DEPARTMENT OF COMPUTER SCIENCE

Implementation of a Distributed and Threaded Exchange Simulator

[redacted]

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Thursday 18th May, 2023

Abstract

This dissertation focuses on creating and utilising realistic market simulators for academic research in algorithmic trading. The simulation aimed to incorporate real-world latency and the effects of trader algorithm execution times. It presents a novel implementation of a distributed and threaded market simulation capable of handling over 400 traders simultaneously in a single market session. The thesis also includes an analysis of results from two sets of experiments: a balanced multi-trader market and pairwise trader tests.

In summary, the achievements of this research include:

1. Development of a novel distributed and threaded market simulator capable of hosting seven different types of trading algorithms.
2. Integration of the works of Bradley Miles[22] and Michael Rollins[30], combining their contributions to create an enhanced market simulator.
3. Conducting two experiments to analyse the impact of latency and trader execution times on profitability, both individually and in combination.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

[redacted]Thursday 18th May, 2023

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Professor Dave Cliff.

Chapter 1

Introduction

1.1 Context

1.1.1 Financial Markets

The exchange of goods and services between individuals or groups has existed across the ages. The earliest evidence of which being from 130,000-100,000 years ago, in the utilisation of obsidian in tools in Tanzania despite being 200 miles away from the nearest source. This suggests the existence of an exchange system that enabled individuals to acquire this material for tool production. Throughout history the two most prominent forms of exchange are trade and gift-giving. The value of these goods was determined by their utilitarian value or symbolic and associated prestige, as dictated by society. As time went on, commercial trade emerged with the primary goal of generating income. These trades took place through a market, where the value of items was determined by their supply and demand[34].

Between 700-500 BCE, metal coins were introduced, bringing about the use of a common currency and simplifying trade transactions. Throughout history, the fundamental concept of trading has remained unchanged, but the methods of conducting transactions have undergone significant evolution, transitioning from small-scale local trading to large corporations engaging in international markets[8].

Nowadays, the New York Stock Exchange (NYSE) is renowned as one of the world's most influential stock exchanges, but it was preceded by earlier market systems. Venetian moneylenders in the 1300s were the first to sell debt issues to investors and lenders, establishing the practice of brokering[32]. In 1531, Belgium had a functional financial exchange[2] where brokers and moneylenders facilitated transactions involving debt issues for businesses, governments, and individuals. While stocks did not exist at that time, the exchange operated with promissory notes and bonds. Notably, there were partnerships generating income akin to stocks[32].

By 1602, the Amsterdam stock exchange, which had previously focused on commodities trading, expanded to include equities, making it the first modern securities market[32]. Later, in 1698, the London Stock Exchange emerged in England, where coffee shops served as venues for publishing listings of stocks, currencies, and commodities, including company issued shares[11].

Since then, financial markets have played a vital role in the global economy by facilitating the allocation of resources between savers and borrowers[26]. They provide opportunities for investors to earn returns on their savings and for businesses to access capital for growth and operations. Today, financial markets have expanded to include various types of financial instruments, such as derivatives, foreign exchange, equities, and bonds, allowing traders to engage in a wide range of transactions[25].

Exchanges play crucial roles in the modern economy, facilitating capital formation, managing risk[26], and enabling price discovery through supply and demand[40], and providing opportunities for economic growth[23]. The significance of financial markets will be explored, their functions in today's economy, and the benefits of academic research in this field are further discussed in a later section.

The advancement of transmission technology in the 1960s, including the shift from analogue to digital switching systems and the introduction of the Electronic Data Interchange (EDI) protocol[8], paved the way for the digitization of financial transactions. This progress was further accelerated by the establishment of Instinet in 1969, which offered electronic trading platforms to automate trade execution. The NASDAQ, founded in 1971, marked a significant milestone as the first automated electronic quotation system[5].

1.1.2 Algorithmic Trading

The adoption of electronic trading led to the emergence of algorithmic trading[19] and high-frequency trading (HFT)[5].

While electronic communication networks (ECNs) were available in the 1970s[16], algorithmic trading did not take off until the 1990s. One of the earliest systems was the Designated Order Turnaround (DOT) which was introduced in 1976 to route trader orders to specialists residing on the exchange floor[31]. Enhancements such as direct market access in 1980 and the evolution into a comprehensive execution management system in 1993[16] increased the popularity of ECNs within the financial sector[21], increasing accessibility to algorithmic trading. Early on in the development of trading algorithms, it was only used by established institutional investors and hedge funds who had the means to create proprietary algorithms as access to high powered computers was restricted to those with the means. These simple algorithms were used to execute large orders more efficiently and cheaply than manual trades[9].

With the increasing power, accessibility, and affordability of computers, algorithmic trading techniques started gaining popularity. These strategies enable trading at speeds thousands of times faster than traditional human-to-human stock trading.The motivation behind algorithmic trading was to automate and enhance trading strategies like arbitrage, intermarket spreading, market making, and speculation[29].

Nowadays, due to its accessibility, such trading is commonly used among many types of market participants from individuals to small investment firms[18]. Its takeover in financial markets has caused significant impact. On one hand it has lowered transaction costs and increased liquidity[14], on the other hand volatility triggered by automated trading agents (unintentional or otherwise)[4] and potential for market manipulation has caused widespread concern[10]. By 2009 algorithmic trades made up more than 60% of all trades in the US[28]. In the following year, high-frequency trading, a type of algorithmic trading, held a market share ranging from 9% to 40% in Europe, while in the United States, it accounted for a market share ranging from 40% to 70%[13].

1.1.3 High Frequency Trading

High frequency trading practices involve executing a large number of trades at extremely fast speeds using advanced algorithms and high-speed trading infrastructure. These practices often rely on computer algorithms to analyse market data, identify trading opportunities, and execute trades within microseconds or even nanoseconds[17].

High frequency traders typically aim to profit from small price discrepancies, market inefficiencies, and short-term price movements. They may employ various strategies such as statistical arbitrage, market making, and latency arbitrage[17].

One of the driving factors behind this project is the significance of latency arbitrage, as emphasised by the research conducted by Wah[39]. Her work sheds light on the importance of understanding and studying latency arbitrage[39], which serves as a key motivation for undertaking this project (a full discussion on the project motivation can be found in the below section). Wah provides evidence that high-frequency traders have ample opportunities to generate profits through latency arbitrage. Her paper estimates that solely in the year 2014, latency arbitrage opportunities amounted to roughly \$3.03 billion[39].

However, these fast-paced systems can also lead to cascading effects that escalate rapidly and become difficult to control. There is a growing concern that the proliferation of high-frequency trading could exacerbate price volatility. Rapid market fluctuations, known as flash crashes, have been witnessed in different markets. These events include the 2010 flash crash, the 2014 US sovereign bond prices 'flash rally', and the 2015 market spike after the removal of the Swiss Franc-euro peg[10].

The 2010 flash crash was largely attributed to high-frequency trading practices. Within a mere 20 minutes, the Dow Jones Industrial Average experienced a significant decline of approximately 9%[17]. Furthermore, research into the 2015 incidence has highlighted the substantial involvement of algorithmic trading in diminishing liquidity within currency markets[4].

Research in the field of trading is crucial due to the fact that this type of algorithmic trading is likely to continue growing, regardless of whether people support or oppose it. Its substantial effects cannot be disregarded, emphasising the importance of further exploration.

To understand how High-Frequency Traders (HFTs) exploit latency arbitrage, it is crucial to grasp the fundamental significance of two key factors: the speed at which algorithmic execution takes place and the impact of network latency. These factors play a pivotal role in the world of trading, exerting considerable influence on the success and profitability of trading strategies. By understanding and optimising both the speed of algorithmic execution and minimising network latency, traders take advantage of latency arbitrage opportunities.

1.2 Motivation

When it comes to promoting and supporting the economic and social growth of society, the stability of financial markets and systems maintains a position of utmost significance. These markets and systems act as important pillars that support the development and operation of economies by facilitating the effective distribution of funds, resources, and investments. Financial markets aid in the production of wealth, jobs, and economic progress by offering a framework for the exchange of products, services, and financial instruments[41].

Furthermore, societal growth and financial market stability are tightly related. A healthy financial system encourages confidence and trust among citizens, organisations, and businesses, which encourages innovation, entrepreneurship, and economic participation. It promotes social mobility and overall well-being by making it possible for people and households to access capital for housing, education, and personal development[27]. Furthermore, strong financial systems are essential for advancing government programmes and public policies that tackle social issues including poverty alleviation, income inequality, and sustainable development[20].

On the other hand, the collapse or instability of financial systems and markets can have negative effects on the economy and society as a whole. Market disruptions, financial crises, and systemic breakdowns have the potential to cause devastating economic consequences, pervasive unemployment, financial hardship, and social unrest. Such crises can have particularly severe effects on disadvantaged groups, worsening social inequality, escalating poverty, and limiting inclusive progress. The Great Recession, a significant economic downturn that began in 2008 as a result of the Global Financial Crisis, serves as a striking example of the wide-ranging effects of market failure, shedding light on how widespread and profound the effects are when financial markets become unstable[41].

Taking into account the widespread use of algorithmic trading in today's financial markets, as was previously described in Section 1.1.2, it is obvious that this type of trading has a considerable impact on the health and stability of these markets. As mentioned before in Section 1.1.3, removal of the Swiss Franc-euro peg in 2015 and the significant market volatility witnessed during the 2010 Flash Crash highlighted the importance of closely monitoring and regulation of algorithmic trading practices. Regulatory bodies and academia were particularly drawn to the implications and potential risks associated with high-frequency trading, emphasising the need for comprehensive control measures.

It is not surprising that there exists a significant academic interest in studying the intricacies of these economic systems and trading practices. Despite the importance and demand for research in these areas, it can be difficult to obtain the data required to carry out investigations.

There are two primary methods used to gather data for study in this area: by collecting real-world observations or by creating synthetic observations using simulations, both of which have serious drawbacks. Using publicly available data from financial exchanges is appropriate for static statistical research of market patterns and volatility, but it is fundamentally wrong for simulation trials where participants (trading agents) respond to replayed data. The use of past fixed data limits the capacity of the subjects to affect the simulation's state, which is unrealistic outside of high liquidity markets for things like stock prices. Before adopting suggested policies or methods in the actual world, practical evaluations are necessary, even though theoretical evaluations are possible.

Due to financial and regulatory restrictions, it is difficult to carry out these experiments in real-world markets, even if they might be funded by market participants with large budgets like Hedge Funds, High-Frequency Traders (HFT), and Market Makers. This presents challenges for academic researchers working with limited funds. As a result, the predominant method in academia for conducting real-world assessments while creating new trade and policy methods is synthetic data simulations. Currently, the Bristol Stock Exchange (BSE)[7] and its expanded variations, Distributed Bristol Stock Exchange (DBSE)[22], and Threaded Bristol Stock Exchange (TBSE)[30], are the main open-source and freely accessible simulations in this field.

The Bristol Stock Exchange (BSE), a limited agent-based limit-order-book market simulator that contains several automated trader tactics, is the basis for these two works. BSE runs experiments in a virtual simulation time defined by the user. It employs a time-sliced approach to evaluate traders' strategies, ensuring each agent executes its algorithm exactly once per virtual second. While BSE has been successful as an educational and research tool for testing simple trading algorithms, it abstracts away some key features of real-world exchanges, which DBSE and TBSE aim to address.

DBSE, developed by Miles, tackles the issue of the zero-latency assumption in BSE's virtual simulation time. It adopts a real-time simulation method and a cloud-native design, allowing remote startup of clients worldwide, with each client capable of running up to 40 trader agents. The physical distance between clients and the exchange introduces real-world latencies, resembling those found in actual markets. The inclusion of real-time latency significantly impacts profits among traders in different geographic locations, highlighting its importance in creating a realistic market simulator. However, DBSE still inherits BSE's time-sliced evaluation approach for each trader's algorithm at remote clients.

In contrast, TBSE addresses the trade-off between a trader's algorithm complexity and execution time. It employs a parallel and synchronous evaluation of each trader agent's strategy, with each trader running on a separate thread. Through one-vs-one experiments, TBSE demonstrates that a simpler but faster algorithm can outperform its more complex counterpart, challenging the established dominance hierarchy of trading algorithms in prior literature. This highlights the incorrect assumption in both BSE and DBSE that the trader's strategy execution time is negligible, emphasising the need to model this aspect for a realistic market simulator. However, TBSE does not address the client-exchange latency factor, making it unsuitable for latency-based research on real world markets.

More in depth discussion on the technical details and functionality of each simulation can be found in the Technical Background Chapter.

Although DBSE and TBSE both make significant contributions to realistic market simulation, each of these separate applications lack important features that the other application provides. DBSE addresses real-world latency, while TBSE focuses on asynchronous trader execution.

As mentioned earlier, the significance of trader execution time and network latency is discussed, with latency arbitrage serving as a prime example. To thoroughly investigate techniques like this, it is crucial to use a simulation that enables traders to be executed in parallel, allowing for an evaluation of the effect of their execution times on profitability. Additionally, the simulation needs to also incorporate realistic network latency, ensuring that different traders experience varying levels of delays. By addressing these requirements, researchers would be able to more effectively analyse and assess the impact of both trader execution time and network latency on the outcomes of trading strategies, thereby enhancing the understanding of latency arbitrage and related trading practices. The study of techniques such as latency arbitrage, as demonstrated by Wah's research, holds significant importance.

Therefore a synthesis of DBSE and TBSE would result in a more realistic market simulator that encompasses the desired characteristics, thereby becoming a potentially valuable research tool in the mentioned topics.

1.3 Project Summary

The project aims to develop a distributed and threaded market simulator based on the previous works of Cliff[7], Miles[22] and Rollins[30], with the objective of demonstrating its capability to handle upwards of 400 of traders seamlessly in a single market session. The planned architecture of the project will incorporate notable features such as remote data logging, configurable settings through a unified configuration file, automatic client synchronisation and exchange reset via a TCP server, and the ability to queue multiple experiments and runs. The system should be designed to be scalable, establishing the groundwork for scaling up the system using technology such as Kubernetes clusters in future work. Once the distributed and threaded system is implemented, a series of experiments will be conducted to analyse the impact of latency and asynchronous trader execution. These experiments aim to compare the market interactions on the distributed and threaded version of BSE with those observed on its previous iterations, drawing inspiration from the research conducted by Miles[22] on DBSE and Rollins[30] on TBSE.

1.4 Challenges

Due to restricted permission with the provided AWS Academy labs, the project was initially restricted to only two regions which sufficed for initial testing purposes but meant wasn't able to test the system with varying levels of latency between clients. This would pose a challenge later on in the project as a key aspect of future experiments required testing the system with varying levels of latency between clients. Contacting administration revealed that the permissions could not be changed, leading to the need to migrate work to a personal AWS account. While lab accounts allowed easy addition of credits, personal accounts incurred charges directly to the bank account. Consequently, it became necessary to carefully calculate a more accurate budget and plan all experiments to avoid exceeding it. The lab account served as a testing ground before conducting actual experiments on the personal account, but unfortunately, the other available region became locked on the lab account, resulting in the loss of work accomplished up to that point. In order to perform latency-sensitive testing and ensure network connectivity between regions, migration to the personal account was undertaken. However, concerns arose about the potential risks associated with using one's own account, as there was no fail-safe mechanism to prevent incurring significant expenses if an experiment was accidentally left running. Therefore, ensuring proper system shutdown after each test became crucial to avoid unnecessary financial implications. Despite tests running in the background, continuous monitoring was necessary to verify that they were operating as expected, particularly during the development phase when resolving bugs and issues.

During the project's completion, an additional challenge emerged in comprehending and integrating two prior master's level dissertations. To fully grasp the architectural decisions and key technologies employed, a thorough exploration of these two theses was required. Analysing the source code of these projects was essential to integrate their separate distributed and threaded natures. While the information presented in their theses offered valuable insights into the system operations, the lack of professional documentation posed a challenge in comparison to other published applications. Furthermore, it was crucial to thoroughly comprehend their experiments, results, and their implications in relation to the results obtained from my own simulation.

To ensure the correctness of the system, it was necessary to run multiple tests. However, conducting extensive tests in real-time proved to be time-consuming and could potentially incur extra expenses on AWS. The presence of randomness in the system, particularly in the assignment of orders, introduced variability in the results of the same market configuration when combined with the distributed and asynchronous nature of the system. To mitigate this, the system needed to be tested multiple times, which sometimes involved running it 100 times at a time, resulting in significant time requirements. As a result, testing each stage of development took considerably longer than expected. Additionally, to minimise AWS costs and reserve funds for actual experiments, certain verification tests focusing on the threaded aspects of the system, such as the threaded traders, were conducted locally. However, testing the distributed system on a local machine was not feasible due to insufficient latency to observe differences between clients, necessitating testing on AWS.

1.5 Aims and Objectives

The aims and objects of this project is summarised as follows:

1. Integrate the work from Miles' and Rollins' theses to develop a unified fully operational market simulator that incorporates real-time latency and the execution of traders in an asynchronous manner by reimplementing their system architectures and trading algorithms into a distributed and threaded system. This is divided into the following subtasks:
   (a) Investigate the system architecture and evaluate the final implementation of the Distributed Bristol Stock Exchange (DBSE) by Bradley Miles[22].
   (b) Assess the suitability of DBSE as a foundation for the project and its compatibility with a threaded system.
   (c) Establish a secure distributed network that enables remote clients from various locations worldwide to connect to a central exchange, ensuring proper configuration and connectivity.
   (d) Conduct research and examine Rollins' work on the Threaded Bristol Stock Exchange, specifically investigating its architectural design as well as the implementation of its adaptive trading algorithms[30].
2. Enhance the client synchronisation mechanism to ensure simultaneous start without manual timing.
3. Enable the execution of multiple experiment configurations and trials in a single batch job.
4. Implement a mechanism to reset the state of the exchange after each experiment in a distributed system, triggered by the clients.
5. Create a scalable system that can handle more than 400 traders simultaneously in one market session.
6. Compare the results with those obtained in previous research to understand the combined effects of latency and trader execution time on profitability.

Chapter 2

Technical Background

2.1 Experimental Economics

Building upon the foundations laid by previous studies such on BSE, DBSE, and TBSE, this thesis employs concepts from experimental economics to study these interactions. Unlike macroeconomics, which examines the economy as a whole, this research focuses on microeconomics, which investigates the behaviour and decision-making of individual traders.

The study draws inspiration from the influential experiments conducted by Vernon Smith in 1962, as his findings hold significant relevance.

Smith's experiment drew inspiration from the work of Chamberlin in 1948. Chamberlain's experiment involved dividing volunteers into two groups: buyers and sellers. Participants were allowed to initiate negotiations with individuals from the opposing group and engage in price haggling. Interestingly, the experiment revealed that the participants in these market conditions did not reach an equilibrium price[3].

Smith utilised the experimental method developed by Chamberlin to investigate market interactions. Similarly, he divided participants into two groups, with one group acting as buyers and the other as sellers. However, Smith introduced a double auction mechanism in his experiment. This type of auction involved two sides: the buy traders and the sell traders where orders are placed publicly rather than negotiated on a one to one basis. Every trader in the experiment was assigned a limit price, which represented the lowest price at which a seller (known as an ask) was willing to sell or the highest price at which a buyer was willing to buy (known as a bit) in order to execute a trade. The auction matched buyer bids with seller asks if they were compatible. The execution of a trade influenced the market-clearing price, the price where asset demand equals asset supply. Notably, all trades conducted during the experiment were made public.

The results of Vernon's experiments demonstrated that these double auction market experiments rapidly converge to the equilibrium price, denoted as P_0.

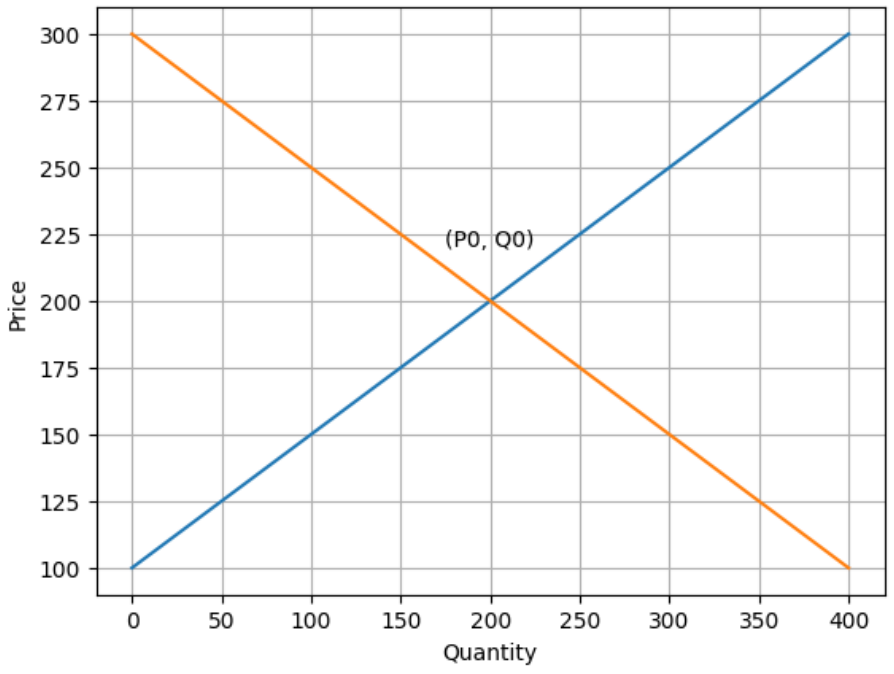

Figure 2.1: Supply and demand curve example with equilibrium point (P0, Q0).

The equilibrium price, P_0, in these experiments was determined by the interaction between the supply and demand of the traded commodity. The supply and demand curves were commonly used to model price changes in the market. The supply curve represents the quantity of the asset available, while the demand curve represents the quantity of the asset that buyers desire. The supply and demand curve together illustrate how the price of the asset changes as the quantities of supply and demand vary. The equilibrium point, denoted as (P_0, Q_0), is found at the intersection of the supply and demand curves, representing a specific quantity Q_0 of the asset. Figure 2.1 provides a visual representation of this concept.

The work conducted by Smith in this field has laid the foundations for future research, including computerised market simulations based on the same approach. Notable examples of such work include the creation of BSE, DBSE and TBSE, as well as the present study. This approach enables researchers to isolate specific variables within a market and investigate their impact on profits, which is often challenging to study in real-world markets. Additionally, it provides a means to gather empirical data to support economic analysis. However, when applying the results of experiments based on this method, it is essential to remember the assumptions made in the experiments and to acknowledge the presence of other variables that influence real-world markets.

2.2 Financial Market

The concepts discussed in this subject can be applied to analyse financial markets using the framework of a continuous double auction, similar to Smith's experiment. Trades (bids and asks) conducted through the continuous double auction mechanism are public and recorded in a Limit Order Book[33]. By applying the results from Smith's experiment, it is expected that an equilibrium price, P0, can be identified if the supply and demand curves remain unchanged. However, it is important to note that in real-life markets, the supply and demand curves can be influenced by various factors. When these curves shift, the equilibrium price also changes.

2.2.1 Limit Order Book (LOB)

The Limit Order Book (LOB) serves as a central record for all active orders related to a specific asset within the financial exchange. Traders place limit orders in the LOB, which specify the maximum bid price for buyers or the minimum ask price for sellers. Market orders, on the other hand, are immediately executed at the equilibrium price (also known as the market-clear price) and are not added to the LOB.

BID		ASK	
Quantity	Limit Price	Quantity	Limit Price
200	13.00	13.25	100
100	11.25	13.30	100
100	11.20	13.35	100
100	11.15		

Table 2.1: Example LOB corresponding to Supply and demand curve in Figure 2.2.

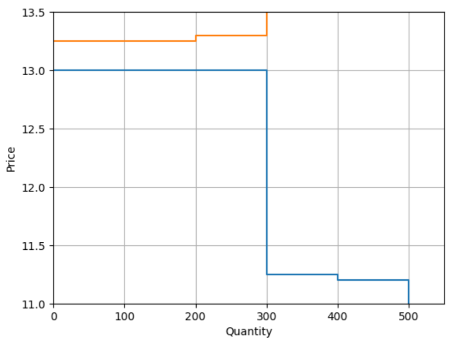

Table 2.2: Supply and demand curve corresponding to example LOB in Figure 2.1.

The LOB publicly displays the price information, with trader identities kept anonymous. When the LOB receives multiple orders that can be matched, the priority is given based on the competitiveness of the price and then the order's submission time. If no match is found, the order is added to the book.

The LOB is updated after each new order or trade, providing valuable information on bid/ask prices, market liquidity, and depth. An example LOB and its corresponding supply-demand curve can be observed in Figure 2.1 and 2.2.

The bids in the Limit Order Book (LOB) are arranged in ascending order based on price, with the best bid representing the highest buy price. On the other hand, the asks are organised in descending order, with the best ask representing the lowest sell price. This arrangement can be observed in the example shown in Figure 2.1, where the bids are displayed in the left column and the asks in the right column.

The spread, also referred to as the bid-ask spread, represents the difference between the best bid and best ask prices in a market. Crossing the spread involves submitting a bid or ask order that surpasses the current highest bid or lowest ask price, respectively. This action is typically used to execute a trade immediately. For instance, using Figure 2.1, an example of crossing the spread would be placing a new bid at 14.00 or a new ask at 13.00, surpassing the previous best bid of 13.00 or best ask of 13.25, respectively.

Three market metrics that can be derived or estimated using this information are the equilibrium price, mid-price and micro-price.

The mid-price is calculated as the average of the best bid and best ask prices.

Meanwhile the micro-price is a more accurate representation as it takes into account the quantities associated with the best bid and best ask prices, weighting them accordingly. It is frequently used in algorithmic trading to assess the market.

2.3 Bristol Stock Exchange (BSE)

This simulation, developed by Cliff[7] in 2012, is one of the first and only open-source applications available. Initially designed as a teaching tool, it has gained popularity in the research community. The simulation is implemented in Python and aimed to be accessible to beginners.

The architecture of the simulation revolves around several key components. Firstly, there is the exchange, which facilitates market interactions. The Limit Order Book (LOB) is responsible for recording and managing all outstanding orders for a specific asset. The market is populated by various classes of trader objects, each representing different trading strategies or behaviours. A customer order distribution mechanism is in place to distribute new orders among the traders. The simulation operates within a market session process loop, allowing for the repetition and analysis of multiple trading sessions.

2.3.1 Simplifying Assumptions

The simulation makes certain simplifying assumptions, many of which stem from the use of a time slice approach to evaluate trader executions. For instance, in a simulation with N traders, each trader is allocated 1/N seconds of virtual simulation time to execute their trading actions. At the start of each round, the N traders are randomly shuffled, and one trader is selected to respond and place an order in the Limit Order Book (LOB). If a trade is matched, it is immediately executed. The process continues with the selection of the next trader until all traders have completed their actions, and then the simulation time advances.

It's important to note that this virtual simulation time does not correspond to real-time and assumes zero latency. Furthermore, since all traders are able to finish their execution within 1/N virtual simulation time, the simulation assumes negligible trader execution time and forces traders to be evaluated one after another, rather than asynchronously as in real-world markets. Additionally, the simulation focuses on the trading of a single symbol and restricts traders to having only one trade on the LOB at a time.

It's crucial to keep these limitations in mind when interpreting the simulation results and their application to actual market conditions.

2.3.2 Exchange

The exchange holds the Limit Order Book (LOB) and executes the matching engine logic. It listens for orders from traders through the exchange object's `process_order` function. The order is published to the LOB and checked to see if the trader already has an existing order in the LOB. If an existing order is found, the previous order is deleted. The system then checks if the new order can be matched with either the best bid or best ask price in the LOB. If the spread is crossed, resulting in a trade, the LOB is updated, and the exchange provides a simplified execution report (called the `transaction_report` in the application). This execution report is also added to the LOB tape. Additionally, the anonymised trade update is sent to other market participants by using the `publish_lob` function to send an anonymised version of the updated LOB to traders.

2.3.3 Limit Order Book (LOB)

The exchange comprises two versions of the Limit Order Book (LOB): one version includes trader IDs, enabling the exchange to notify traders when their trades have been executed, while the other version is anonymised and used to update market participants. Additionally, the exchange includes a tape that maintains a list of the last 10,000 trade outcomes, which can be either trade cancellations or successful trades.

The general operations of a Limit Order Book (LOB), which are not specific to any particular implementation, are discussed in Section 2.2.1.

2.3.4 Traders

Trader python objects in BSE can be understood by examining how they are evaluated in the simulation. These trader objects utilise a base Trader class, which serves as the base for all runnable trader algorithms, where all child trader types inherit from this class.

The Trader class implements several key functions: `add_order`, `del_order`, `bookkeep`, `respond`, and `getorder`. These functions play vital roles in tracking live orders, recording trade outcomes, and updating internal values of the traders.

The `add_order`, `del_order`, and `bookkeep` functions are consistent across all trader types. They are responsible for managing the trader's orders, recording the execution of trades, and maintaining relevant data.

The `respond` function is called after each market update to update the trader's internal values, which are used to calculate new limit order prices. While simple traders may not perform any specific actions in this function, it is implemented differently by each trader type to reflect their respective strategies.

The `getorder` function determines whether the trader is able to place a new order. If so, it uses its internally calculated values from the `respond` function to calculate a new limit order price and return the executable order.

Further details on the trader algorithms themselves will be provided in the below Section.

2.3.5 Order Schedule

The user-defined order schedule allows for the specification of supply and demand in the market through the use of a supply schedule and demand schedule. By defining these schedules, the algorithm determines the supply-demand curve that characterises the market.

In addition to the supply and demand schedules, the algorithm allows for the specification of minimum and maximum prices within a given time-frame. This flexibility enables the incorporation of various changes to the supply-demand curve, such as introducing sudden market shifts or incorporating an offset function to introduce variations in limit prices over time.

To control the timing of order issues, the algorithm uses an `order_interval` variable, which specifies the time interval between issuing new orders to the trader. The step-mode parameter is employed to generate prices from the specified price range. Furthermore, a time-mode parameter is used to determine the variation in the arrival times of new orders to the trader, such as whether all traders receive new orders simultaneously at the start of the `order_interval` or if the orders are spread out throughout this period.

2.3.6 Market Session

The market session is operated by a time loop that runs throughout the entire simulation. This time loop employs a time-sliced approach, which is responsible for the zero latency assumption and the negligible trader execution time assumption discussed in Section 2.3.1.

During the time loop, customer orders are distributed to the traders. The order schedule is utilised to determine the price and timing of new order issues, as explained in Section 2.3.5.

At the end of each market session, a log is created containing a list of transactions and the profits earned by the traders.

2.4 Trading Algorithms

To gain insights into the performance of different trader types across various simulations, it is essential to understand the fundamentals of each trader algorithm. Below is a summary of the trader algorithms used in BSE, DBSE, and TBSE:

2.4.1 Giveaway (GVWY)

GWVY is a simple trading agent that operates by placing limit orders at specified prices, with the expectation of not generating any profit. This agent does not adapt to market changes and does not rely on real-time updates from the exchange. Due to its fast execution without waiting for market updates, it can occasionally generate profits in certain market conditions with rapidly changing prices.

2.4.2 Zero Intelligence Constrained (ZIC)

This trading agent randomly generates a price within the range of the least competitive price to its maximum limit order price. It exhibits slightly more intelligence than GVWY as it aims to make some profit, albeit randomly, without adapting to market conditions. Similar to GVWY, it does not rely on market updates, making it a fast execution strategy.

2.4.3 Shaver (SHVR)

This trading strategy selects the most competitive price on the limit order book (LOB) and improves it by one unit. However, in order to compete with the best bid or ask prices, it does require waiting for market updates.

2.4.4 Sniper (SNPR)

This trading strategy functions similarly to SHVR. As the name implies, it waits to execute trades near the end of a market session then tries to snipe trades close to the end of the market session. It operates by surpassing the most competitive price on the limit order book (LOB) by an increasingly larger margin as the session nears its conclusion.

2.4.5 Zero Intelligence Plus (ZIP)

Developed by Cliff in 1997[6], ZIP stands as one of the first adaptive trading algorithms that was able to respond to the current market conditions. It uses a modified version of the Widrow-Hoff rule to update a margin value. A concise high-level summary of this algorithm will be presented below.

In the ZIP algorithm, a desired profit margin variable is maintained to determine the trader's order price. This margin is adjusted based on recent market activity.

$$f_b(p) = \frac{ABL(p) + AL(p)}{ABL(p) + AL(p) + UBG(p)}$$

Figure 2.2: GD Buyer's belief function

Where: $ABL(p)$ is the number of Accepted Bids Less than p, $AL(p)$ is the total number of Asks Less than p, and $UBG(p)$ is the number of Unaccepted Bids Greater than p

$$f_s(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)}$$

Figure 2.3: GD Seller's belief function

Where: $AAG(p)$ is the number of Accepted Asks Greater than p, $BG(p)$ is the total number of Bids Greater than p, and $UAL(p)$ is the number of Unaccepted Asks Less than p

$$p^* = arg\ max\ f_b(p)(l - p)$$

Figure 2.4: GD Buyer's expected profit function

$$p^* = arg\ max\ f_s(p)(p - l)$$

Figure 2.5: GD Sellers's expected profit function

1. If the trader's latest order is accepted, it adjusts the margin to move its order price closer to the executed trade price.
2. However, if the trader's order is not executed and its order price is worse than the unexecuted order, the margin value is reduced.

This margin value is updated using the Widrow-Hoff rule allowing the trader to dynamically modify its profit margin in response to fluctuations in the limit order book (LOB).

In order to avoid keeping the trader's order price constant when the last LOB order price is similar to the current price, two random variables Ri and Ai are introduced. This approach is used to prevent the trader from becoming stagnant and encourages it to actively seek better prices. Ri modifies the target price in relation to the last order on the LOB, while Ai introduces a slight absolute adjustment to the order price.

2.4.6 GD eXtended (GDX)

GD, developed by Gjerstad and Dickhaut in 1998[12], served as the foundation for further modifications. One such modification is MGD, introduced by Tesauro and Das in 2001[38]. Although not relevant to this paper, the original details of MGD can be found in Tesauro and Das' research[38] or in Rollin's paper[30]. GD was then developed into GDX by Tesauro and Bredin in 2002[37].

Gjerstad-Dickhaut (GD)

GD utilizes a "belief function" described in Equations 2.2, 2.3.

Using this belief function, the GD algorithm aims to strike a balance between setting prices that maximise potential profit and prices that are more likely to be accepted. This is achieved by utilising an expected profit value calculated through specific equations (refer to Equations 2.4, 2.5. This expected profit value need to be recalculated every time there is a new event or development in the market.

Note: Equations 2.2, 2.3, 2.4, 2.5 are sourced from Rollins' paper[30].

Extending into GDX

Dynamic programming techniques were incorporated into the GD algorithm to enhance its capabilities, resulting in the creation of the GDX algorithm.

Uses the following variables to calculate a table of predicted values:

- Trading opportunities $N = T/K$ where T is remaining Time, K represents the estimated duration in seconds until the trader can place the next order
- Belief function $f(p, \pm\overrightarrow{q}, t)$ where the inputs represent a particular order detail including the quantity of assets q with positive values for asks and negative values for bids, the price p, and the remaining time t in the trading period and outputs the resulting probability of trade execution

- Predicted value $V(x, n)$ where x is the trader's internal state and N is the remaining bidding opportunities
  - This is used to generate a table of predicted values as an output. It's important to note that the LOB implementation in BSE/DBSE/TBSE only allows a trader to have one active trade, so in this case, the internal state x is equal to the assets being traded M. Therefore, the algorithm calculates $V(m, n)$.

The traders then uses dynamic programming to calculate the table of $V(m, n)$ values by:

1. Setting $V(m, 0) = 0$ for all values of m
2. Setting $V(0, n) = 0$ for all values of n
3. Then $V(m, n-1)$ is used to calculate $V(m, n)$ with algorithm 2.1

```
for $n = 1$ upto $N$ do
    for $m = 1$ upto $M$ do
        $V(m, n) = max(f(p, t_n))\,[s(p) + \gamma V(m-1, n-1)] + (1 - f(p, t_n))\gamma V(m, n-1)$
    end
end
```

Where: γ is the discount paramter, p is the proposed price, $s(p)$ resulting profit if asset is traded at p.

Algorithm 2.1: Expected value computation for GDX[30].

After calculating the table of V(m, n), the trader selects the optimal price for the order based on the calculation shown in equation 2.1. This optimal price is determined anew each time GDX intends to place an order.

$$p^*(T) = arg\ max(f, (p, T)\,[s(p) + \gamma V(M-1, N-1)] + (1 - f(p, T)\gamma V(M, N-1)) \tag{2.1}$$

Note: Equation 2.1 and Algorithm 2.1 are sourced from Rollins' paper[30].

2.4.7 Adaptive-Aggressive (AA)

The AA algorithm incorporates an aggressiveness value, denoted as r which influences how the trader calculates the order price relative to the equilibrium price p^*. The value of r falls within the range $[-1, 1]$.

There are three distinct cases to consider:

1. When $r < 0$, the trader is defined as aggressive. In this case, the trader submits orders with prices better than p^*, increasing the likelihood of execution but resulting in lower profit potential.
2. When $r = 0$, the trader is defined as active. The order prices align with p^*.
3. When $r > 0$, the trader is defined as passive. The order prices are less competitive than p^*, making execution less likely but offering higher profit potential.

The AA algorithm adapts the aggression value r based on recent market conditions.

The algorithm can be outlined in the following steps:

1. Estimate equilibrium price p^*
2. Building the aggressiveness model
3. Adaptive layer
   (a) Short term learning for r
   (b) Long term learning for θ
4. Bidding layer

1. Estimate equilibrium price p^* The algorithm calculates the average transaction prices over the N most recent trades, starting from the latest transaction T and going back to $T-N$. It assigns a weighting to each transaction with a higher weight given to more recent transactions. This weighting, represented by ρ, allows the algorithm to converge quicker towards the equilibrium price p*, assuming that prices are always moving towards this equilibrium price. This calculation is shown in Equation 2.2.

$$p^* = \frac{\sum_{1=T-N}^{T}(w_i \cdot p_i)}{N} \quad where: \quad \sum_{1=T-N}^{T} w_i = 1, w_{i-1} = \rho w_i \tag{2.2}$$

2. Building the aggressiveness model The aggressiveness model utilises the parameter r to generate a target price, denoted as T. Traders are divided into two groups based on their limit prices compared to the equilibrium price p^*.

The first group is called intra-marginal traders, consisting of buyers/sellers with limit prices that are more competitive than p^*. These traders are expected to execute trades and actively participate in the market. The calculation of T for intra-marginal traders is defined in Equations 2.4.7. 2.4.7

The second group is referred to as extra-marginal traders, comprising buyers/sellers with limit prices that are less competitive than p^*. These traders are not expected to execute trades on their own but rely on other traders posting orders that cross the equilibrium price p^*. The calculation of T for extra-marginal traders is defined in Equations 2.4.7, 2.4.7

For an intra-marginal buyer,

$$\tau = \begin{cases} p^*(1 - \frac{e^{-r\underline{\theta}}-1}{e^{\underline{\theta}}-1}), & \text{if } r \in (-1,0) \\ p^* + (\ell_i - p^*)(\frac{e^{r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (1,0) \end{cases}$$

For an intra-marginal seller,

$$\tau = \begin{cases} p^* + (MAX - p^*)(\frac{e^{-r\underline{\theta}}-1}{e^{\underline{\theta}}-1}), & \text{if } r \in (-1,0) \\ c_j + (p^* - c_j)(\frac{e^{r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (1,0) \end{cases}$$

For an extra-marginal buyer,

$$\tau = \begin{cases} \ell_i(1 - \frac{e^{-r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (-1,0) \\ \ell_i, & \text{if } r \in (1,0) \end{cases}$$

For an extra-marginal seller,

$$\tau = \begin{cases} c_j + (MAX - c_j)(\frac{e^{-r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (-1,0) \\ c_j, & \text{if } r \in (1,0) \end{cases}$$

Where l is a buy's limit price and c is a seller's limit price. Meanwhile the calculations for θ and r are defined in the Adaptive layer.

3.a Adaptive layer: Short term learning The algorithm utilises the Widrow-Hoff equation to adjust the aggressiveness parameter, denoted as r, towards a desired aggressiveness called $\delta(t)$. The adjustment process differs for buyers and sellers.

For buyers, the desired aggressiveness is determined as the minimum value between their limit price and a price slightly better than the best bid in the market.

For sellers, the desired aggressiveness is calculated as the maximum value between their limit price and a price slightly lower than the best ask in the market.

By adjusting the aggressiveness parameter based on the desired price levels, the algorithm aims to strike a balance between competitiveness and potential execution in order to optimise trading strategies for both buyers and sellers.

The logic for adjusting the value of parameter R is as follows:

1. **For a buyer:**
   (a) Check if a transaction occurred at a certain price, q.
   (b) If a transaction did occur and the current value of r is greater than q, then the buyer needs to be less aggressive.
   (c) Otherwise, the buyer can be more aggressive.
   (d) If no transaction occurred but a bid at price b was submitted, check if the bid was not accepted. In this case, the buyer needs to be more aggressive.

2. **For a seller:**
   (a) Check if a transaction occurred at a certain price, q.
   (b) If a transaction did occur and the current value of r is less than q, then the seller needs to be less aggressive.
   (c) Otherwise, the seller can be more aggressive.
   (d) If no transaction occurred but an ask at price a was submitted, check if the ask was not accepted. In this case, the seller needs to be more aggressive.

By considering the transaction history and the acceptance of bids or asks, the algorithm adjusts the aggressiveness of buyers and sellers to optimise their trading strategies based on market conditions.

3.b Adaptive layer: Long term learning The aggressiveness curve is determined by the parameter θ, which helps fit the trading strategy to market conditions. The values of θ depend on an estimate of Smith's α, which measures market volatility.

The adaptation of θ involves the use of the learning rate β_2 and the function $\theta^*(\alpha)$, which calculates the optimum θ values. The optimum θ values are defined as the values that maximise the expected profit given the estimated market volatility.

The range $[\theta_{min}, \theta_{max}]$ represents the allowable values for θ to be updated. α_{max} corresponds to the maximum observed value of α in the market, while αmin represents the minimum value of α. The parameter γ influences the shape of the function used to update θ.

4. Bidding layer This layer is responsible for determining whether to place a bid
ask and at what price, based on the target price T calculated using the values of r and θ.

2.5 Distributed Bristol Stock Exchange (DBSE)

The Distributed Bristol Stock Exchange (DBSE) was developed by Miles in 2019[22]. It aimed to incorporate real-time latency into the simulation environment. This tool was specifically designed to study latency arbitrage and race-to-market phenomena observed in real-world globally distributed financial markets.

The architecture of the simulation tool includes the utilisation of a Financial Information Exchange (FIX) Server and Clients, as well as User Datagram Protocol (UDP) market data publishers. These features mirror the real technologies employed by actual exchanges, adding a level of realism to the simulation. The tool can be deployed through the AWS cloud infrastructure, enhancing its accessibility and scalability.

Miles conducted experiments involving remote trader agents located across the world, each experiencing varying levels of latency. By simulating these different latency scenarios, the tool provided insights into the impact of latency on trading activities in a distributed market environment.

In order to synthesise Miles' work with the Threaded Bristol Stock Exchange, it is essential to understand the underlying architecture and technologies employed. Thus, this section provides a summary of the key aspects presented in Miles' paper[22].

Additionally, a summary of the DBSE (Distributed Binary Stock Exchange) experiment and its results[22] will be provided, as this project aims to compare results on the DTBSE to that of DBSE.

2.5.1 Architecture

Real World Architectural Design: JX Framework

As explained by Miles' in his thesis[22], the design of DBSE was influenced by the architecture of Jane Street Exchange (JX), created by Jane Street which is an established liquidity provider and global quantitative trading firm specialising in a wide range of financial assets, conducting trading activities across a vast network of over 200 trading platforms in 45 countries around the globe[35].

The design and model of Jane Street's own exchange (JX)[24] which, although not classified as an alternative trading system, operates as a proprietary platform that enables clients to access their private liquidity[36]. JX's architecture is inspired by traditional exchanges and it is recognized as one of the top three single-dealer platforms in the United States[35]. Due to its suitability for testing trading algorithms, JX became the architectural inspiration for DBSE.

The following provides a brief summary of the key architectural features observed in Miles' case study of JX, which had a significant influence on the design of DBSE[22]. This summary is essential to understand how these features were adapted in DTBSE, as DTBSE is built upon the design principles of DBSE.

Within the network infrastructure, there is a private subnet that comprises several components including a matching engine, cancel and auction fairies, and a retransmitter. Meanwhile, the public subnet consists of a trader reporter, drop port, client ports, and a market publisher. Communication between these components is facilitated through UDP multicast, which ensures fairness by sending messages to all recipients simultaneously, although there is no guarantee of message delivery.

The matching engine, located on a single machine, houses the limit order book (LOB), performs order matching, and sends updates to internal services within the public subnet. In close proximity to the matching engine. The cancel fairy stores cancel order messages and forwards them to the matching engine at the appropriate time. Similarly, the auction fairy aggregates orders with the same price. To mitigate the risk of dropped UDP messages, the retransmitter retransmits data as needed.

Client ports serve as connection points for clients to connect to the exchange. The drop port facilitates communication with clearing firms, which are responsible for transferring funds after a trade execution. Execution reports are sent by the exchange through this port. The trade reporter publishes anonymized data to external trade reporting firms, while the market data publisher serves a similar purpose but sends data to both internal services and clients. High-frequency trading firms often opt for a direct connection to the market data publisher, bypassing the reporting firms, by paying a substantial fee.

Financial Information eXchange (FIX) Protocol

The FIX protocol is widely used as a trading communications standard by thousands of trading firms. It is a standardised protocol maintained and updated by the FIX trading community to adapt to the evolving products in the financial world. The protocol is organised into different versions known as the "FIX Family of Standards." For DBSE, the implementation was based on FIX.5.0SP2 and FIXT1.1, which were the most recent standards available at the time. Notably, the application level standard (which defines financial messages) and the session level standard (which governs message communication) were decoupled into FIX.5 and FIXT1.1, respectively. This separation allows for upgrading one standard without impacting the other[22].

QuickFIX

The Python QuixFIX module is a free open-source FIX protocol application, which was used to implement DBSE. It provides various methods for managing FIX sessions, including server-side behaviour (acceptors) and client-side behaviour (initiators) upon connecting or disconnecting from a counterparty, as well as sending and receiving messages.

To configure sessions, a configuration file is used, specifying the FIX/FIXT versions, connection type (acceptor or initiator), and Sender/Target IDs.

Real Time

DBSE introduced a significant change to BSE by transitioning from virtual simulation time to real-time execution. This shift was crucial for studying real-time latency and its effects. One of the key modifications involved implementing real-time scheduling of new customer orders at regular intervals. To achieve this, a background scheduler was utilised to trigger the execution of orders at predetermined time intervals. This adjustment allowed for a more realistic simulation of market dynamics and the evaluation of real-time trading strategies.

Market Data

To ensure traders can respond to changes in the limit order book (LOB), it was necessary to publish market data to clients. While JX utilises UDP multicasting, which allows for simultaneous delivery of a single message to multiple receivers without overloading the network, AWS infrastructure limitations required the use of unicast in DBSE. UDP Unicast involves sending each message separately to each client, although they are not delivered simultaneously. Nonetheless, unicast is faster than TCP since it does not require acknowledgments. In DBSE, a UDP sender and receiver are implemented for this purpose.

Miles' analysis of the DTBSE latency between the exchange (LDN) and each of the four DBSE clients is as follows: LDN 1 (0.4 ms) < LDN 2 (0.5 ms) < OH (43.7 ms) < SYD (134.9 ms)[22].

Traders

Since Miles' thesis focuses on the Race to Arms experiment, it therefore only needed to implement simple GVWY, SHVR, ZIC, and SNPR agents. These trader types are considered sufficient to achieve the main objective of the experiment in DBSE.

2.5.2 AWS Cloud Deployment

AWS offers a wide range of scalable services, with particular interest in storage and compute power for both DBSE and DTBSE. At the time of DBSE's development, Amazon Cloud spanned 21 geographical regions[22], and it has since expanded to 31 regions[1], enabling the distribution of clients across the globe.

In Miles' design, the networking environment is controlled using Virtual Private Clouds (VPCs), which manage IP address ranges, subnets, and routing tables. Three separate VPCs were configured in the London, Sydney, and Ohio regions.

DBSE clients were deployed on t2.micro instances, which are part of the free tier, while the exchange ran on a more powerful t2.medium instance to handle incoming orders from multiple clients.

To ensure that clients were synchronised to the same time zone despite being in different regions, Miles utilised Amazon's Time Sync service. Additionally, the market session start was synchronised by delaying the session to begin at the start of the next minute.

2.5.3 Race to Market: Experiment

To explore the Race-to-Market phenomenon, which refers to the competitive advantage gained by traders who respond to market changes faster than others, Miles conducted an experiment using the real-time latency of DBSE. In the DBSE experiment, four clients were involved, two located in London, one in Sydney, and one in Ohio. A total of 160 trading agents participated, with an equal split of 50% buyers and 50% sellers, and an equal distribution of GVWY, SHVR, ZIC, and SNPR agents among all four clients. All agents had the same order schedule configuration, which is given in Listing 2.1.

One notable feature of the experiment was the introduction of a sudden change in the price range, creating a market shock that simulated real-world scenarios where prices fluctuate due to external factors. Miles conducted the experiment ten times, allowing for reliable analysis and evaluation of the results.

The results of the experiment showed consistent rankings for each algorithm within each client, with SNPR performing the best, followed by GVWY, SHVR, and ZIC. However, the total profits varied across clients due to the impact of latency. The results are as follows: LDN 2 achieved the highest profit percentage at 25.72%, followed by LDN 1 at 25.19%, OH at 24.99%, and SYD at 24.10%[22]. The variation in profits correlated to latency experienced by the clients, proving that latency was a limiting factor.

Miles noted that if latency had no effect on trader profits, all clients would have achieved a profit percentage of 25%. The correlation between profits and client latency demonstrated that not only was the distributed system functioning as expected but also that latency had a noticeable effect on traders.

2.6 Threaded Bristol Stock Exchange (TBSE)

Rollins developed the Threaded Bristol Stock Exchange in 2020 [30] to investigate the impact of algorithm execution times on profitability.

The project utilised a threaded architecture, aiming to replicate real-world market conditions more accurately. Pairwise experiments were conducted, systematically varying the ratio of traders of type A to B from (1:19) to (19:1). The section summarises aspects of Rollins' paper and provides a comprehensive overview of its essential components, as a brief overview of its architecture is needed for adapting them to the distributed architecture. Furthermore, to facilitate a meaningful comparison between DTBSE and TBSE, a concise summary of the TBSE experiment and its relevant results will be presented.

2.6.1 Architecture

In TBSE both the exchange itself and each individual trader are assigned their own dedicated thread. The primary Python thread is responsible for calculating customer orders and distributing them to the respective traders. To facilitate communication and data exchange between the different components, Python's thread-safe queues are utilised. These queues serve as the means to send and receive market data, orders, and execution reports between traders and the exchange.

Exchange

The system incorporates an order queue where all traders enqueue their orders, which are then read and processed by the limit order book (LOB). Upon trade execution, an execution report is generated and enqueued into the respective trader's transaction queue. It's important to note that this process is specifically applicable to the two traders involved in the executed trade and does not replace the LOB tape present in the original BSE. The LOB tape, as described in Section 2.3.3, is utilised by certain trading algorithms.

Multi-Threading

Python provides both multithreading and multiprocessing capabilities. While multithreading gives the illusion of simultaneous execution via rapid context switching, multiprocessing restricts the number of traders to the number of available CPUs. Taking into account these considerations, DTBSE aligns these design choices (details in Section 3.1.2) found in TBSE to utilise multithreading over multiprocessing.

Real-Virtual Time Mechanism

On further investigation into TBSE's execution it was found to use a virtual to real time mechanism wherein the user can specify the simulation time and real time for the experiments to run. This works in conjunction with the order interval to speed up the passing of time felt by the simulation to run in the specified real time. For example if an experiment were to run for 60 seconds real time for 120 seconds of simulation time with 40 second order intervals, the order interval would be scaled down by a factor of (sim-time / real-time = 120/60 = 2) to become 20 seconds. Therefore the simulation would still undergo three rounds of new customer limit orders. In theory this should produce the same result as if running the simulation for 120 seconds with 40 second order intervals as after each trader issues an order the market will remain idle until the next custom limit order. However if the simulation to real time ratio is too high then trader threads could be issued new orders before executing their previous order. Therefore in order to recreate experiments the real time execution value must be specified as well as the simulation time and order interval. This will be important to note in TBSE/Pairwise Variable Ratio experiment comparisons to DTBSE as mentioned in Section 4.1.2.

2.6.2 Experiments: Pairwise Variable Ratio

Each experiment conducted in the project consisted of 20 buyers and 20 sellers, ensuring a balanced representation. The buyer and seller schedules were symmetric, maintaining a consistent ratio (A:B) of two algorithms on both sides of the market where the ratio of Algorithm A to Algorithm B varied from (1:19) to (19:1). This design enabled the analysis of algorithm performance as the A to B ratio was altered. In total, Rollins tested 15 pairs of algorithms across 19 different ratios, resulting in 285 distinct experiments.

For each of the 285 unique experiments, 10 random supply-demand curves were generated. These configurations were then used to run 100 market sessions. The sessions employed a periodic time mode and a fixed step mode, lasting for 600 seconds with 30-second order intervals. Two sets of tests were conducted: one with a Variable offset function applied to the price range and one without. The shape of the function can be seen in Figure 2.6, lifted from Rollins' paper for reference.

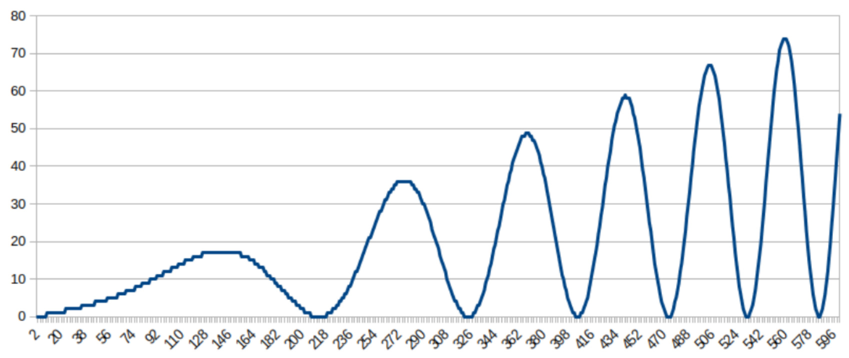

Figure 2.6: Variable Offset used in Rollins' Variable Offset Experiments[30].

Equal-Availability Ratio (EAR)

One key aspect of Rollins' experiments was the use of an Equal-Availability Ratio (EAR) to determine the winners in each pairwise test[30]. Due to the presence of different ratios among traders, it is not sufficient to solely compare the average profits of each group. Even if we consider two hypothetical perfect traders operating at the equilibrium price, the varying ratios and assignment of limit order prices mean that the trader with fewer participants of their type will have a lower average profit, despite both traders executing flawless trades.

Rollins best explains this phenomenon with the following example which has been summarised from his thesis[30]:

> Consider a market experiment with 5 traders with a ratio of (A:B) as (1:4) and 5 limit prices of 50, 100, 150, 200, 250, with an equilibrium price $P_0 = 150$. Since this thought experiment uses hypothetical perfect traders, only those assigned a limit price above $P_0 = 150$ can execute a trade. Running 1000 tests, there will be 5 different cases, each occurring 200 times.
>
> 1. In the first case, if trader A is randomly assigned a limit price of 250, A's average profit is 100, while trader B's average profit is 12.5.
> 2. In the second case, if trader A is randomly assigned a limit price of 200, its average profit is 50, surpassing trader B's average profit of 100/4.
> 3. In the remaining three cases, trader A makes no profit, while trader B is consistently assigned limit prices of 250 and 200, resulting in an average profit of 37.5.
>
> Consequently, trader B emerges as the winner in 600 instances, while trader A wins in 400 instances.

Hence, Rollins defined an Equal Availability Ratio for the pairwise ratio experiments in the following manner:

$$EAR = \frac{Win_A - Win_B}{Trials} \tag{2.3}$$

The Equal Availability Ratio (EAR) is calculated to determine the required raw win ratio for one trader to surpass the other trader type. For each ratio ranging from (1:N) to (N:1), the EAR is computed, allowing for a comparison between the raw win rate and the EAR line. If the raw win value surpasses the line, it indicates that trader A is the winner, while trader B takes the victory if the value is below the line. This consideration of the EAR was crucial in Rollins' pairwise experiments[30] and was also employed when replicating these experiments in DTBSE.

Results

Although Rollins conducted a significant number of experiments, only a subset of them were replicated in DTBSE. Hence, this section will focus solely on summarising the results and conclusions of relevant experiments that were recreated in the DTBSE framework. An explanation detailing the rationale behind selecting these particular experiments is provided in Section 4.1.2.

		BSE		TBSE	
Trader A	Trader B	A Wins	B Wins	A Wins	B Wins
AA	ZIC	14146	4854	16276	2724
ZIP	ZIC	8753	10247	17025	1975
SHVR	ZIP	15875	3125	8717	10283
GVWY	SHVR	7661	11339	8430	10570

Table 2.3: Outcomes of Relevant Pairwise Variable Ratio Experiments[30].

It is important to note that the figures from these experiments conducted in TBSE are provided alongside the results of DTBSE and DBSE pairwise experiments in Section4.1.2 for convenient comparison.

AA vs ZIC AA outperforms ZIC across all ratios in both BSE and TBSE, which is to be expected given that AA is adaptive while ZIC generates prices randomly.

SHVR vs ZIP In the results of BSE, SHVR outperforms ZIC by a significant margin in both non offset and variable offset experiments. Rollins notes that this observation is consistent with other pairwise experiments conducted in his paper, indicating that SHVR is capable of tracking the equilibrium price by competing with other traders' orders. However, in TBSE, the results are quite different, as ZIP is able to narrowly outperform SHVR with the variable offset but not in the non-offset experiment. This discrepancy makes the performance of SHVR in TBSE an outlier compared to other experiments involving SHVR.

ZIC vs ZIP The comparison between offset and non-offset variable results yielded interesting findings. In both BSE non-offset and BSE offset experiments, a similar pattern was observed, with ZIC achieving more wins. Rollins suggests that ZIP may not be as effective in generating profit when the equilibrium price is fluctuating[30]. Surprisingly, in the TBSE offset experiments, ZIP emerged as the clear winner by a significant margin when it lost by a significant margin in the non-offset version. Rollins notes that this outcome was unexpected, considering the assumption that ZIP would be adversely affected by its slower nature compared to ZIC in the offset scenario and the paper suggests possible further investigation into the cause of these unexpected results.

GVWY vs SHVR In both BSE and TBSE, SHVR emerges as the winner. This outcome is expected since both BSE and TBSE utilise simple traders that do not adapt to the market. Therefore, there is no significant difference between the results obtained from BSE and TBSE experiments in this regard.

```
{
    "duration": 180,
    "traders": {
        "buyers": {
            "GVWY": 10,
            "ZIC": 10,
            "SHVR": 10,
            "SNPR": 10
        },
        "sellers": buyers
    },
    "order_schedule": {
        "demand": [
            {
                "from": 0,
                "to": 60,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            },
            {
                "from": 60,
                "to": 120,
                "ranges": [{"min": 150.0, "max": 250.0}],
                "stepmode": "fixed"
            },
            {
                "from": 120,
                "to": 180,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            }
        ],
        "supply": demand
        ],
        "interval": 30,
        "timemode": "drip-poisson"
    }
}
```

Listing 2.1: Simulation Configuration for Race to Market Experiment.

Chapter 3

Project Execution

3.1 Architecture

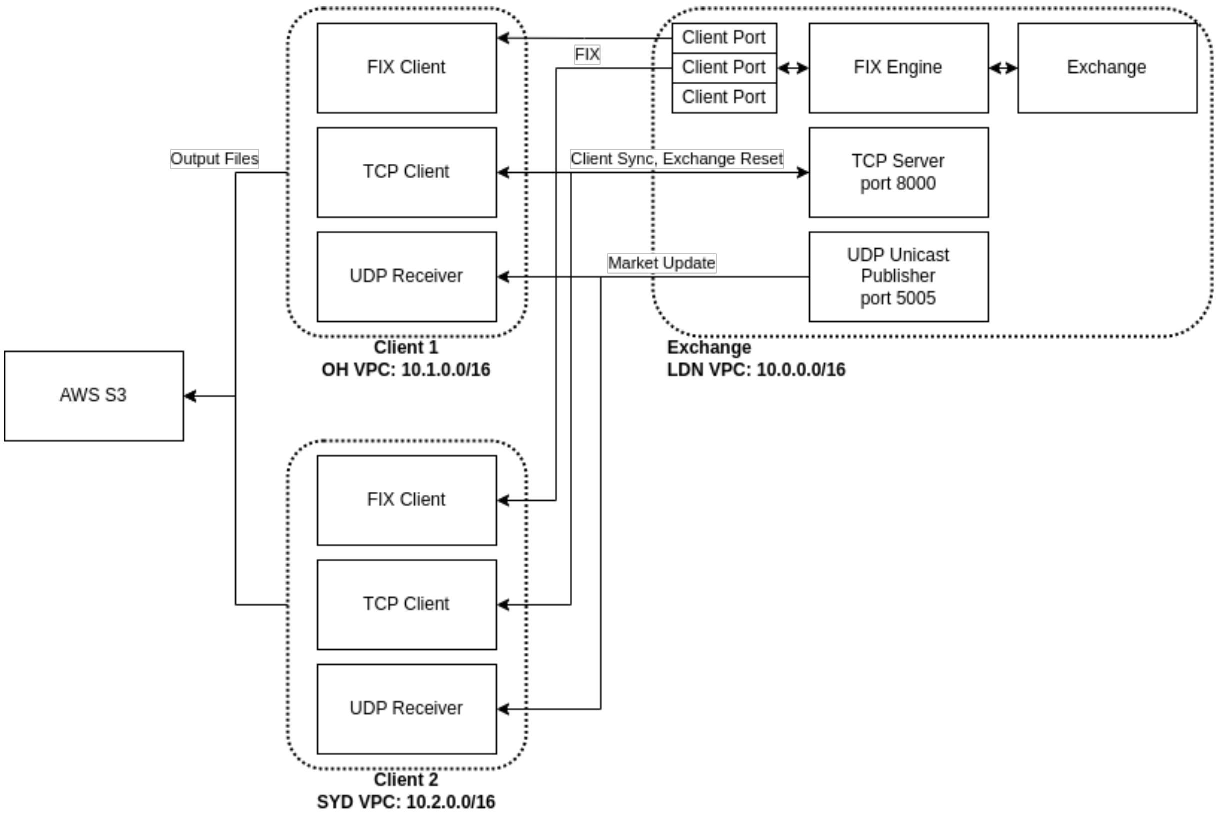

Figure 3.1: Illustration of DTBSE's Distributed Architecture.

The chosen design approach centred around using DBSE as the foundation and integrating selected features from TBSE into the distributed architecture. A full overview of the Distributed and Threaded BSE (DTBSE) architecture can be seen in Figure 3.1.

3.1.1 Real Time Execution

An initial design choice was made to exclude TBSE's real to virtual time conversion mechanism. Although this feature could enhance the efficiency of run executions by eliminating idle time after traders execute their orders, it would conflict with the primary objective of DTBSE, which is to explore the impact of algorithm time complexity and network latency on trader profits. Consequently, simulation configurations had to be carefully considered to address the potential concern of extended periods of trader inactivity

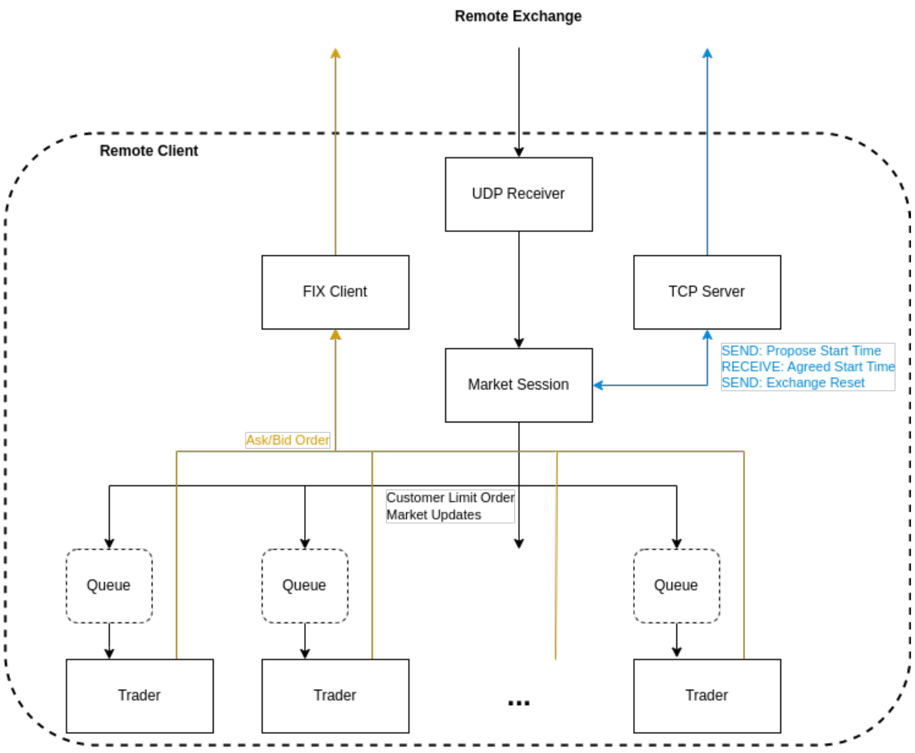

Figure 3.2: Illustration of DTBSE's Internal Client Architecture.

on large order intervals.

3.1.2 Trader Threads and Queues

This entailed incorporating thread-based functionality for each trader within the DBSE framework. However, since orders were sent using a FIX client and market updates were obtained through a UDP receiver, it was necessary to introduce a mechanism to enable these operations within each thread. Within the DBSE system, the initiation involves scheduling new customer limit orders at the outset, followed by a random shuffling of traders. Subsequently, the client sends these new customer limit orders to the traders one by one, who then generate their bid/ask orders to be transmitted to the exchange using the FIX client. Every second, the exchange's UDP sender transmits the real-time status of the limit order book to each remote client. After shuffling the traders, the remote clients proceed to call the `get_order` function for each trader successively, allowing them to place an order through the FIX client if there are any customer limit orders available. The process continues until all traders have acknowledged the current market state, after which the next market state update is received through UDP unicast.

Within DTBSE, individual traders are assigned separate threads to operate independently, yet they still need to receive all market updates. To address this, a thread-safe queue was implemented for each trader, taking inspiration from TBSE's transaction queue, enabling the UDP receiver to send each market state update via each of the traders' queues. Likewise, when a fresh customer limit order is ready to be dispatched to the traders, it is enqueued to the respective trader's queue according to the order schedule. The execution of each trader and order placement via FIX client is performed concurrently. The overall architecture can be seen in Figure 3.2.

Separate FIX clients for each trader thread were unnecessary since the `sendToTarget` function of the FIX client is inherently thread safe (although this is not explicitly mentioned in the documentation, it can be inferred from the source code and the use of the `SWIG_PYTHON_THREAD_BEGIN_ALLOW` and `SWIG_PYTHON_THREAD_BEGIN_DISALLOW` macro functions before and after the `sendToTarget` function implementation). Moreover, each new FIX client would require the registration of its own unique socket port and client ID in the exchange's FIX engine configuration which would impede the scalability of virtual computers (nodes) that run multiple threaded traders.

From this point forward, the term "simple traders" will refer to the GVWY, SHVR, SNPR, and ZIC traders from the original DBSE. On the other hand, the term "complex traders" will refer to the AA, ZIP, and GDX traders.

3.1.3 Traders AA, GDX, ZIP

The DTBSE's design involved integrating the AA, GDX, and ZIP traders from TBSE.

However, the existing trader types in DBSE were relatively straightforward and had fast algorithms, making it difficult to explore the relationship between algorithm execution time and latency with only these traders. As the existing trader types in DBSE were relatively uncomplicated and executed within short and similar time-frames, this posed challenges in investigating the combined impact of algorithm execution time and latency using only these traders. The inclusion of these trader types necessitated the incorporation of a `respond` function and a Limit Order Book tape, which were not utilised in DBSE. In order to preserve correctness of these trader algorithms, only the most necessary changes were made to the TBSE trader implementations. The verification of these algorithms can be found in the verification section.

With the aim of conducting a direct comparison between DBSE and DTBSE (as described in the experiments section), these traders were also integrated into DBSE. Additionally, an option to run DBSE was introduced in DTBSE for convenient usage.

3.1.4 Multi-Experiment Multi-Trials Runs

Although TSBE had the capability to conduct multiple experiments with multiple runs, the distributed nature of the DBSE architecture, which served as the project's foundation, did not support this functionality.

Thus, various modifications were required, starting with the necessity for users to manually terminate the trading agents at the end of the market session, since the UDP receiver ran continuously in the background. To resolve this, DTBSE incorporated a mechanism to signal the UDP receiver when the trader's execution concluded, allowing for a graceful program termination.

Moreover, DBSE lacked the feature to store session results in a file. Instead, it relied on displaying them on the standard output of the terminal. Similarly, TBSE did not offer a means of remotely retrieving its output files. Considering the objective of scaling up the number of clients and client nodes, this would pose challenges in gathering results from multiple experiments. Hence, in DTBSE, an enhancement was made by incorporating functionality to save and transmit both existing and additional logging information to a remote S3 bucket (Figure 3.1) using the boto3 module, which facilitates remote AWS access.

To accomplish this, the user's AWS credentials and a valid S3 bucket under the associated account are necessary. Therefore, the capability to enable users to specify a bucket name and configure their credentials through the configuration file was introduced. To optimise storage usage and manage costs, these files were compressed into zip archives before being uploaded to S3.

TCP Server

The addition of a basic TCP server served two primary objectives within the system: client synchronisation and exchange reset. The integration of this server within the client's internal architecture can be seen in Figure 3.2 and 3.1.

Client Synchronisation Although DBSE and TBSE had individual approaches for synchronising remote clients and trader threads, relying solely on these mechanisms proved insufficient. Therefore, a basic TCP server was developed to synchronise remote clients and reset the exchange state after each trial.

TBSE utilised a threading event to indicate the start of a market session, which worked well for a locally executed program but posed challenges in a distributed system.

On the other hand, DBSE incorporated Amazon's time synchronisation feature in conjunction with a simple mechanism to initiate each run at the start of the next minute. Nonetheless, this necessitated the user to manually coordinate the initiation of each client to avoid overlapping the next minute mark for the last few clients. For instance, if client 1 was executed at 0:55 minutes and client 2 at 1:05 minutes, client 1's market session would commence at 1:00 minutes, while client 2 would start at 2:00 minutes.

However, as the number of clients increases, timing each client to execute within the same minute becomes increasingly cumbersome. Moreover, when running multiple experiments and trials, this approach has a higher chance of falling out of synchronisation after multiple trials or if any of the other clients experience disruptions.

Thus, the introduction of a TCP server enabled client synchronisation within the exchange. Every remote client transmits a message containing their proposed start time (aligned with the beginning of the next minute, akin to DBSE) and the expected number of clients. Subsequently, the server employed a threading barrier to wait for the specified number of clients to establish connections, followed by selecting the latest proposed start time and relaying it back to each client. Within each remote client, a threading event (similar to TBSE) was utilised to synchronise each trader thread with the agreed-upon time.

Exchange Reset The TCP server was also used to reset the exchange state after each trial.

After each trial, it was necessary to reset the exchange's Limit Order Book for the start of the next trial. While the DBSE already had a cancel order FIX message to remove existing orders at the end of a session, this approach was insufficient for DTBSE due to the inclusion of new traders (AA, GDX, ZIP) that relied on a tape to record recent transactions (as mentioned in the above subsection 3.1.3). Simply relying on a high volume of order cancellations to signify the end of a trial's market session was not reliable. Instead, the TCP server was used to signal the exchange about the session's conclusion and the subsequent clearing of the tape in preparation for the next session.

3.1.5 UDP Unicast

DTBSE utilises the same UDP sender-receiver mechanism as DBSE including the use of UDP unicast over multicast. As stated in Miles' DBSE paper, due to the AWS deployment setup, multicast cannot be used within the AWS VPC. A potential solution was to establish multiple concurrent UDP senders to each client receiver on distinct ports. While this approach might have been feasible, it would have complicated the configuration process for new clients to connect to the exchange, especially as the number of clients increased.

	LDN 1	LDN 2	SDY	OH
Mean	0.841	0.842	138.3	44.34
Standard Deviation	0.278	0.247	1.365	1.102
Minimum	0.419	0.403	136.2	42.13
Q1	0.631	0.669	137.4	43.34
Median	0.781	0.795	138.1	44.12
Q3	1.023	0.971	139.6	45.14
Maximum	1.847	1.821	143.4	49.31

Table 3.1: UDP Latencies Across Clients.
Note: Times are given in milliseconds

To ensure accurate comparison between DTBSE and DBSE, the evaluation of experiment results had to account for varying latencies and compared against those reported in Rollin's paper[30]. Although the clients were set up in the same regions as described in Rollin's paper, it is important to note that there could be minor hardware differences when new EC2 instances are allocated at startup.

Latencies were measured by comparing the timestamp when market data was published to the time it was received by each client.

The latencies are presented in Figure 3.3, and a summary is provided in Table 3.1. When comparing these latencies to those in Rollin's experiment, they exhibit a close resemblance, with the Sydney client experiencing a slight average delay of 2-3 milliseconds.

The most notable difference can be observed in the latency results of the LDN clients. LDN 2 exhibited slightly slower response times compared to LDN 1, as expected due to the unicasting sequence. However, in the DTBSE client latencies, except for the median value, LDN 2 demonstrated slightly faster response times by fractions of a millisecond, along with a slightly smaller standard deviation. This unexpected outcome may be attributed to the hardware configuration of the EC2 instance in AWS. Although these differences are minute, they will be taken into consideration during the analysis of the experiment results to ensure accurate comparisons.

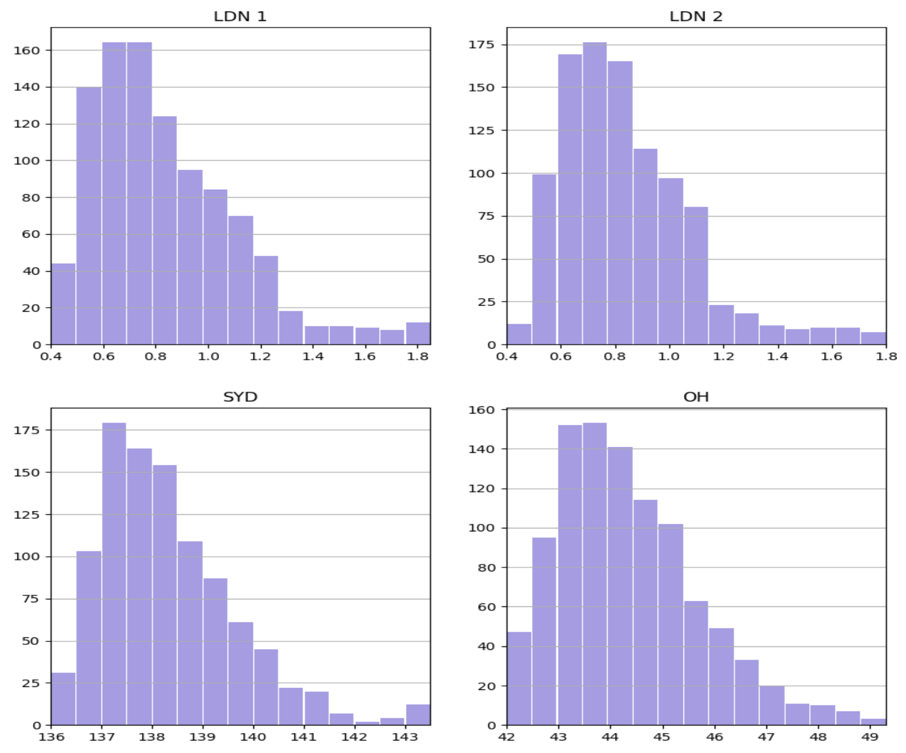

Figure 3.3: Latency Distributions across DTBSE's Clients.

3.1.6 Exchange/Client Configuration

Lastly, a centralised configuration file system was implemented to minimise the number of required arguments and enhance parameter flexibility. This allowed for various configurations such as automatic generation of FIX client config files, specification of AWS credentials, multi-experiment multi-trial runs, TCP/UDP addresses and ports, and more. The introduction of this system simplified the scalability of the overall system.

3.1.7 AWS Cloud Deployment

The project utilises the existing distributed architecture of DBSE on AWS, employing a Virtual Private Cloud (VPC) in each AWS region for network customisation and enhanced security. Peering connections are established between the exchange's VPC and the remote client VPCs. Security groups are configured to direct TCP and UDP traffic within specified IP address ranges to these peering connections. EC2 instances were then deployed within these VPCs. A brief overview of the cloud infrastructure is illustrated in Figure 3.4 and more comprehensive information can be found in Miles' original DBSE thesis[22].

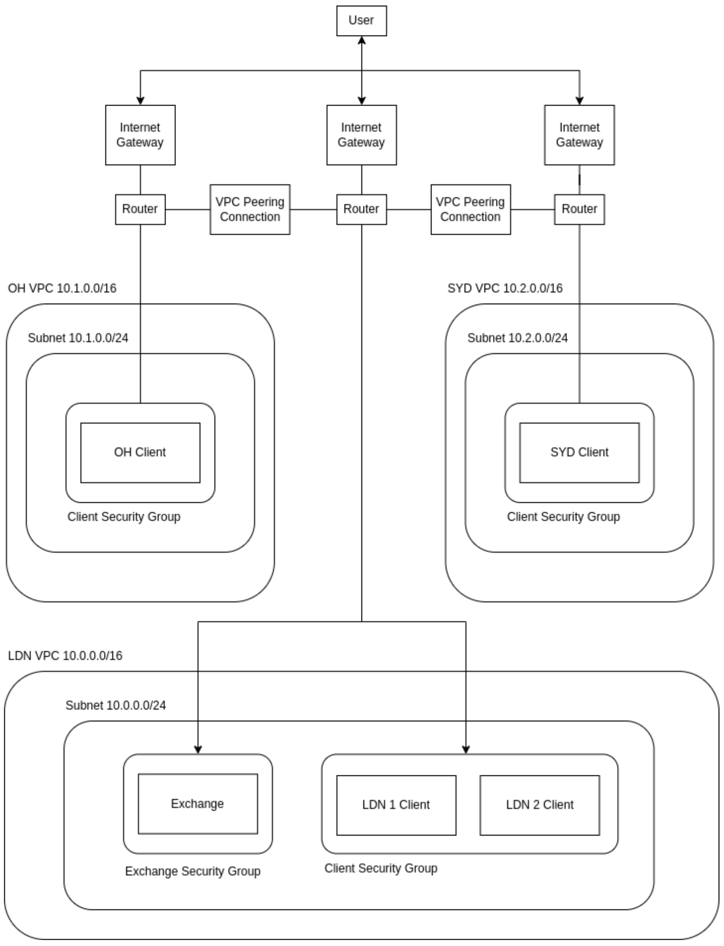

Figure 3.4: Illustration of DTBSE's Cloud Deployment Infrastructure.

AWS provides a diverse range of EC2 instances tailored for various specific purposes, each with its own trade-offs in terms of compute power, storage, and cost. To identify the most appropriate EC2 instance type, a series of tests were conducted to assess the storage, RAM, and CPU utilisation of the DTBSE exchange and clients. These tests aimed to determine the optimal configuration that strikes a balance between performance requirements and cost-effectiveness.

The test parameters are as follows: 5 clients each with 140 trader threads using the simulation configuration shown in Listing A.1.

	Disk Usage (GiB)	RAM Usage (GiB)	CPU Usage (%)
Client	3.050	0.403	13.2%
Exchange	3.307	0.412	17.3%

Table 3.2: Average Application Resource Usage Statistics.

	RAM Usage (GiB)	CPU Usage (%)
Application	0.412	25.3%
Quickfix Installation	3.788	54.3%

Table 3.3: Peak Resource Usage Statistics Across All Operations.
Note: Given CPU usage is observed on a computer with a clock speed of 3.1 GHz.

As seen in Table 3.3, in this specific test configuration, the installation of the quickfix module emerged as the most demanding process, reaching a peak RAM utilisation of 3.79GiB and CPU usage ranging from 47% to 54% at a clock speed of 3.1 GHz. Disk usage was found to be inconsequential, as all Ubuntu instances provided a generous 8GiB of free storage.

However, it should be noted that increasing the number of threads and clients would of course affect resource requirements (5 clients with 140 trader threads each). Consequently, the minimum RAM requirement for the EC2 instances is 4GiB.

To meet these requirements while maintaining cost efficiency, the T3a.medium instance was selected for clients, offering 4GiB RAM at a rate of \$0.0376 per hour. Less expensive instances were either incapable of running the experiments or unavailable.

Meanwhile, given the critical role of the exchange as the central node in the simulation, a deliberate decision was made to utilise a more powerful M5.large instance. This instance type was selected to ensure sufficient resources and enhance the system's stability. Priced at \$0.096 per hour, it provides the necessary computational power and resilience, as any performance degradation or failure in the exchange could potentially disrupt the entire simulation.

It is important to note that running clients on t2.micro instances, which were initially supported by DBSE as part of the free tier (mentioned in Section 2.5.2), became impracticable for this project due to the new RAM requirements for installing QuickFIX. T2.micro instances have only 1GiB of RAM, causing crashes when attempting to install the module. Additionally, the DTBSE exchange needs to handle a significantly higher workload of 560 traders' worth of orders, in contrast to the 160 traders handled by DBSE.

More detail specifications of the two instances are listed below:

- T3a.medium: 2vCPU, 4GiB RAM, AMD EPYC 7000 series processors (AMD EPYC 7571) with an all core turbo clock speed of 2.5 GHz
- M5.large: 2vCPU, 8GiB RAM, Intel Xeon® Platinum 8175M processors/3.1 GHz Intel Xeon Scalable processor (Skylake 8175M or Cascade Lake 8259CL)

3.2 Verification

The verification process focuses on comparing the results of the market session consisting of simple traders to those obtained from DBSE. This analysis serves to validate the functionality and performance of the distributed system and exchange application.

The conclusions drawn from Rollins' paper concerning the alterations in trader dominance relationships in TBSE indicate that the dominance relationship between SHVR, GVWY, and ZIC remains unaltered between the initial non-threaded BSE and TBSE[30]. Therefore, if the system is implemented correctly, there should not be a significant difference between the results of DTBSE and DBSE for this market configuration. The details of this configuration can be seen in Listing 3.1.

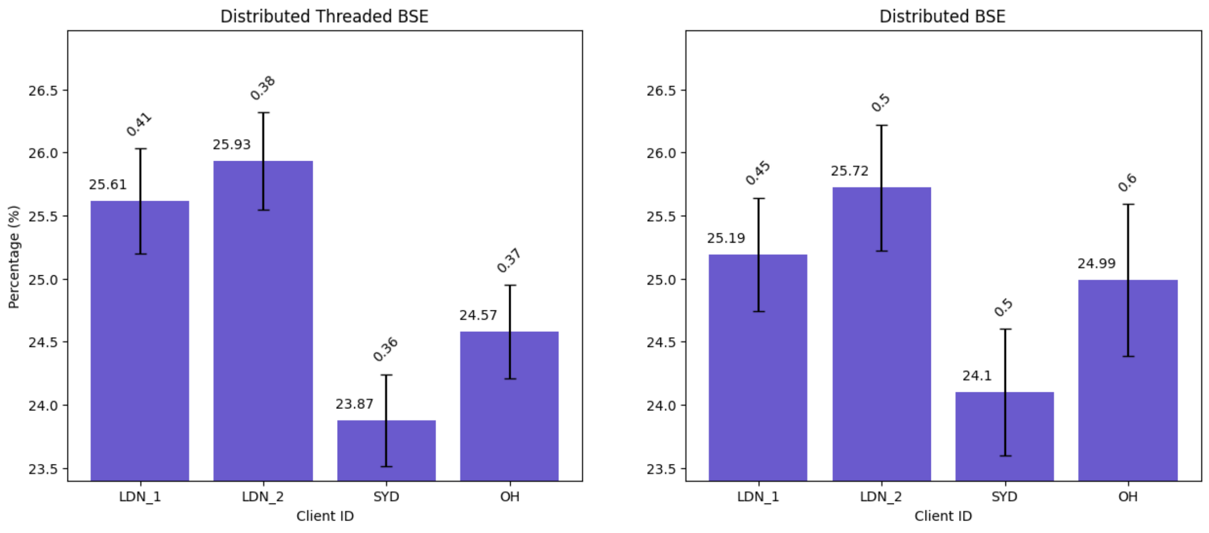

Figure 3.5: Percentage of Total Profit Generated Per Region.

Note: The results displayed in the right figure correspond to the findings described in Miles' paper[22] including error-bar estimates from these results.

Figure 3.5 depicts the average percentage of profits generated per client across 100 runs, with the left graph illustrating the results obtained from DTBSE and the right graph displaying the outcomes from Miles' paper[22].

Both graphs demonstrate the consistent ranking of regions, with both LDN_1 and LDN_2 being the highest, followed by OH and SYD. This ranking aligns with the proximity of the clients to the exchange (LDN) with the largest deviation in the breakdown of remote client profits per region being 0.42%, observed in clients LDN_1 and OH. This discrepancy is negligible and is likely attributed to probability.

Figure 3.6, showing a more detailed analysis of profit per trader, further confirms the consistency of the dominance relationship across regions and in comparison to DBSE. This demonstrates that the dominance relationship is consistently upheld, not only across the entire market, but by each individual client as well as, aligning with the findings of Miles' paper[22].

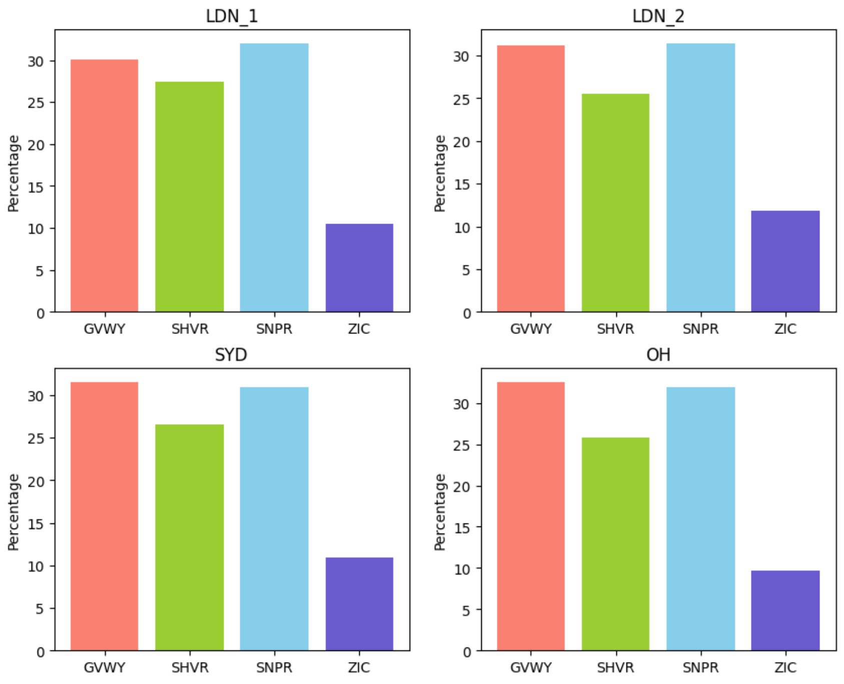

Figure 3.6: DTBSE Profit Percentage Breakdown by Trader Across Regions.

Hence, the results obtained from DTBSE align with those of DBSE, confirming the continued effectiveness of the distributed architecture.

```
{
    "duration": 180,
    "traders": {
        "buyers": {
            "GVWY": 10,
            "ZIC": 10,
            "SHVR": 10,
            "SNPR": 10
        },
        "sellers": buyers
    },
    "order_schedule": {
        "demand": [
            {
                "from": 0,
                "to": 60,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            },
            {
                "from": 60,
                "to": 120,
                "ranges": [{"min": 150.0, "max": 250.0}],
                "stepmode": "fixed"
            },
            {
                "from": 120,
                "to": 180,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            }
        ],
        "supply": demand
        ],
        "interval": 30,
        "timemode": "drip-poisson"
    }
}
```

Listing 3.1: Simulation Configuration for Distributed System Verification Testing.

Chapter 4

Critical Evaluation

4.1 Experiments

Two main experiments were conducted, one following Miles' experiment with DBSE, which involved a balanced multi-trader environment. The second experiment followed Rollin's approach with TBSE and comprised multiple pairwise experiments with varying ratios of the two trader types.

4.1.1 Balanced Multi-Trader Experiment

The initial experiment focused on comparing the results of executing the newly integrated, more complex algorithms in DTBSE with those of DBSE (as explained in Section 3.1.3). This aimed to observe the impact of threading traders in a distributed simulation.

All trader types participated in the market session, engaging with a fixed-stepped symmetric supply-demand curve using a periodic order interval. There were a total of 4 clients, mirroring Miles' experiment on DBSE for a more direct comparison to the results presented in his paper. The exchange was situated in the London VPN, with two clients located in the same London VPC (LDN 1 and LDN 2), while clients 3 and 4 were positioned in Sydney (SYD) and Ohio (OH) respectively. Each client had 10 traders of each type GVYW, SHVR, SNPR, ZIC, ZIP, AA, GDX) on both the seller and buyer sides. Further details of the simulation configuration can be found in Listing A.1.

In the market simulation, it was observed that simpler algorithms tended to perform better, particularly as the distance from the exchange increased. At the LDN 2 client, GVWY, SNPR, and SHVR accounted for 20.4%, 19.4%, and 17.2% of the total profit, respectively. These percentages increased significantly at SYD, with GVWY, SNPR, and SHVR contributing 23.1%, 23.0%, and 20.3% of the total profit. A similar pattern was observed for ZIC, although it appeared to be the least successful among all traders in this scenario. This pattern could be attributed to the high number of traders (140) at each location, totaling 560 traders overall, resulting in rapid price movements. Consequently, speed became a crucial factor in this market.

Meanwhile the performance of complex traders exhibited an opposite trend compared to simple traders. As the distance from the exchange increased, the performance of complex traders seemed to deteriorate. The increased latency seems to have provided simple traders with a higher probability of successfully executing their algorithms and placing orders before the market conditions changed.

Among the complex traders, ZIP stands out as the most consistent performer in terms of profit stability. Its profit ranges from 11.6% to 9.5%, decreasing by approximately 2.1% with increased latency. Analysing Figure 4.1, it can be observed that at clients LDN 1 and LDN 2, ZIP is able to marginally outperform GDX by 0.9% to 1.0% respectively. This outcome aligns with Rollin's findings, indicating that ZIP is capable of surpassing GDX in a threaded environment. As observed in his paper, this is likely due to the fact that ZIP does not take as large of a performance hit as its execution time is far quicker but still able to adapt to the quickly changing market.

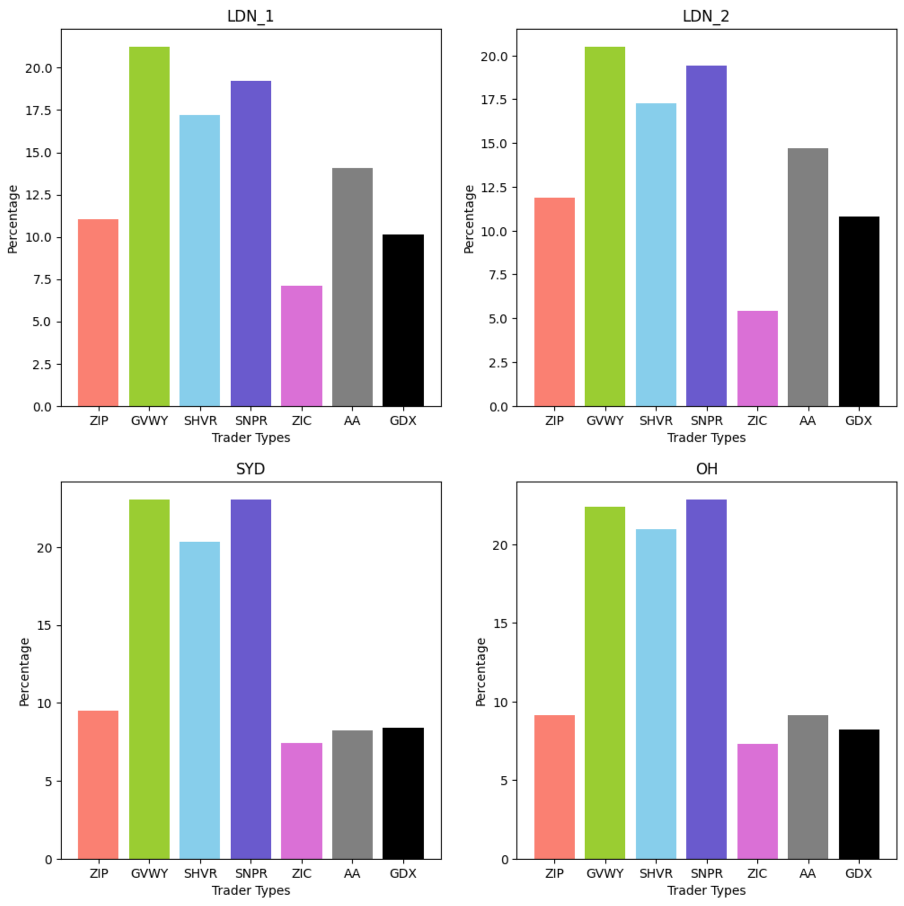

Figure 4.1: Breakdown of trader profits across regions in DTBSE.

As an adaptive algorithm it is expected that the dramatic increase in latency from the LDN client to SYD should incur a higher decrease in profits between these two locations. This may be due to the fact that it is fastest of the three complex traders and as such is able to balance the speed of the simple traders with the adaptive features of the complex algorithms. Another factor may be that as other adaptive strategies experienced a greater decline in performance with increased latency, ZIP is able to maintain its profit better in relation. Its performance in relation to AA and GDX is further discussed later in this section.

In contrast, AA exhibits the most significant variation in performance among the complex traders. At LDN 2, AA manages to capture 14.7% of the total profit, while this slightly decreases to 13.9% at LDN 1, despite both clients being in the same region. As demonstrated in the latency testing (Section 3.1.5), LDN 2 shows a slightly faster response in receiving market updates, resulting in slightly higher profits compared to LDN 1, as observed in the verification test (Section 3.2). The discrepancy in AA's profit can be attributed to its high sensitivity to market conditions and the most recent bid/ask prices registered on the exchange. AA employs a highly adaptive trading algorithm that relies on a rolling average of the past N bid/ask prices, assigning greater importance (i.e. higher weighting) to more recent transactions. This approach is based on the assumption that the market is constantly moving towards an equilibrium price.

As the algorithm heavily relies on recent trades, it is plausible to assume that obtaining market data slightly ahead of its competitors could enhance its likelihood of executing profitable orders. Consequently, even minor latency variations resulted in differing profits for AA between the two London clients. This trend is further evident at the Sydney and Ohio clients, where AA's profits significantly decrease to 8.2% and 9.1% respectively.

GDX, on the other hand, exhibits a less noticeable decline in profits but is still impacted by increased latency. Although it is also an adaptive algorithm that relies on a history of the last N trades, GDX operates differently from AA. Unlike AA, GDX does not assign higher weights to more recent trades, as it does not explicitly assume that the market is consistently moving toward an equilibrium price. Instead GDX utilises a belief function to estimate the probability of a trade being accepted within the remaining time window, based on a history of accepted and rejected trades. This belief function converges as the observed trades also converge but it is not an explicit assumption.

Additionally, GDX's longer run time may mitigate the impact of increased latency to some extent, as its order submissions are consistently behind the current market. This observation is supported by the findings in Rollins' paper[30], which also noted the execution time impact on GDX when moving to a threaded environment. As previously mentioned, being an adaptive strategy, the GDX strategy relies on a belief function that is dependent on a history of trades from the exchange. Therefore, it is still influenced by the latency in receiving market updates. This is evident in the results, as its profits decline from 10.7% (LDN 2) to 8.4% (SYD).

The overall ranking of complex traders aligns with expectations, reflecting the dominance hierarchy observed in Rollin's paper[30]. AA consistently outperforms other traders across most clients, while ZIP follows closely in second place but with a more stable profit. GDX, with its longer execution time, consistently ranks last among the three traders at each location. An interesting observation is that at the furthest location, SYD, ZIP manages to maintain its profit despite the significant increase in latency (from approximately 0.84 ms to 138.3 ms), while AA experiences a significant drop in profit.

In comparison to Rollin's results in the non-offset experiment[30], it is surprising to see that ZIP was able to outperform AA in the non-offset conditions. It is important to note that there are several differences between the experiments, such as the market session length, equilibrium price, and timemode (full details can be found in Listing A.1) which could have also influenced these results.

When considering all combined traders, their rankings closely align with those described in Rollin's no-offset experiment. This suggests that, for this specific market configuration, the inclusion of real-time latency similarly tends to benefit algorithms with faster execution in this specific market configuration in a way similar to the introduction of an asynchronous market. It would be intriguing to compare these findings with a variable offset condition in TBSE's threaded but non-distributed system to determine if there are more significant differences in the results in such a market environment.

Additionally, it is important to acknowledge that the large number of traders (560) could potentially contribute to a decrease in the profits of complex traders, as the high volume of trades placed at the exchange may lead to increased competition and lower individual profits. Hence, it would be intriguing to explore two scenarios: one without simple traders in order to focus observations on the interaction and response of complex traders to increased latency, and another with a smaller number of traders, creating a more manageable environment for adaptive traders to avoid being overwhelmed by the high volume of trades.

4.1.2 Pairwise Variable Ratio Experiments

Experiment Setup

Due to limitations in time and budget, a subset of Rollin's experiments was replicated, with some simplifications made to the tests[30]. Additionally, a variable offset was introduced in this iteration.

The experiment was set up as follows: The exchange was located in London, with one client (LDN 1) in close proximity to the exchange and another client (SYD) situated remotely in Sydney. Each client was assigned only one type of trader. This setup aimed to control other variables and focus specifically on examining the impact of latency and execution time on trader dominance.

Initially, the London client had 2 traders of type A, while the Sydney client had 38 traders of type B. The ratio between the two types would then be adjusted until there were 38 type A traders in London and 2 type B traders in Sydney.

Due to limitations in time and cost, a total of four paired tests were conducted, with each pair consisting of five different ratios. For each ratio, 100 runs were performed, resulting in a total of 2000 individual runs.

In this experiment, the market configuration was designed to replicate Rollin's experiment. It utilised a symmetric supply-demand curve with a variable offset, where customer limit orders were randomised and a fixed stepmode was employed[30]. However, there were slight differences compared to Rollin's experiment. In DTBSE, the market session duration was set to 100 seconds, and the order interval was fixed at 5 seconds, occurring periodically. The original experiment was conducted for a duration of 600 seconds, with an order interval of 30 seconds. This setup resulted in a total of 20 customer limit orders being placed during the experiment.

However, in this configuration, the periodic schedule mode resulted in traders being idle for most of the 30-second order interval. To address this, a calculation was performed to determine a suitable market session length and order interval that would allow for the execution of 20 customer limit orders while providing sufficient time for each trader to participate.

The selected traders for the experiments were ZIC, SHVR, GVWY, ZIP, and AA. Among them, AA had the slowest execution time, estimated to be approximately (TODO). This information can be found in Section (TODO).

In this experiment, the maximum number of threads on a single client was determined to be 38 traders * 2 buy/sell sides, resulting in a total of 76 trader threads. Considering a liberal worst-case scenario, one round of AA trade executions would take 4 overhead scale factor * 76 trader threads * AA execution time 0.155ms resulting in 47.12ms. Therefore, in theory, a 5-second order interval should provide sufficient time for all traders to execute. To verify this, a trial run was conducted, and it was observed that all eligible orders were successfully submitted to the exchange within the chosen order interval, thanks to the periodic nature of the order schedule

To accommodate 20 customer order limits, a market session length of 100 seconds was selected. This choice aimed to replicate Rollin's original experiment as closely as possible while minimising the overall run time and, consequently, reducing AWS EC2 and cloud infrastructure costs.

From a combination of the above mentioned traders, four different pairs were selected to encompass various scenarios from Rollin's original experiment[30]. These pairs included one where trader A emerged as the winner in both BSE and TBSE (AA vs ZIC), one where trader A won in BSE but lost in TBSE (SHVR vs ZIP), and vice versa, with trader B (GVWY vs SHVR and ZIP vs ZIC).

This experimental setup was designed to investigate the impact of latency on different trader cases, considering how the threaded nature of TBSE affected each trader differently. Additionally, five ratios of A:B traders were selected to represent a range of scenarios, these cases being: few vs many, roughly a quarter vs three-quarters, balanced, three-quarters vs one-quarter, and many vs few. This selection aimed to cover the full range of trader scenarios and ratios within a reasonable time-frame and budget.

The following section provides an analysis of the experimental results. It is important to mention that the results from both BSE and TBSE are sourced from Rollin's paper for a convenient side-by-side comparison with his findings. In each set of experiments, the average Equal Availability Ratio (EAR) is used to determine a winner, and further details on this metric can be found in Section (TODO).

AA vs ZIC

In the AA vs ZIC experiment depicted in Figure 4.2, the results in all four markets exhibit a consistent pattern where AA outperforms and dominates ZIC across all ratio scenarios. This outcome is not unexpected, considering that ZIC is a zero intelligence algorithm, which has been shown to perform poorly in both BSE and TBSE (as observed in Rollin's research). While this observation is not groundbreaking, it serves as an example to emphasise that although execution speed plays a critical role in a trader's performance, there is a threshold beyond which speed alone cannot compensate for the intelligence of an algorithm.

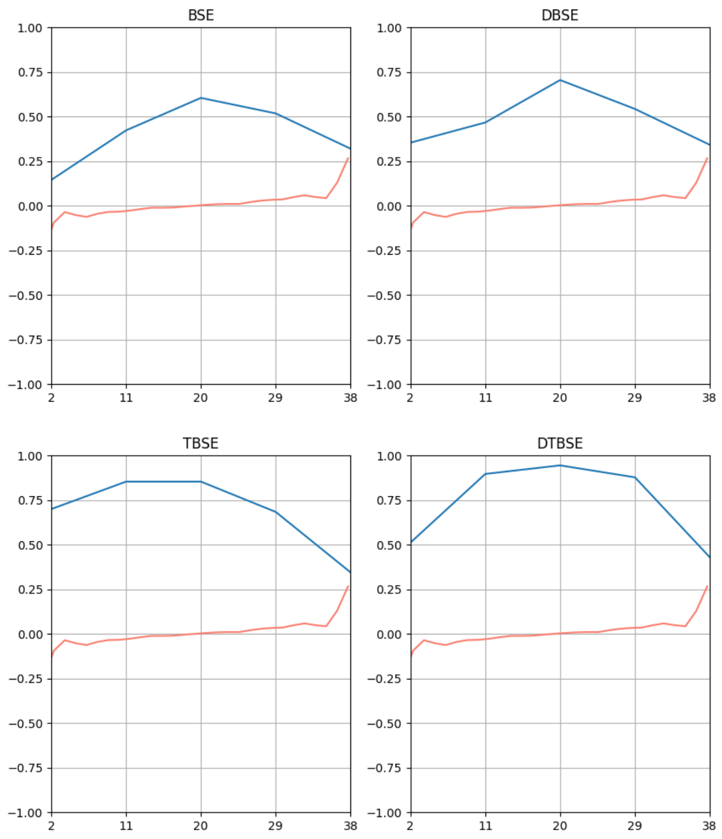

Figure 4.2: AA vs ZIC Win Ratio Comparison.

Note: The average Equal Availability Ratio (EAR) is shown in red on the graph. The x-axis represents the number of AA traders (Type A), while the y-axis represents the raw AA/ZIC win ratio. The range of the y-axis is [-1, 1], where a value y of +1 indicates a complete victory for AA, 0 represents an equal number of wins for both AA and ZIC, and -1 signifies a total victory for ZIC. BSE and TBSE results are sourced from Rollins' paper for an easier side by side comparison.

As noted in Rollin's study, AA demonstrates the ability to exploit random ZIC orders that deviate from the equilibrium price, while other ZIC traders simply disregard such opportunities by posting their own random orders[30]. This observation holds true in both the distributed experiments. These findings highlight the importance of striking a balance between execution speed and trader intelligence.

A noteworthy finding is that in all simulations, the performance of AA tends to decrease at ratio extremes, except for TBSE when there are few AA traders. This trend is especially prominent when there are significantly more AA traders compared to ZIC. In this scenario, the AA/ZIC raw win ratio line closely approaches the EAR line, indicating a closer competition between AA and ZIC.

In scenarios with few AA traders, the observed decrease in performance may be attributed to the introduced latency, which allows other ZIC traders to react to prices close to equilibrium before AA. This effect is reminiscent of Rollin's findings regarding BSE, where the synchronous nature of the system limits the opportunity for AA traders to take advantage if another ZIC trader is selected to respond after the close-to-equilibrium price. This is likely why DBSE exhibits a similar pattern to BSE. On the other hand, DTBSE's asynchronous clients allow AA to dominate the market completely, particularly within the ratio range of 11:29 to 29:11, in a fashion more similar to TBSE.

However, when there is a large number of AA traders, regardless of the simulation type, they start competing against each other, resulting in a significant decrease in AA performance across all four simulators.

SHVR vs ZIP

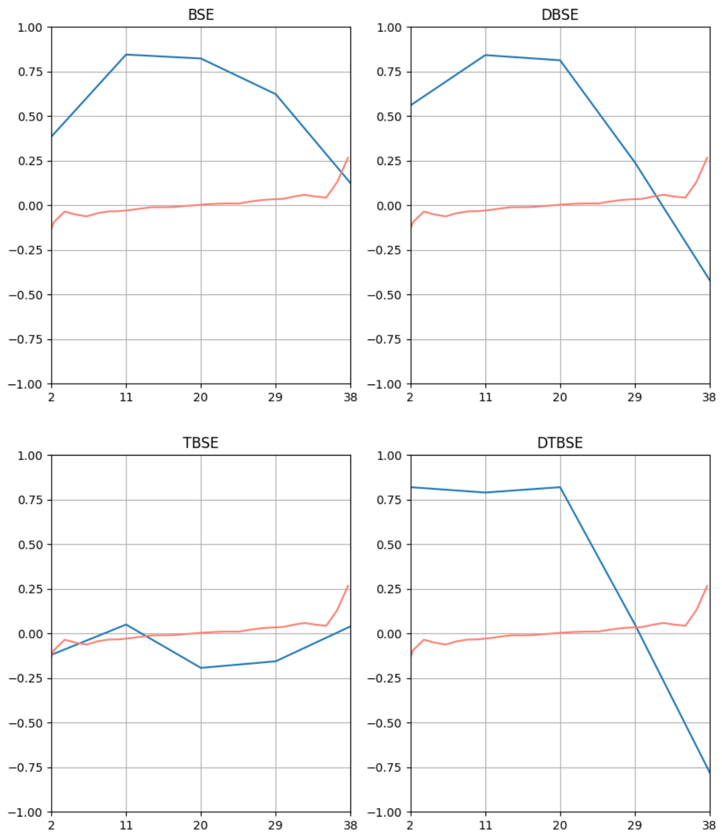

Figure 4.3: SHVR vs ZIP Win Ratio Comparison.

Note: The average Equal Availability Ratio (EAR) is shown in red on the graph. The x-axis represents the number of SHVR traders (Type A), while the y-axis represents the raw SHVR/ZIP win ratio. The range of the y-axis is [-1, 1], where a value y of +1 indicates a complete victory for SHVR, 0 represents an equal number of wins for both SHVR and ZIP, and -1 signifies a total victory for ZIP. BSE and TBSE results are sourced from Rollins' paper for an easier side by side comparison.

As observed in Rollin's paper, in the BSE market, SHVR dominates over ZIP due to its ability to execute a higher number of trades. This dominance extends to all market configurations in the current experiment.

From Figure 4.3, it is evident that DBSE initially follows a similar trend as BSE, with SHVR dominating over ZIP for a lower number of SHVR traders. However, as the number of ZIP traders exceeds the number of SHVR traders, there is a sharp decline in SHVR performance in DBSE. This decline is more pronounced than in BSE. There are two potential factors contributing to this:

1. Firstly, when there are too many SHVR traders in the market, there are not enough ZIP traders to provide reasonable orders for the SHVR traders to undercut. As a result, the SHVR traders end up competing with each other without effectively executing trades against other SHVR traders. On the other hand, ZIP traders can quickly adapt to the market conditions and secure more trades compared to the SHVR traders.

2. The second potential factor is related to the distributed nature of DBSE. While each client executes trades sequentially, the execution between clients is done in parallel, introducing a slight degree of asynchronous trader execution. However, since there are only two clients in this configuration, the impact of this asynchronous execution is minimal. This is reflected in the results, as DBSE closely follows the trend of BSE rather than TBSE.

Rollin's experiment on TBSE demonstrates that the asynchronous execution leads to a faster movement of the equilibrium price, making it difficult for SHVR traders to successfully undercut trades. As a result, ZIP is able to dominate the market in TBSE. Interestingly, DTBSE, which was initially expected to exhibit a similar pattern to TBSE, displayed a shape more akin to BSE and DBSE. The introduction of significant latency between the two locations (138.3ms) had a larger impact than anticipated. This could be attributed to the latency being so substantial that even though ZIP traders are generally faster than other complex traders at adapting to market changes in TBSE, the SHVR traders located closer to the exchange are able to undercut ZIP's orders by default in DTBSE. It would be worthwhile to explore this experiment further by varying the latency to examine how it influences ZIP's performance against the Shaver traders.

ZIC vs ZIP

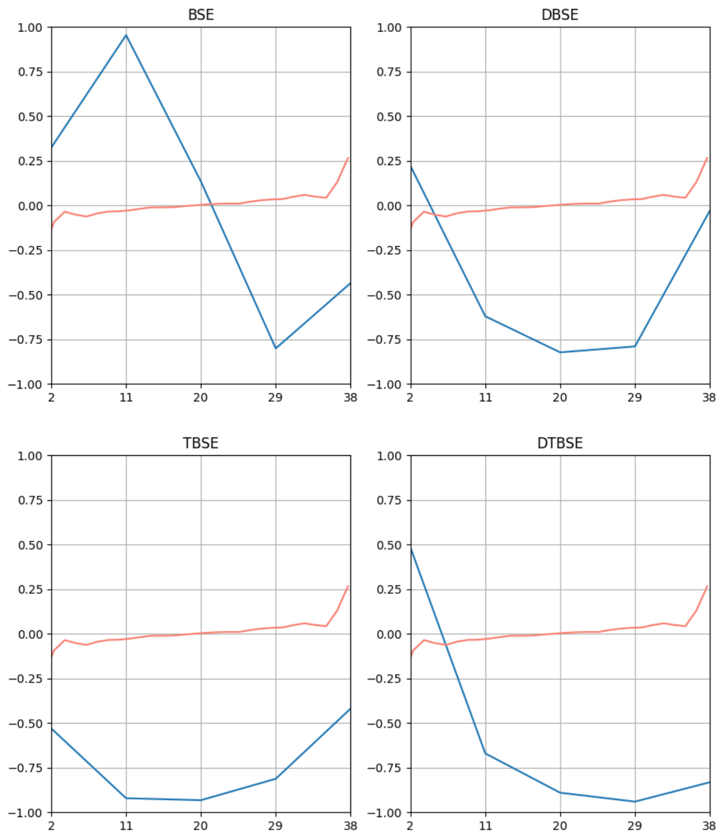

Figure 4.4: ZIC vs ZIP Win Ratio Comparison.

Note: The average Equal Availability Ratio (EAR) is shown in red on the graph. The x-axis represents the number of ZIC traders (Type A), while the y-axis represents the raw ZIC/ZIP win ratio. The range of the y-axis is [-1, 1], where a value y of +1 indicates a complete victory for SHVR, 0 represents an equal number of wins for both ZIC and ZIP, and -1 signifies a total victory for ZIP. BSE and TBSE results are sourced from Rollins' paper for an easier side by side comparison.

Figure 4.4 depicts the results for the ZIC vs ZIP comparison.

As explained by Rollins, BSE exhibits a relatively equal distribution of wins between ZIC and ZIP. This outcome is somewhat unexpected, given that ZIP is considered a more intelligent algorithm. In a previous experiment conducted by Rollin on BSE and TBSE without the variable offset, both simulations favoured ZIC, similar to the BSE trend observed in Figure 4.4. However, in this BSE simulation with the variable offset, ZIP performs worse compared to Rollin's no offset experiment. Rollins' suggested that it might be possible that the variable offset used in these experiments presents challenges for ZIP to adapt effectively to market fluctuations. Furthermore, the situation becomes even more unexpected as ZIP manages to completely overpower ZIC in TBSE, despite facing greater difficulties due to the asynchronous nature of the system, which makes it harder for ZIP to adapt to market conditions[30].

Moreover, DBSE and DTBSE demonstrate a consistent trend similar to TBSE, indicating that ZIP has the ability to outperform ZIC in these simulations. However, it is intriguing that this dominance is not observed in the BSE environment. One possible explanation for this disparity is that in the BSE simulation, there might not be a sufficient number of accepted trades for ZIP to accurately update its margin in response to the variable offset changes. In contrast, TBSE's asynchronous nature enables ZIP to promptly respond to all LOB trades and adjust its margin in real-time as each trade occurs. This immediate response may account for ZIP's success in TBSE. Additionally, the similar patterns observed in DBSE and DTBSE can be attributed to their parallel execution mechanisms, where DBSE allows parallel execution between clients and DTBSE allows parallel execution between clients and traders within each client. Nonetheless, further research is necessary to draw conclusive findings in this regard.

While overall they exhibit similarities, some noticeable distinctions can be observed among DBSE, TBSE, and DTBSE, especially at the extremes of the ratio. At lower ratios of ZIC traders, their behaviours diverge, with ZIP clearly winning in TBSE, the opposite in DTBSE, and DBSE showing a slightly closer outcome than DTBSE. On the other hand, when the ratio of ZIP traders is smaller, DTBSE allows ZIP to completely dominate, while TBSE follows suit but to a lesser degree, and DBSE presents a much closer competition, with the raw ZIC/ZIP win line approaching the EAR line.

One limitation of this experiment, which becomes particularly evident in these findings, is that the ratio resolution is insufficient to capture the finer details of the graph's shape. It is possible that DBSE and DTBSE actually exhibit a similar trend to BSE at the beginning, specifically between ratios 2:38 and 11:29. This will be further explored in Section 5.3.2.

GVWY vs SHVR

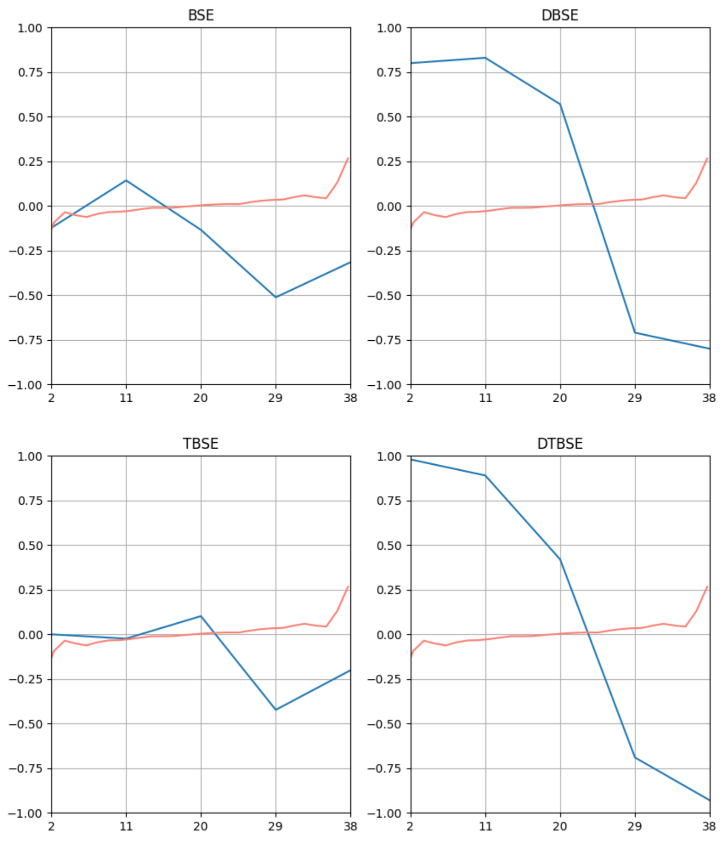

Figure 4.5: GVWY vs SHVR Win Ratio Comparison.

Note: The average Equal Availability Ratio (EAR) is shown in red on the graph. The x-axis represents the number of GVWY traders (Type A), while the y-axis represents the raw GVWY/SHVR win ratio. The range of the y-axis is [-1, 1], where a value y of +1 indicates a complete victory for SHVR, 0 represents an equal number of wins for both GVWY and SHVR, and -1 signifies a total victory for SHVR. BSE and TBSE results are sourced from Rollins' paper for an easier side by side comparison.

Figure 4.5 presents the outcomes of the GVWY vs SHVR experiment. It is interesting to observe that, in contrast to the findings in Rollin's TBSE paper[30], where the threaded trader execution had minimal impact on the dominance relationship between GVWY and SHVR compared to the original BSE, a noticeable switch occurs when incorporating the distributed architecture in both DBSE and DTBSE simulations.

This observation aligns with the findings reported in Miles' DBSE paper, where GVWY dominates SHVR in the presented results. It's important to note that the experiments differ in terms of market configurations and the inclusion of additional trader types (ZIC and SNPR) in Miles' study. Nevertheless, it is interesting to see that the pairwise experiment's result is supported by Miles' findings[22].

It is worth noting the variation in wins observed across different ratios, with all four simulations demonstrating that SHVR starts to win or win more decisively as the number of GVWY traders increases. This could be attributed to the increased number of GVWY traders placing orders, creating more opportunities for SHVR to place orders that are only one price unit better.

Conversely, in markets with many SHVR traders, they end up competing against each other to outbid/outask one another, allowing the few GVWY traders to secure a number of trades that accumulate small profits over time, while many SHVR traders are unable to trade. This is evident by the proximity of the GVWY/SHVR line to the EAR line in BSE and TBSE, or its position above the EAR line in DBSE and DTBSE, only in markets with fewer GVWY traders. However, the line drops below the EAR line again once more GVWY traders are added. However, despite this trend, it is important to note that GVWY does not emerge as the overall winner in the BSE and TBSE simulations.

A noteworthy difference between BSE/TBSE and DBSE/DTBSE, aside from the inverted dominance relationship, is the more pronounced shape and steep change in gradient observed between GVWY:SHVR ratios of 11:29 to 29:11. While BSE and TBSE exhibit a range of approximately 0.1 to 0.5, DBSE and DTBSE cover almost the entire range from -1 to +1.

The introduction of such significant latency (approximately 138.3ms as discussed in Section 3.1.5 has resulted in a dramatic exaggeration of the curve, making it an interesting observation.

One potential explanation is that the significant latency exaggerates the effect previously mentioned regarding the many-to-few relationship between the two types of traders. In the scenario where there are fewer Giveway traders and more Shaver traders, even if the Giveway trader is farther away from the exchange, the nearby Shaver traders have ample time to compete with each other, continuously adjusting the ask or bid prices. However, due to the relaxed nature of the Shaver traders, the price will gradually converge to the equilibrium price, resulting in fewer Shaver traders executing their orders while there are only Shaver trader-initiated orders. By the time Giveway is able to respond to the market, the price has already moved in a more favourable direction due to the faster response of the Shaver traders.

Under similar market conditions, but with the Giveway trader positioned closer to the exchange, the trader's more urgent nature allows it to execute multiple trades before the Shaver traders can respond. As a result, the Giveway trader can gradually accumulate profits. The same principle applies to the Shaver trader in the opposite market conditions, where its quicker response enables it to secure trades before the Giveway traders can react, leading to potential profit gains.

4.1.3 Limitations

Due to time and budget constraints, the cost of conducting experiments was a limiting factor. The cost involved replicating and running experiments for both DBSE and DTBSE setups, requiring multiple instances and corresponding S3 storage for data runs. Additionally, since there was no mechanism for real-virtual time conversion (see Section 3.1.1), the experiments had to be run in real time, resulting in approximately 17 hours for each set of experiments. This limited the extent of the tests that could be performed compared to Rollin's paper[30], which involved 285,000 market sessions per experiment, requiring 280 hours of experiments. Given the constraints of this project, replicating such a comprehensive set of tests was unrealistic. However it may be interesting to hone in on certain pairwise experiments to create a more fine grained win ratio curve.

For similar reasons it was not feasible to run a large number of balanced multi-trader experiments. These experiments required more resources, including 4 T3a.medium and 1 M5.large instances per experiment, compared to the pairwise experiments. However, potential experiments focusing on this area are discussed in more detail in the Further Work Section.

If a future study were to prioritise one of these experiments, it would offer the opportunity for a thorough investigation of the market conditions and the resulting interactions among traders. This deeper analysis would shed more light on the behaviour of the system under specific scenarios.

Chapter 5

Conclusion

5.1 Summary

The project successfully achieved the goal of developing a distributed and threaded version of BSE, demonstrating its ability to handle 560 traders in a single market session without any issues. The project architecture incorporated various features such as remote data logging, configurable settings through a single config file, automatic client synchronisation and exchange reset via a TCP server, as well as queuing multiple experiments and multiple runs of each experiment. Based on this current architecture, it is anticipated that the system can be further scaled up to handle even larger numbers of traders in a single session. This scalability can be achieved through manual addition of more clients (horizontal scaling) or by utilising more powerful instances (vertical scaling).

The system's design includes plans for future development to incorporate automatic scaling of remote clients. However, the project does not currently implement this approach, focusing on laying the groundwork for scaling up the system using AWS EKS's Kubernetes clusters. This is further discussed in Section 5.3.1.

Following the initial implementation of the distributed and threaded system, a series of experiments were carried out to examine the effects of latency and asynchronous trader execution. These experiments sought to compare the distributed and threaded version of BSE with its previous iterations, drawing inspiration from the research made by Miles[22] in DBSE and Rollins[30] in TBSE. The two main experiments were carried out: one replicating Miles' experiment with DBSE in a balanced multi-trader setting, and the other following Rollin's approach with TBSE, consisting of multiple pairwise experiments using different ratios of the two trader types.

Balanced Multi-Trader Experiment The experiment compared the performance of complex algorithms in DTBSE and DBSE, focusing on the impact of threading traders in a distributed simulation. The market session included various trader types and followed a fixed-stepped symmetric supply-demand curve with a periodic order interval. Four clients were involved, located in London, Sydney, and Ohio, each with 10 traders of each type on both the buyer and seller sides.

The results showed that simpler algorithms performed better, especially with increased latency. GVWY, SNPR, and SHVR were the most profitable traders, particularly in markets with higher latency. ZIC performed the worst among all traders. The performance of complex traders deteriorated with increased latency, but ZIP stood out as the most consistent performer, at times surpassing AA and GDX. While AA was able to secure high profits in LDN 1 and LDN2, it was sensitive to market conditions, resulting in varying profits even within the same region. GDX exhibited a less noticeable decline in profits due to its longer execution time and belief function approach. ZIP consistently outperformed GDX and demonstrated better adaptability to increased latency.

The rankings of complex traders were consistent with previous findings, with AA consistently outperforming others, ZIP closely following, and GDX ranking last. The inclusion of real-time latency tended to benefit algorithms with faster execution, similar to an asynchronous market. The large number of traders may have contributed to a decrease in complex traders' profits due to increased competition. Further exploration could involve scenarios without simple traders or with a smaller number of traders to observe how complex traders interact and respond to latency.

Pairwise Variable Ratio Experiments The pairwise experiments were conducted to compare DTBSE and DBSE with Rollin's experiments, focusing on the impact of latency and execution time on trader dominance. A subset of Rollin's experiments was replicated with simplifications and a variable offset due to time and budget constraints. The setup involved an exchange in London and a client in Sydney, with each client assigned a specific trader type. The experiments included adjusting the trader ratios and conducting four paired tests. The results were analysed using the average Equal Availability Ratio (EAR) and compared to Rollin's findings.

AA vs ZIC In the AA vs ZIC experiments, AA consistently outperformed and dominated ZIC across all ratio scenarios in both the DTBSE and DBSE simulations. This aligns with previous findings that highlight the superior performance of AA compared to ZIC. These tests show that although execution speed is crucial, there is a limit to how much speed alone can compensate for the intelligence of an algorithm.

Interestingly, the performance of AA tends to decrease at ratio extremes in all simulations, except for TBSE when there are few AA traders. This decrease in performance can be attributed to the introduced latency, which allows other ZIC traders to react to prices close to equilibrium before AA. This observation is similar to the findings in BSE, where the synchronous nature of the system limits the opportunity for AA traders to exploit deviations from equilibrium. However, in DTBSE with asynchronous clients, AA can dominate the market more effectively within a certain ratio range.

On the other hand, when there is a large number of AA traders, they start competing against each other, leading to a significant decrease in AA's performance across all simulations. This suggests that a balance needs to be struck between the number of AA traders and their performance in order to achieve optimal results.

SHVR vs ZIP In the SHVR vs ZIP experiment,it was observed that SHVR dominates over ZIP in all market configurations in the above experiment.

In DBSE, initially, the trend is similar to BSE, with SHVR dominating over ZIP when there are fewer SHVR traders. However, as the number of ZIP traders exceeds the number of SHVR traders, there is a significant decline in SHVR's performance in DBSE, more pronounced than in BSE. Two potential factors contribute to this: the increased competition between SHVR traders as well as the inherent client level parallelism. In TBSE, the faster movement of the equilibrium price due to asynchronous execution poses challenges for SHVR traders to undercut trades effectively, enabling ZIP to dominate the market. However, in DTBSE, contrary to expectations, the introduction of substantial latency results in a pattern more similar to BSE and DBSE. This is because the latency allows SHVR traders located closer to the exchange to undercut ZIP's orders by default, despite ZIP traders' general agility in adapting to market changes in TBSE.

ZIC vs ZIP Rollins' findings on BSE suggest a relatively equal distribution of wins between ZIC and ZIP, contrary to expectations considering ZIP's intelligence. However, in this BSE simulation with a variable offset, ZIP performs worse compared to Rollins' no offset experiment[30]. It was suggested that the variable offset may present challenges for ZIP to adapt effectively[30]. Surprisingly, ZIP manages to overpower ZIC in TBSE, despite facing greater difficulties due to the asynchronous nature of the system. DBSE and DTBSE exhibit a consistent trend similar to TBSE, indicating ZIP's continued dominance over ZIC. However, this dominance is not observed in the BSE environment, possibly due to insufficient accepted trades for ZIP to update its margin accurately. The differences among DBSE, TBSE, and DTBSE become more apparent at extreme ratios. Unfortunately the experiment's limitation includes insufficient ratio resolution to capture finer details in the graph's shape (expanded upon in Section 5.3.2.

GVWY vs SHVR The GVWY vs SHVR experiment revealed a notable switch in dominance between the two traders when incorporating the distributed architecture in both DBSE and DTBSE simulations. This finding aligns with the results reported in Miles' DBSE paper, where GVWY outperformed SHVR. The variation in wins across different ratios suggests that SHVR tends to win or win more decisively as the number of GVWY traders increases. However, in markets with many SHVR traders, competition among them allows a few GVWY traders to secure profitable trades over time. This trend is more pronounced in DBSE and DTBSE simulations.

Both sets of experimental results demonstrate the significant impact of latency, especially for complex traders, when competing against simpler traders in markets with a large number of participants. The advantages of certain traders, were able to overcome the negative effects of latency in specific scenarios, as observed in the pairwise experiments involving AA vs ZIC, SHVR vs ZIP, and ZIC vs ZIP.

5.2 Project Status

The project's progress will be summarised and quantified in relation to the list of aims and objectives provided in the Introduction.

1. The successful creation of a Distributed and Threaded BSE (DTBSE) involved integrating features from both Miles' and Rollins' work. Through experiments, it was observed that traders are executed asynchronously, taking into account real-world latency conditions.
    (a) The DBSE section in the Technical Background provides a comprehensive summary of Miles' architecture, demonstrating the thorough investigation of his work.
    (b) Based on the above analysis, it was concluded that DBSE served as a suitable foundation for DTBSE.
    (c) The deployment of DTBSE in the AWS cloud was successful, enabling remote clients to establish connections with the exchange
    (d) The TBSE section in the Technical Background, evidence is presented regarding the architectural design of TBSE that influenced the development of DTBSE. The complex traders (AA, GDX, ZIP) from TBSE were also successfully incorporated into DTBSE.
2. Implemented a TCP server and client to facilitate the synchronisation of clients automatically in DTBSE.
3. DTBSE was designed to be configurable, allowing for the execution of multiple experiments and trials by utilizing a straightforward configuration file.
4. The TCP server and clients were enhanced to include a functionality that signals the exchange to reset its state for the next experiment.
5. DTBSE demonstrated its capability to handle a total of 560 traders in a single market session.
6. An analysis of the results obtained from recreating experiments based on the work of Miles and Rollins in DTBSE was conducted.

5.3 Future Work

There are numerous avenues for future development and expansion of both the system and future experiments.

Due to the project's dual focus on developing the DTBSE system and conducting experimental investigations, it was not possible to fully delve into specific avenues of exploration. As a result, numerous opportunities for system development and further investigations were left unexplored. The project's divided attention limited the depth of exploration in both areas, leaving room for future work to explore and expand upon.

5.3.1 DTBSE Extensions

In order to enhance scalability and ease of use, a possible extension would be to utilise AWS EKS (Elastic Kubernetes Service) with Kubernetes.

The system was already initially designed with scalability in mind, and incorporating Kubernetes clusters would enable automatic scaling of resources. While it is already fairly simple to add additional clients to the exchange, this extension would make it easier and faster to add new clients to the system, as the scaling process would be automated.

Additionally, running multiple experiments simultaneously would become more feasible, eliminating the need for manual connection and execution of clients. Currently, in order to conduct simultaneous experiments, the tool "Screen" was utilised as a terminal multiplexer[15]. This enabled the running program to be detached from the terminal window, allowing us to close the AWS SSH connection without interrupting the client's execution. By utilising Screen, we were able to run experiments concurrently and maintain the program's functionality even when the terminal session was disconnected. With Kubernetes, it would be possible to manage and monitor all nodes and experiments more efficiently, simplifying the overall workflow and enhancing the user experience.

Further expanding on that, the creation of a user-friendly web interface would make the program more accessible. The implementation of a web interface would open up opportunities for users without prior experience in AWS or computer science to utilise the program effectively. This inclusiveness may allow individuals from diverse backgrounds to contribute to research in the field by easily testing their own trading algorithms. Furthermore, a web interface holds educational potential, as it could be utilised to introduce students without computer science skills to the concepts and practices of algorithmic trading. By lowering barriers to entry, the web interface could potentially wider participation and promote more research in the field.

Also, a web interface would provide researchers with a centralised platform where they can conveniently monitor all clients and execute multiple experiments simultaneously. By having a single area to oversee the clients and manage experiments, the web interface streamlines the process and makes it more convenient for researchers to conduct their work efficiently. This could then even possibly be developed into a permanently hosted web server.

Both the creation of a user-friendly web interface and hosting a 24/7 web server would entail additional expenses in terms of AWS infrastructure. The continuous operation of a web server would involve costs for server maintenance, data storage, and network usage. Therefore, for a future project in this direction it is important to consider the financial implications of these features when planning the implementation and deployment of the system.

5.3.2 Further Investigations

Balanced Multi-Trader Experiment

Due to limitations in resources and time (Section 4.1.3, it was not possible to run a comprehensive set of balanced multi-trader experiments. Instead, the focus was on directly comparing the results to those obtained by Miles[22], incorporating a wide range of traders including both simple and complex types, in order to examine the combined impact of execution time and latency on the market. However, after obtaining the results found in Section 4.1.1, it would be intriguing to explore further into specific scenarios where different groups of traders face off against each other, such as considering only complex traders. This would shed light on the trade-off between intelligent adaptive mechanisms and execution speed, without the influence of simple traders dominating and overshadowing these interactions. Additionally, it would be interesting to observe if the significant advantage leveraged by the simple traders in a larger market would still hold true in a smaller market with fewer traders. These potential experiments could provide interesting insights into the dynamics of the market and the performance of different types of traders.

Pairwise Variable Ratio Experiments

Among the four pairwise experiments conducted, two trader pairs showed promising avenues for future research, these being SHVR vs ZIP and ZIC vs ZIP. These experiments present interesting possibilities for further exploration and investigation.

SHVR vs ZIP As previously discussed in Section 4.1.2, the introduction of latency between clients (138.3ms) had an unforeseen influence on the outcomes of the experiment. This latency advantage provided a greater opportunity for SHVR traders in closer proximity to the exchange to successfully undercut ZIP traders. However, this SHVR advantage quickly flips in favour of ZIP in high ratio ZIP markets.

Hence, it would be valuable to explore the impact of latency on the results.

One approach could involve conducting experiments using non-threaded BSE and DBSE, with two subsets of tests aimed at understanding why ZIP outperforms SHVR in a market with numerous SHVR traders in the distributed markets.

One test to explore is maintaining the same latency, distance, and client location but increasing the number of clients. This investigation aims to determine whether the dominance of ZIP traders is solely due to the parallel execution of clients. At what point do ZIP traders start to dominate, and how does the win ratio curve and its rate of change vary with the number of clients?

Another test involves keeping the number of clients fixed but varying the distance, which in turn affects the latency. This investigation seeks to explore how distance and latency affect ZIP profits and determine whether the observed changes in Figure 4.3 from BSE to DBSE and DTBSE are attributed to client-level parallelism or the influence of latency, and to what extent.

Another approach would aim to compare the results of TBSE and DTBSE, specifically focusing on the effect of latency as the only differing factor between the two simulations. By varying the client latency, the investigation seeks to determine if the trend in DTBSE starts to resemble the results observed in TBSE, particularly at smaller distances.

Please note that the latency measurements in the experiment are influenced by the choice of AWS regions and although distance can serve as a general indicator of latency, it is important to consider that variations in underlying hardware and infrastructure across different regions can also impact latency. Therefore, testing the specific AWS regions selected will be a crucial factor in determining the observed latency in the experiment. While there are several AWS regions to choose from, achieving a highly detailed resolution for varying latency might prove challenging. While unlikely, it is possible that different regions could have similar latencies, making it difficult to obtain a fine-grained distinction between them. Nonetheless, it is still possible to obtain a meaningful set of results for the experiment.

ZIC vs ZIP In Section 4.1.2, the observed trends between BSE, TBSE, DBSE, and DTBSE were not fully understood. One possible explanation is that in the market with a variable offset, the equilibrium price undergoes significant changes, requiring ZIP to adapt in order to generate profits.

In the case of BSE with a small number of ZIC traders, the sequential execution nature means that by the time one ZIP trader places an order and the next ZIP trader reacts to it, the offset has caused the market to shift slightly, necessitating ZIP to readjust to the new market conditions. The oscillating shape of the offset function, which can be observed in Figure 2.6, constantly pushes ZIP traders above and below the underlying equilibrium price.

Hence, the ZIP traders are consistently trying to catch up with a fluctuating equilibrium price. Supporting evidence for this hypothesis can be observed in the homogenous ZIP verification tests conducted in Rollin's paper[30], where ZIP takes around 4-5 orders to stabilise, while more sophisticated traders like GDX and AA stabilise after 2-3 orders. Meanwhile, ZIC generates random prices within its limit price constraint, potentially introducing additional market noise that disrupts ZIP's stabilisation process. Additionally, there is a chance that ZIC randomly matches one of the offers from ZIP that other ZIP traders are unable to fulfil. As a result, trades between ZIP traders are less likely to occur, leading to more ZIC to ZIC or ZIC to ZIP trades, both of which involve ZIC and potentially increase ZIC profits. In contrast, in TBSE, ZIP traders are not limited to a sequential order of response and order submission. This flexibility may enable them to keep pace with the real-time equilibrium price despite the variable offset.

However, this is only a hypothesis, and further investigation is needed to validate it. One potential avenue for exploration is comparing BSE to DBSE (which exhibits similar ZIP market dominance) while varying the number of DBSE clients at the same distance, using a similar approach as described in the above SHVR vs ZIP outline. By increasing the ratio of clients to serial traders per client, it may be possible to manipulate the ratio of concurrently executed traders and gain further insights into the significant difference observed from BSE to TBSE (and DBSE/DTBSE).

Furthermore, it would be valuable to investigate the peculiar spike in ZIC wins at the ZIC:ZIP ratio 2:38 in DTBSE, which exaggerates the trends observed in TBSE and DBSE at this point. However, this analysis would require a higher resolution of ratios, as the current low resolution may obscure certain fluctuations, which is a limitation imposed by the constraints of this study.

Another interesting extension of this experiment could involve varying the latency between clients, effectively modifying the setup from the large latency employed in these experiments (138.3ms) to minimal or no latency (0.8ms on the local London client) in order to create a market simulation resembling BSE. This approach would allow for the observation of changes in the win ratio resulting from increased latency, providing further insights into the impact of latency on the performance of different trading algorithms.

Bibliography

[1] Amazon. Aws global infrastructure. URL: https://aws.amazon.com/about-aws/global-infrastructure/.

[2] Antwerpen. URL: https://visit.antwerpen.be/en/info/stock-exchange.

[3] Theodore C. Bergstrom and Eugene Kwok. Extracting valuable data from classroom trading pits. *The Journal of Economic Education*, 36(3):220–235, 2005. URL: http://www.jstor.org/stable/30042656.

[4] Francis Breedon, Louisa Chen, Angelo Ranaldo, and Nicholas Vause. Staff working paper no. 711 judgement day: algorithmic trading around the swiss franc cap removal, Feb 2018.

[5] Michael Castelle, Yuval Millo, Daniel Beunza, and David C. Lubin. Where do electronic markets come from? regulation and the transformation of financial exchanges. *Economy and Society*, 45(2):166–200, 2016. doi:10.1080/03085147.2016.1213985.

[6] Dave Cliff. Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets. 01 1998.

[7] Dave Cliff. Bristol stock exchange, Mar 2021. URL: https://github.com/davecliff/BristolStockExchange.

[8] ConnectAmericas. The evolution of trade: From barter to mobile commerce. URL: https://conexionintal.iadb.org/2017/03/06/la-evolucion-del-comercio-del-trueque-al-movil/?lang=en.

[9] Ian Domowitz and Henry Yegerman. The cost of algorithmic trading: A first look at comparative performance. *The Journal of Trading*, 1:33–42, 01 2006. doi:10.3905/jot.2006.609174.

[10] Latter Schooling Edwin. *FIX 2016 EMEA Trading Conference*. 2016. URL: https://www.fca.org.uk/news/speeches/electronification-trading.

[11] London Stock Exchange. London stock exchange: Our history. URL: https://www.londonstockexchange.com/discover/lseg/our-history.

[12] Steven Gjerstad and John Dickhaut. Price formation in double auctions. *Games and Economic Behavior*, 22(1):1–29, 1998. URL: https://www.sciencedirect.com/science/article/pii/S0899825697905765, doi:https://doi.org/10.1006/game.1997.0576.

[13] Peter Gomber, Björn Arndt, Marco Lutat, and Tim Elko Uhle. High-frequency trading. *SSRN Electronic Journal*, 2011. doi:10.2139/ssrn.1858626.

[14] Terrence Hendershott and Ryan Riordan. Algorithmic trading and the market for liquidity. *Journal of Financial and Quantitative Analysis*, 48, 04 2012. doi:10.2139/ssrn.2001912.

[15] Connor Imes. Ubuntu documentation, Sep 2017. URL: https://help.ubuntu.com/community/Screen.

[16] LLC Instinet. Instinet. URL: https://rsvp.instinet.com/about-instinet/history.html.

[17] Mehdi Khosrowpour. *High Frequency Trading*. Business Science Reference, 2015.

[18] K. Kim. *Electronic and Algorithmic Trading Technology: The Complete Guide*. Complete Technology Guides for Financial Services. Elsevier Science, 2010. URL: https://books.google.co.uk/books?id=xYaW3l23h4sC.

[19] Marc Lenglet. Conflicting codes and codings: How algorithmic trading is reshaping financial regulation. *Theory, Culture & Society*, 28(6):44–66, 2011. arXiv:https://doi.org/10.1177/0263276411417444, doi:10.1177/0263276411417444.

[20] Justin Yifu Lin and Will Martin. The financial crisis and its impacts on global agriculture. *Agricultural Economics*, 41:133–144, 9 2010. doi:10.1111/j.1574-0862.2010.00495.x.

[21] Michael McGowan. The rise of computerized high frequency trading: Use and controversy. *Duke Law and Technology Review*, 16, 11 2010.

[22] Bradley Miles. Architecting and implementing a globally distributed limit order book financial exchange for research and teaching, 2019.

[23] Merton H. Miller. Financial markets and economic growth. *Journal of Applied Corporate Finance*, 11(3):8–15, 1998. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-6622.1998.tb00498.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1745-6622.1998.tb00498.x, doi:https://doi.org/10.1111/j.1745-6622.1998.tb00498.x.

[24] Brian Nigito. How to build an exchange, 2017. URL: https://www.janestreet.com/tech-talks/building-an-exchange/.

[25] OCC. Financial markets, Apr 2019. URL: https://www.occ.treas.gov/topics/supervision-and-examination/capital-markets/financial-markets/index-financial-markets.html.

[26] Marco Pagano. Financial markets and growth: An overview. *European Economic Review*, 37(2):613–622, 1993. URL: https://www.sciencedirect.com/science/article/pii/001429219390051B, doi:https://doi.org/10.1016/0014-2921(93)90051-B.

[27] Alex Preda. The sociological approach to financial markets. *Journal of Economic Surveys*, 21(3):506–533, 6 2007. doi:10.1111/j.1467-6419.2007.00512.x.

[28] Deutsche Bank Research. High-frequency trading: Reaching the limits. URL: https://www.dbresearch.de/PROD/RPS_DE-PROD/PROD0000000000454703/Research_Briefing%3A_High-frequency_trading.PDF.

[29] RialtoTrade Rialto.ai. Beginnings of algorithmic trading, Jun 2018. URL: https://medium.com/rialto-ai/beginnings-of-algorithmic-trading-19eccce902a1.

[30] Michael Rollins. Studies of response-time issues in popular trading strategies via a multi-threaded exchange simulator, 2020.

[31] SECURITIES and EXCHANGE COMMISSION. Release no. 34-59593; file no. nysealtr-2009-28, Mar 2009. URL: https://www.sec.gov/rules/sro/nysealtr/2009/34-59593.pdf.

[32] B. Mark Smith. *A history of the global stock market: From ancient rome to Silicon Valley*. The University of Chicago Press, 2004.

[33] Eric Smith, J Doyne Farmer, László Gillemot, and Supriya Krishnamurthy. Statistical theory of the continuous double auction. *Quantitative Finance*, 3(6):481–514, 2003. doi:10.1088/1469-7688/3/6/307.

[34] Richard L. Smith. *Premodern trade in world history*. Routledge, 2009.

[35] Jane Street. URL: https://www.janestreet.com/.

[36] Jane Street. Jx client disclosure, Oct 2021. URL: https://www.janestreet.com/wp-content/themes/janestreet/pdf/JX.Client.Disclosures.pdf.

[37] Gerald Tesauro and Jonathan L. Bredin. Strategic sequential bidding in auctions using dynamic programming. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, page 591–598, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/544862.544885.

[38] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, EC '01, page 206–209, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/501158.501183.

[39] Elaine Wah. How prevalent and profitable are latency arbitrage opportunities on u.s. stock exchanges? *SSRN Electronic Journal*, 2016. doi:10.2139/ssrn.2729109.

[40] Bingcheng Yan and Eric Zivot. A structural analysis of price discovery measures. *Journal of Financial Markets*, 13(1):1–19, 2010. URL: https://www.sciencedirect.com/science/article/pii/S1386418109000470, doi:https://doi.org/10.1016/j.finmar.2009.09.003.

[41] Inci Ötker Robe and Anca Maria Podpiera. The social impact of financial crises: Evidence from the global financial crisis. *Policy Research Working Papers*, 11 2013. doi:10.1596/1813-9450-6703.

Appendix A

Appendix

```
{
    "duration": 180,
    "traders": {
        "buyers": {
            "GVWY": 10,
            "ZIC": 10,
            "SHVR": 10,
            "SNPR": 10,
            "ZIP": 10,
            "AA": 10,
            "GDX": 10
        },
        "sellers": buyers
    },
    "order_schedule": {
        "demand": [
            {
                "from": 0,
                "to": 60,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            },
            {
                "from": 60,
                "to": 120,
                "ranges": [{"min": 150.0, "max": 250.0}],
                "stepmode": "fixed"
            },
            {
                "from": 120,
                "to": 180,
                "ranges": [{"min": 100.0, "max": 200.0}],
                "stepmode": "fixed"
            }
        ],
        "supply": demand
        ],
        "interval": 30,
        "timemode": "drip-poisson"
    }
}
```
Listing A.1: Simulation Configuration for Resource Usage Testing and Balanced Multi-Trader Experiment.