.
- **Testing**
- Stress testing the backend was completed using Locust, an automated testing tool.
- Various stress tests were conducted on our backend API and frontend.
- Preprocessed CSV files increased ability to handle 1000 simultaneous users.
- Having data ready in memory managed by threads proved to be faster.

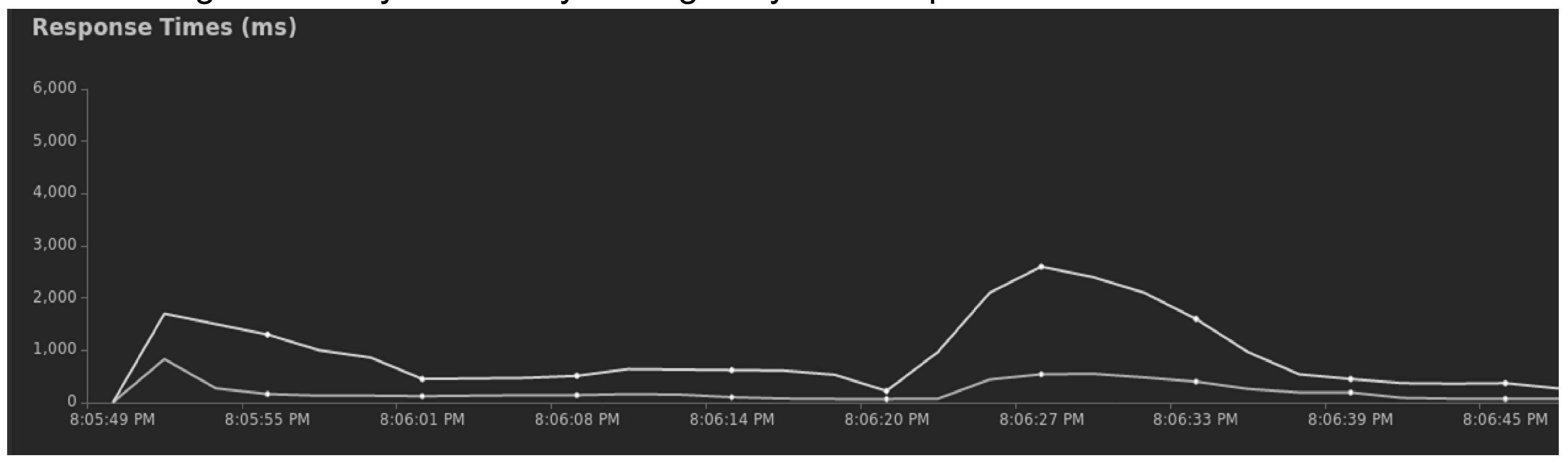

Figure 3: Server Stress Test with 1000 users sending requests on update

Conclusion and Future Recommendations

Conclusion

- Planning is key. Acting without a fully formed plan led to several reworks and refactorings.
- Indicator is not as accurate as initially hoped, knew it would not be easy.
- Further research into indicators and new innovations might be required.
- Nearly all objectives were met successfully, client notifications were not implemented.

Recommendations

- Instead of using SQLite and several CSV files, use a full-fledged database server for data handling and account storage.
- Use a combination of several more trading indicators for a better prediction
- Use machine learning to further increase accuracy of indicator.