

https://dspace.org/

JSE Matching engine simulator

Sing, Dharmesh

2017

https://hdl.handle.net/10539/25136

Sing, Dharmesh (2017) JSE matching engine simulator, University of the Witwatersrand, Johannesburg, https://hdl.handle.net/10539/25136

Downloaded from a DSpace repository. https://dspace.org/

University of the Witwatersrand

JSE Matching Engine Simulator

Author: D. Sing 1133465
Supervisor: Prof. T. Gebbie

A dissertation submitted in fulfilment of the requirements
of the degree of Master of Science
in the
School of Computer Science and Applied Mathematics



June 11, 2017

Declaration

I declare that this dissertation is my own unaided work. It is being submitted for the degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

_____

Dharmesh Sing

_____ day of _____ 2017 in _____

Abstract

The Johannesburg Stock Exchange (JSE) started High Frequency Trading when their matching engine moved from London to Johannesburg in 2012. The study of market microstructure at the JSE is not possible without access to their matching engine. This dissertation investigates the challenges of studying market microstructure and describes the design and implementation of an open source matching engine. CoinTossX was developed as an open source low latency high throughput stock exchange. The software was developed in Java and used open source libraries. The software is tested using an 8-variate mutually-exciting Hawkes process to govern the times of coupled liquidity demand and supply events, while trade and quote prices and volumes are generated consistent with the event type. The testing showed that CoinTossX is able to support multiple clients, stocks and matching algorithms.

Dedication

This masters is dedicated to my significant other, Melisha.

Acknowledgements

I would like to thank my supervisor Professor T. Gebbie for his guidance, advice and his ability to accommodate my schedule during my masters. Also many thanks to Professor T. Celik for initially contributing as a co-supervisor at the start of the project. A special thanks to Dieter Hendricks, Diane Wilcox and the rest of the QuERI lab team.

I also would like to thank my parents who made the sacrifices to ensure I could achieve my goals.

Contents

1 Introduction **19**

1.1 Stock Exchange 19
1.2 Limit Order Book 19
1.3 Aim 20
1.4 Delimitations 20
1.5 Limitations 20
1.6 Research Methodology 20
1.7 Dissertation Outline 21

2 Literature Review **22**

2.1 Introduction 22
2.2 What Is Market Microstructure ? 22
2.3 Challenges In Studying Limit Order Books 22
2.3.1 Rational Traders VS Zero-Intelligence Traders 22
2.3.2 Order Flows 22
2.3.3 Feedback And Coupling 23
2.3.4 Priority 23
2.3.5 Ice-berg and hidden orders 24
2.3.6 Dark Pools 25
2.3.7 Volatility 25
2.3.8 Resolution parameters 25
2.3.9 Bilateral trade agreements 25
2.3.10 Opening and closing auctions 25
2.4 Matching Engine 26
2.5 High Frequency Trading (HFT) 27
2.6 Regulation Impact In U.S And Europe 28
2.6.1 The Limit Order Display Rule 28
2.6.2 Regulation National Market System (Reg NMS) 29
2.6.3 Markets in Financial Instruments Directive (MiFID) 30
2.7 Technology Impact 30
2.7.1 Hardware 30
2.7.2 Message Protocols 31
2.8 Johannesburg Stock Exchange 32
2.9 Conclusion 33

3 Requirements **34**

3.1 High Level Overview 34
3.2 Scope 34
3.3 Components 34
3.3.1 Clients 34
3.3.2 The Trading Gateway 34
3.3.3 Matching Engine 35
3.3.4 The Market Data Gateway 35
3.3.5 The Website 35
3.4 Functional Requirements - Trading Gateway 35
3.4.1 Login 35
3.4.2 Logout 36
3.4.3 Order Request 36
3.4.4 Change Trading Session 36
3.5 Functional Requirements - Matching Engine 36
3.5.1 Trading Sessions 37
3.5.2 Order Types 37
3.5.3 Time In Force (TIF) 39
3.5.4 Valid Combinations 41
3.5.5 Price-Time Priority Algorithm 42
3.5.6 Passive Price Improvement Algorithm 48
3.5.7 Filter and Uncross Algorithm 48
3.5.8 Auction Algorithm 54
3.5.9 Static And Dynamic Price 56
3.6 Functional Requirements - Market Data Gateway 57
3.6.1 Market Data Update 57
3.7 Functional Requirements - Website 57
3.7.1 Manage clients 57
3.7.2 Manage stocks 58
3.7.3 View Limit Order Books 58
3.7.4 Configure Testing Framework 58
3.7.5 Run Testing Framework 58
3.8 Deployment Requirements 58

4 Design **59**
4.1 Introduction . . . 59
4.2 Architecture Overview . . . 59
4.3 Implementation Language . . . 60
4.3.1 Performance . . . 60
4.3.2 Portability . . . 61
4.3.3 Development time . . . 61
4.3.4 Open source libraries . . . 61
4.4 Message Protocol . . . 61
4.5 Communication . . . 62
4.5.1 TCP . . . 62
4.5.2 UDP . . . 63
4.5.3 JSE Communication . . . 63
4.5.4 ZeroMQ . . . 63
4.5.5 Aeron . . . 64
4.6 Limit Order Book Data Structure . . . 66
4.6.1 Execution and Storage Requirements . . . 67
4.6.2 Efficient use of CPU cache . . . 68
4.6.3 Memory Access Patterns . . . 71
4.6.4 Data Structure . . . 71
4.7 Business Rules . . . 74
4.8 Website . . . 74
4.9 Web Event Listener . . . 75
4.10 Conclusion . . . 76

5 CoinTossX **77**
5.1 Introduction . . . 77
5.2 Technical Documentation . . . 77
5.3 Website Screenshots . . . 77
5.3.1 Splash Screen . . . 77
5.3.2 Stocks . . . 78
5.3.3 Clients . . . 79
5.3.4 Limit Order Book . . . 79
5.3.5 Hawkes Configuration . . . 80
5.3.6 Run Simulation . . . 80
5.4 Conclusion . . . 81

6 Testing Methodology **82**
6.1 Introduction . . . 82
6.2 Hawkes Process . . . 82
6.3 Hawkes Process In Financial Modeling . . . 83
6.3.1 Modeling A Stream Of Orders . . . 84
6.4 Simulating A Hawkes Process . . . 86
6.5 Conclusion . . . 87

7 Hawkes Simulation **88**
7.1 Hawkes Algorithm . . . 88
7.2 Algorithm for Price and Volume Generator . . . 89
7.3 Initialization . . . 89
7.4 General Routine - Client . . . 90
7.5 General Routine - Client MarketData Subscriber . . . 90
7.6 VWAP . . . 90
7.7 Aggressive Buy Trade (Type 1) . . . 91
7.8 Aggressive Sell Trade (Type 2) . . . 91
7.9 Aggressive Buy Quotes (Type 3) . . . 92
7.10 Aggressive Sell Quotes (Type 4) . . . 93
7.11 Passive Buy Trade (Type 5) . . . 93
7.12 Passive Sell Trade (Type 6) . . . 94
7.13 Passive Buy Quotes (Type 7) . . . 94
7.14 Passive Sell Quotes (Type 8) . . . 95
7.15 Intensity Charts . . . 96

8 Test Analysis **98**
8.1 Introduction . . . 98
8.2 Unit Tests . . . 98
8.3 Java Microbenchmark Harness (JMH) . . . 98
8.4 Performance Tests . . . 98
8.4.1 Scenario A: Throughput Testing . . . 99
8.4.2 Scenario A: Latency Testing . . . 99
8.4.3 Scenario A: Limit Order Book Storage Testing . . . 100
8.4.4 Scenario B: Throughput Testing . . . 102
8.4.5 Scenario B: Latency Testing . . . 102
8.5 Analysis . . . 104

9 Conclusion **106**

9.1 Discussion 106

9.2 Further Work 106

A Appendix **115**

A.1 Java vs Google Protocol Buffers Test 115

A.2 Aeron Performance Tests 115

A.2.1 Aeron ThroughPut Performance Test 115

A.2.2 Aeron Latency Performance Test 117

B Appendix **118**

B.1 Matching Engine Test Cases 118

B.1.1 Market Order Test Case - Test 1 119

B.1.2 Market Order Test Case - Test 2 120

B.1.3 Market Order Test Case - Test 3 121

B.1.4 Market Order Test Case - Test 4 122

B.1.5 Market Order Test Case - Test 5 123

B.1.6 Market Order Test Case - Test 6 124

B.1.7 Limit Order Test Case - Test 1 125

B.1.8 Limit Order Test Case - Test 2 126

B.1.9 Limit Order Test Case - Test 3 127

B.1.10 Limit Order Test Case - Test 4 128

B.1.11 Limit Order Test Case - Test 5 129

B.1.12 Limit Order Test Case - Test 6 130

B.1.13 Limit Order Test Case - Test 7 131

B.1.14 Limit Order Test Case - Test 8 132

B.1.15 Limit Order Test Case - Test 9 133

B.1.16 Limit Order Test Case - Test 10 134

B.1.17 Limit Order Test Case - Test 11 135

B.1.18 Limit Order Test Case - Test 12 136

B.1.19 Hidden Order Test Case - Test 1 137

B.1.20 Hidden Order Test Case - Test 2 138

B.1.21 Hidden Order Test Case - Test 3 139

B.1.22 Hidden Order Test Case - Test 4 140
B.1.23 Hidden Order Test Case - Test 5 141
B.1.24 Hidden Order Test Case - Test 6 142
B.1.25 Hidden Order Test Case - Test 7 143
B.1.26 Hidden Order Test Case - Test 8 144
B.1.27 Hidden Order Test Case - Test 9 145
B.1.28 Hidden Order Test Case - Test 10 146
B.1.29 Hidden Order Test Case - Test 11 147
B.1.30 Hidden Order Test Case - Test 12 148
B.1.31 Hidden Order Test Case - Test 13 149
B.1.32 Hidden Order Test Case - Test 14 150
B.1.33 Hidden Order Test Case - Test 15 151
B.1.34 Hidden Order Test Case - Test 16 152
B.1.35 Hidden Order Test Case - Test 17 153
B.1.36 Stop Order Test Case - Test 1 154
B.1.37 Stop Order Test Case - Test 2 155
B.1.38 Stop Order Test Case - Test 3 156
B.1.39 Stop Order Test Case - Test 4 157
B.1.40 Stop Order Test Case - Test 5 158
B.1.41 Stop Order Test Case - Test 6 159
B.1.42 Stop Order Test Case - Test 7 160
B.1.43 Stop Order Test Case - Test 8 161
B.1.44 Stop Order Test Case - Test 9 162
B.1.45 Stop Order Test Case - Test 10 163
B.1.46 Stop Order Test Case - Test 11 164
B.1.47 Stop Order Test Case - Test 12 165
B.1.48 Stop Limit Order Test Case - Test 1 166
B.1.49 Stop Limit Order Test Case - Test 2 167
B.1.50 Stop Limit Order Test Case - Test 3 168
B.1.51 Stop Limit Order Test Case - Test 4 169
B.1.52 Stop Limit Order Test Case - Test 5 170
B.1.53 Stop Limit Order Test Case - Test 6 171

B.1.54 Stop Limit Order Test Case - Test 7 172
B.1.55 Stop Limit Order Test Case - Test 8 173
B.1.56 Stop Limit Order Test Case - Test 9 174
B.1.57 Stop Limit Order Test Case - Test 10 175
B.1.58 Stop Limit Order Test Case - Test 11 176
B.1.59 Stop Limit Order Test Case - Test 12 177
B.1.60 Filter And Uncross Test Case - Test 1 178
B.1.61 Filter And Uncross Test Case - Test 2 179
B.1.62 Filter And Uncross Test Case - Test 3 180
B.1.63 Filter And Uncross Test Case - Test 4 181
B.1.64 Auction Test Case - Test 1 182
B.1.65 Auction Test Case - Test 2 183
B.1.66 Auction Test Case - Test 3 184
B.1.67 Cancel Order Test Case - Test 1 185
B.1.68 Cancel Order Test Case - Test 2 186
B.1.69 Cancel Order Test Case - Test 3 187
B.1.70 Cancel Order Test Case - Test 4 188
B.1.71 Cancel Order Test Case - Test 5 189
B.1.72 Cancel Order Test Case - Test 6 190
B.1.73 Cancel Order Test Case - Test 7 191
B.1.74 Cancel Order Test Case - Test 8 192
B.1.75 Cancel Order Test Case - Test 9 193
B.1.76 Cancel Order Test Case - Test 10 194
B.1.77 Replace Order Test Case - Test 1 195
B.1.78 Replace Order Test Case - Test 2 196
B.1.79 Replace Order Test Case - Test 3 197
B.1.80 Replace Order Test Case - Test 4 198
B.1.81 Replace Order Test Case - Test 5 199
B.1.82 Replace Order Test Case - Test 6 200
B.1.83 Replace Order Test Case - Test 7 201
B.1.84 Replace Order Test Case - Test 8 202
B.1.85 Replace Order Test Case - Test 9 203

B.1.86 Replace Order Test Case - Test 10 204
B.1.87 Replace Order Test Case - Test 11 205
B.1.88 Replace Order Test Case - Test 12 206
B.1.89 Replace Order Test Case - Test 13 207
B.1.90 Replace Order Test Case - Test 14 208
B.1.91 Replace Order Test Case - Test 15 209
B.1.92 Replace Order Test Case - Test 16 210
B.1.93 Replace Order Test Case - Test 17 211
B.1.94 Replace Order Test Case - Test 18 212
B.1.95 Replace Order Test Case - Test 19 213
B.1.96 Replace Order Test Case - Test 20 214
B.1.97 Replace Order Test Case - Test 21 215
B.1.98 Replace Order Test Case - Test 22 216
B.1.99 Replace Order Test Case - Test 23 217
B.1.100Replace Order Test Case - Test 24 218
B.1.101Replace Order Test Case - Test 25 219
B.1.102Replace Order Test Case - Test 26 220
B.1.103Replace Order Test Case - Test 27 221
B.1.104Replace Order Test Case - Test 28 222
B.1.105Replace Order Test Case - Test 29 223
B.1.106Replace Order Test Case - Test 30 224
B.1.107Replace Order Test Case - Test 31 225
B.1.108Replace Order Test Case - Test 32 226
B.1.109Replace Order Test Case - Test 33 227
B.1.110Replace Order Test Case - Test 34 228
B.1.111Replace Order Test Case - Test 35 229
B.1.112Replace Order Test Case - Test 36 230

C Appendix **231**

C.1 Getting Started Guide 231
C.1.1 Download Java 231
C.1.2 Clone the repository 231
C.1.3 Build the project 231

C.1.4 Location of configuration files 231
C.1.5 Location of connection files 231
C.1.6 Deploy 232
C.1.7 Start the Gateways on the server 232
C.1.8 Simple client 232
C.1.9 Send a new order to the Trading Gateway 233

List of Tables

1 Order book before matching 23
2 Order book after matching 23
3 Pro-Rata order book before matching 23
4 Pro-Rata order book after matching 24
5 Java vs Google Protocol Buffers 61
6 SBE vs GPB [82] 62
7 Scenario A: Throughput 99
8 Scenario B: Intraday Auction Throughput 102
9 Java Test 115
10 Google Protocol Buffer Test 115

List of Figures

1 HFT vs AT [26] . . . 27
2 HFT/AT Workflow [15] . . . 28
3 Order Spread Impact [91] . . . 29
4 Trading Gateway Use Case . . . 35
5 Matching Engine Use Case . . . 36
6 Trading Sessions [39] . . . 37
7 TIF Order Type Valid Combinations [39] . . . 42
8 Trading Session TIF Order Type Valid Combinations [39] . . . 42
9 Market Data Gateway Use Case . . . 57
10 Website Use Case . . . 57
11 High Level Architecture Diagram . . . 59
12 Cern MOM Libraries . . . 64
13 Aeron Architecture [81] . . . 65
14 Aeron Latency Test . . . 66
15 JSE orders [34] . . . 67
16 Snapshot of Apple LOB on NASDAQ at 8:43[12] . . . 68
17 CPU cache . . . 69
18 Data Structure . . . 73
19 Order List . . . 74
20 Business Rules . . . 74
21 Splash Screen 1 . . . 77
22 Splash Screen 2 . . . 78
23 Splash Screen 3 . . . 78
24 Stocks . . . 79
25 Clients . . . 79
26 Limit Order Book . . . 80
27 Hawkes Configuration . . . 80
28 Hawkes Simulation . . . 81
29 Intensity Chart Type 1 - 4 . . . 96
30 Intensity Chart Type 5 - 8 . . . 97
31 Scenario A: Latency . . . 100

32 Scenario A: Volume of Orders 101
33 Scenario A: Count of Orders 101
34 Scenario A: Trades Executed Count 102
35 Scenario B: Latency 103
36 Scenario B: Trades Executed Count 104

List of Algorithms

1 Pro-Rata Algorithm 24
2 Passive Price Improvement Algorithm 48
3 Filter Algorithm (Hill Climber Algorithm) - Step 1 49
4 Filter Algorithm (Hill Climber Algorithm) - Step 2 49
5 Filter Algorithm (Hill Climber Algorithm) - Step 3 50
6 Filter Algorithm (Hill Climber Algorithm) - Step 4 50
7 Filter Algorithm (Hill Climber Algorithm) - Step 5 51
8 Uncrossing Algorithm - Calculate the target price 51
9 Uncrossing Algorithm - Calculate maximum executable volumes at each price level in the crossed region 51
10 Uncrossing Algorithm - Calculate the target trade price that maximizes the executable volume 52
11 Uncrossing Algorithm - Agress Order Book 52
12 Auction Algorithm 54
13 Step 1 - Initialization 86
14 Step 2 - First event 86
15 Step 3 - General routine 87
16 Step 4 - Output 87
17 General Routine - Client 90
18 General Routine - Client MarketData Subscriber 90
19 Volume Weighted Average Price (VWAP) 90
20 Aggressive Buy Trade (Type 1) 91
21 Aggressive Sell Trade (Type 2) 91
22 Aggressive Buy Quotes (Type 3) 92
23 Aggressive Sell Quotes (Type 4) 93
24 Passive Buy Trade (Type 5) 93
25 Passive Sell Trade (Type 6) 94
26 Passive Buy Quotes (Type 7) 94
27 Passive Sell Quotes (Type 8) 95

1 Introduction

A high frequency low latency, high throughput, open research oriented matching engine simulator for the Johannesburg Stock Exchange (JSE) does not exist. High Frequency Trading (HFT) at the JSE was not possible until 2012 when the JSE moved it's matching engine from London to Johannesburg [38]. Roundtrip latency has reduced from 300 milliseconds to 2400 microseconds and when a company moves it's servers to the co-location site, latency can reduce to 50 microseconds [40]. The JSE has a propriety test environment, but this is only targeted at business users and has not been made available for academic research.

This research investigates and discusses the challenges in designing, implementing and testing a low latency, high throughput matching engine. The key research output of the research project was the software system, the test harness (framework) and it's access via the CoinTossX web interfaces. This dissertation documents and discusses these research outputs.

1.1 Stock Exchange

Stock exchanges allow traders to come together to trade securities. Buyers and sellers find each other when they want to trade on a specific security, price, quantity and urgency[58]. There are 2 types of markets: a dealer market and a limit order market [58]. A dealer market uses dealers to speed up the search process by buying or selling to traders. They quote a bid (buy) and an ask (sell) price for a security. The spread is the difference between the bid and the ask price. Dealers make a profit from the spread in exchange for providing liquidity to the market. Traders can only buy or sell at the prices published by the dealers. A limit order market uses limit order books for the search process. This market does not use dealers and this allows traders to submit buy and sell orders at their desired price.

1.2 Limit Order Book

A limit order market maintains a limit order book (LOB) for each security at the exchange [1]. When a new order is received a matching engine is used to match the orders and create the trade. If the order is not matched it will remain active in the limit order book. The order will be removed if it is matched with another incoming order or the order is canceled[47]. Gould *et al.* [28] defines a LOB*(t)*, as the set of all active orders in a market at time *t*. The bid price at time *t,* bid*(t)* is the highest active buy order price and the ask price at time *t,* ask*(t)* is the lowest active sell order price. The mid price at time *t,* mid*(t)* is $[bid(t) + ask(t)]/2$. Orders to buy or sell can be classified as limit orders or market orders [28]. A market order does not specify a price an executes immediately matching against the best available price. A limit order specifies a price and matches against orders better than or equal to the specified price.

1.3 Aim

The aim of this research is to produce an open source high frequency matching engine. The software has been designed to be reconfigurable to allow for different market structures, matching algorithms and software components. It can be configured for multiple clients, stocks and trading sessions *e.g.* the researcher has the ability to simulate rolling 30 second auctions. The software has been configured to replicate the rules and processes of the JSE. It will allow traders, organizations and academic institutions to test market structure, fragility and dynamics without financial loss. The software will provide a platform to study price formations in stock exchanges and the interplay between regulators, market structure and dynamics.

1.4 Delimitations

The functionality of the software was tested using the test cases made available online from the JSE. The requirements were taken from [39]. The JSE was not contacted for more information. The software only implements one Trading and Market Data Gateway. Testing of the trading sessions was restricted to the Continuous and Intraday Auction trading sessions. Time In Force testing was restricted to DAY orders.

1.5 Limitations

The matching engine was evaluated using unit tests and the Hawkes testing framework. The outputs of these tests were not compared to the JSE or another exchange as the data is not available by the industry. The JSE's test environment is also closed and does not provide realistic order-book dynamics. This research addresses this by providing a framework that can be compared to recorded transaction data arising from the actual market systems using the Hawkes processes.

1.6 Research Methodology

The design and creation research methodology approach was used for this dissertation. The matching engine, trading gateway, market data gateway, web console and a testing framework were the artifacts produced. The software was design and developed using test-driven development (TDD). TDD involves 3 steps[5]:

1. Write a unit test
2. Code the functionality
3. Refactor the design and code.

This method was used as it allowed the code to change and become more efficient while ensuring the implemented functionality continued to work. A Hawkes simulation was implemented to send a large number of orders to the exchange to evaluate the software.

1.7 Dissertation Outline

The rest of this dissertation is structured as follows:

Chapter 2 gives the literature review to my work. This includes the description of market microstructure, the challenges of studying LOBs and the impact that regulation and technology has had on stock exchanges and matching engines.

Chapter 3 presents the requirements for the software. It describes the matching logic and algorithms required.

Chapter 4 explains the design decisions taken to implement the requirements from Chapter 3. Each library and technology used is discussed with it's test cases showing the reasons why it was selected.

Chapter 5 presents CoinTossX.

Chapter 6 gives an overview of the Hawkes model..

Chapter 7 presents the Hawkes simulation that was used to test the software.

Chapter 8 discusses the testing scenarios and analyzes the results of the Hawkes simulation.

Chapter 9 is the conclusion.

2 Literature Review

2.1 Introduction

Designing a matching engine requires understanding of how the broader context influences the design and implementation of the software. There are various studies related to market microstructure, algorithmic trading and price formation. The goal of this chapter is to review the literature of the study of stock exchanges and how the market, regulation and technology impact the design of matching engines. For a more detailed discussion refer to [13, 66, 30].

2.2 What Is Market Microstructure ?

O'Hara [66] defines Market Microstructure as the study of the process and outcomes of exchanging assets under explicit trading rules. Madhavan [54]defines it as the area of finance that is concerned with the process by which investors' latent demands are ultimately translated into transactions. It provides a deeper understanding of how prices are formed and change due to trading strategies, trading volumes, the rules of an exchange and other factors. The rest of this section will describe these factors.

2.3 Challenges In Studying Limit Order Books

LOBs are difficult to study as there are various factors that influence the state of the LOB. The remainder of this section will discuss the challenges in studying LOBs.

2.3.1 Rational Traders VS Zero-Intelligence Traders

Traders that place orders on current information that they have are referred to as rational traders. Traders that place orders based on the current order flow and historical data are referred to as zero-intelligent traders [23]. Traders can use either of these approaches or a combination to determine how they place their orders. The state of the LOB will be affected based on the strategy the trader uses[28].

2.3.2 Order Flows

Order flows depend on the LOB at time t and on recent order flows. Rational traders create their order flows based on the changing market conditions while zero-intelligent traders create their orders based on the LOB(t) and recent history. More understanding is required to determine what information traders use to make decisions or to quantify the conditional structure of order flows. This is difficult to do as the number of variables are large [28, 9, 13, 30].

2.3.3 Feedback And Coupling

The LOB(t) is based on trader's actions and trader's actions are based on the LOB(t). This strong coupling makes modeling a LOB difficult [28].

2.3.4 Priority

In a LOB there can be more than one order at a particular price. A priority is used when determining the order to use when performing the matching. The 3 most popular algorithms are: price-time, pro-rata and price-size.

2.3.4.1 *Price-time priority*

The price-time algorithm is the most commonly used. In tables 1 and 2, a buy trade B1 matches sell trades S1 and S2. Priority is given to the sell trade with the earliest submission time [28]. Therefore B1 will match S1 and will be removed from the LOB. This algorithm forces traders to submit limit orders.

Timestamp	Buy/Sell	Order	Quantity	Price
8:00	**Sell**	**S1**	**100**	**90.01**
8:10	Sell	S2	100	90.01
8:20	**Buy**	**B1**	**100**	**90.01**

Table 1: Order book before matching

Timestamp	Buy/Sell	Order	Quantity	Price
8:10	Sell	S2	100	90.01

Table 2: Order book after matching

2.3.4.2 *Pro-rata priority*

The pro-rata algorithm divides the incoming order B1 between the matching orders S1 and S2. In tables 3 and 4, both the sell trades are reduced in quantity. Orders S1 and S2 will fill a portion of order B1.

Timestamp	Buy/Sell	Order	Quantity	Price
8:00	Sell	S1	100	90.01
8:10	Sell	S2	400	90.01
8:20	Buy	B1	300	90.01

Table 3: Pro-Rata order book before matching

Algorithm 1 Pro-Rata Algorithm

1: $S1 = (100/500) * 300 = 60$
2: $S2 = (400/500) * 300 = 240$

Timestamp	Buy/Sell	Order	Quantity	Price
8:00	Sell	S1	60	90.01
8:10	Sell	S2	240	90.01

Table 4: Pro-Rata order book after matching

2.3.4.3 *Price-size priority*

The price-size priority algorithm chooses between the matching orders by selecting the active order of the largest size.

The priority algorithms change the way traders behave. Traders will submit orders earlier if a price-time priority algorithm is used. Price-size and pro-rata priority will encourage traders to place large limit orders. These large orders will provide greater liquidity to the market [28].

2.3.5 Ice-berg and hidden orders

Ice-berg orders allow traders to submit a limit order with a portion of the order seen by the market and the rest of the order is hidden from the market. The portion of the order seen by the market is called the visible size of the order[28]. The rules for these types of orders vary between exchanges. Examples include:

- If the visible size of the order matches an incoming order, then another quantity equal to the visible size becomes visible. The visible size has the same priority as non ice-berg orders.
- If the visible size of the order matches an incoming order, then the rest of the order is cancelled. This allows iceberg orders to match incoming orders of a very large size.

Some exchanges allow the order to be entirely hidden from the market. These orders are called hidden orders. Hidden orders and the hidden portion of ice-berg orders have a lower priority than all visible active orders. These orders allow traders to hide their intentions from the market.

2.3.6 Dark Pools

Limit order books which hide all active orders are called dark pools. Some dark pools match orders at the midpoint price of the asset using another non-dark pool LOB. Other dark pools execute orders by their price and time priority. Exchanges can implement dark pools in different ways [93, 28].

2.3.7 Volatility

Volatility is a measure of the variation of price of a stock. Stocks with higher volatility have larger price changes compared to stocks with lower volatility. Traders use volatility to determine the risk associated with a stock. They manage their risk exposure by using volatility to select the instruments when constructing their portfolios [28]. Studies have shown the link between LOBs and volatility by taking the bid(t), ask(t) and the mid(t). Traders search for hidden liquidity in the market by submitting orders and then immediately canceling the order. The bid(t) and ask(t) fluctuate due to this and affects the volatility of the LOB. Measuring volatility of a LOB is affected by these price movements.

2.3.8 Resolution parameters

A trader cannot place an order that is smaller than the LOBs tick size. The LOBs tick size control's the smallest order size. Traders that wish to submit large orders, break up the orders into small chunks to minimize the impact on the market. The tick size impacts these traders [28].

2.3.9 Bilateral trade agreements

Traders can block other traders from trading with them by maintaining a block list at the exchange. Trader A cannot trade with Trader B if A is on B's block list. Trades can only execute if the traders have a bilateral trade agreement between them. The exchange will send an update to each trader of their personal LOBs that contain only the active orders owned by traders whom they are allowed to trade with. These personal LOBs affect how market orders are executed. If trader A submits a market order, it will try to match orders in trader's A personal LOB and not consider orders from traders on their block list. The orders from blocked traders could have a higher priority but will be ignored in the matching process. Exchanges with bilateral trade agreements can create markets with negative bid-ask spreads. Modeling of a LOB becomes very difficult due to these factors [28].

2.3.10 Opening and closing auctions

An exchange has opening, continuous and closing sessions [66]. Limit and markets orders submitted during a continuous trading session are matched immediately if

possible. The exchange uses call auctions to match orders at the opening and closing sessions. Limit and market orders submitted during a call auction are not matched immediately. These orders are matched at the end of the call auction at a single price using a price-discovery process. This process finds the price that will match the most number or buy and sell orders. During the call auctions, traders can see the value of the price.

2.4 Matching Engine

A matching engine is a component of an exchange that matches buy and sell orders according to the rules of the exchange [27]. Nair[59] prototyped a simple matching engine using a single stock. This work provided a simplified framework demonstrating the over-all principle of coupling a matching engine with agent based modeling (ABM) of a stock market. This research is aimed at building a realistic matching engine that can be used for large scale ABM simulation work as part of separate future projects. The construction of the ABM components is outside of the scope of this project given the complexity of first resolving the design constraints of a low latency, high throughput matching engine for large scale market simulation.

The LMAX Exchange (London Multi Asset Exchange) is an FX exchange with a ultra low latency matching engine. They are the only exchange that is known to release their matching engine design. LMAX also used the following rules for high performance computing [22]:

1. Have good mechanical sympathy. Martin Thompson used this term to mean that a software developer should take the hardware into consideration when building software [21].
2. Keep the working data in memory.
3. Write cache friendly code.
4. Write clean compact code.
5. Spend time to model your domain.
6. Do concurrency using the correct techniques.

The LMAX Gateways were not able to process events fast enough as the events were received. The events needed to be stored until it could be processed. The events could be stored in a queue which uses an array or a linked list. The problem with this approach is that the linked-list could grow and increase the garbage in the system and the array would need to be checked for its size and resized when required. LMAX solved this problem by creating the Disruptor for inter-thread concurrency. The Disruptor is a ring buffer that follows the design principles listed above. It preallocates memory which is shared with the consumer. The system was developed on the JVM and can process 6 million orders per second on a single thread [85].

"The Disruptor has significantly less write contention, a lower concurrency overhead and is more cache friendly than comparable approaches, all of which results in greater throughput with less jitter at lower latency. On processors at moderate clock rates we have seen over 25 million messages per second and latencies lower than 50 nanoseconds. This performance is a significant improvement compared to any other implementation that we have seen. This is very close to the theoretical limit of a modern processor to exchange data between cores" [85]

2.5 High Frequency Trading (HFT)

Orders are placed manually by a human at a computer terminal or by computer algorithms. Algorithmic trading (AT) is computer algorithms that generate orders without any human intervention [26]. HFT is very similar to algorithmic trading but there are features that are distinct to HFT. Gomber *et al.* [26] defines the similarities and differences between HFT and AT in figure 1.

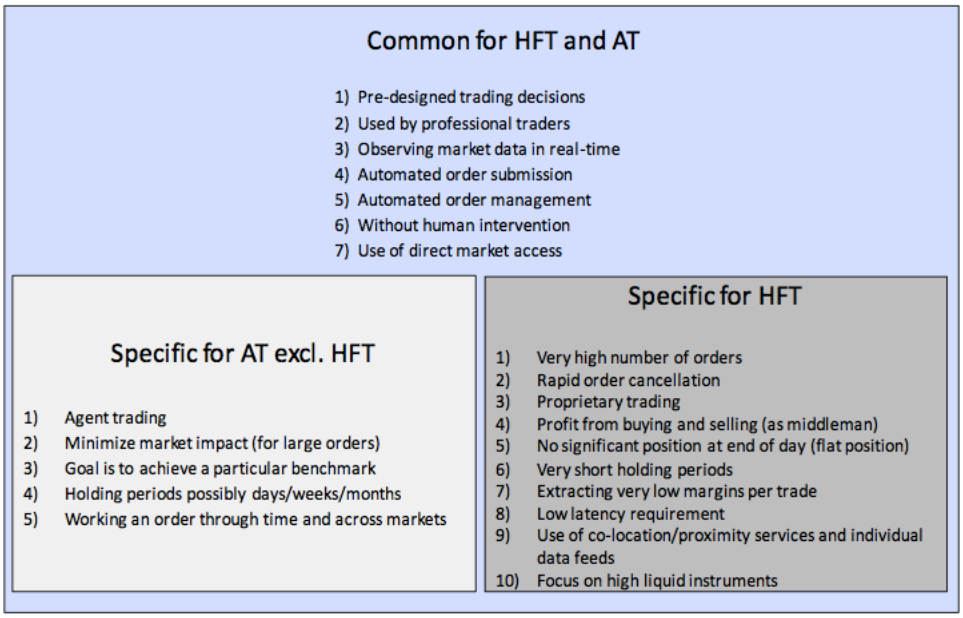


Figure 1: HFT vs AT [26]

The U.S Commodity Futures Trading Commission [15] defines a workflow to determine if the computer software is HFT or AT.

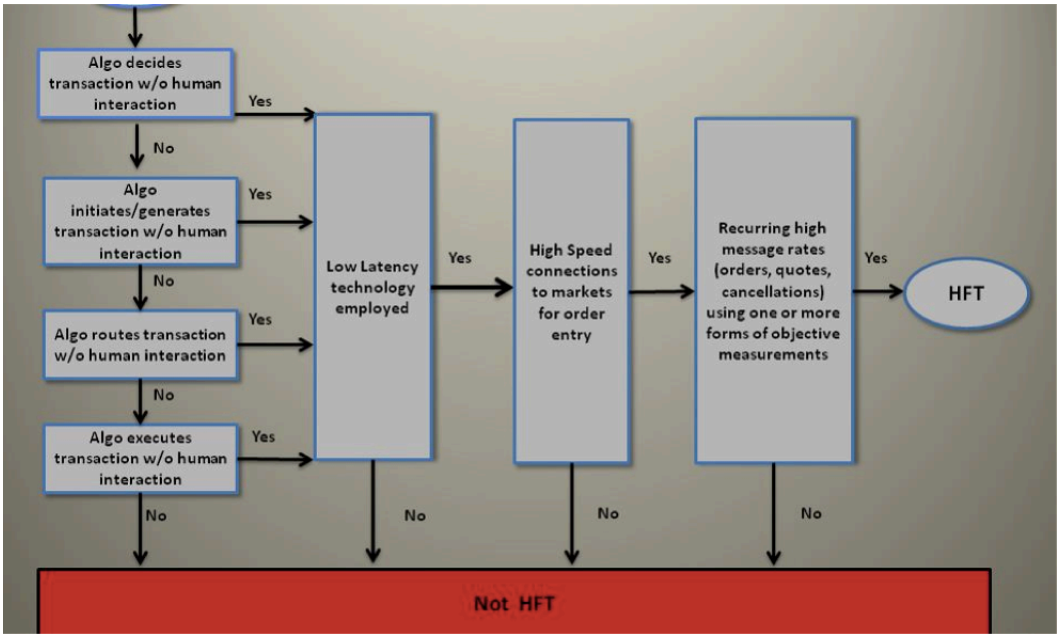


Figure 2: HFT/AT Workflow [15]

2.6 Regulation Impact In U.S And Europe

2.6.1 The Limit Order Display Rule

The U.S. Securities and Exchange Commission (SEC) has significantly impacted how the trading systems are implemented. In 1997, the SEC changed the Order Handling Rules [73]. Market Makers quote bid and ask prices to provide liquidity to the market. They had to display all their outstanding limit orders when the SEC introduced the Limit Order Display Rule.

A venue outside the public exchange was formed to trade large orders. If large orders are placed on the exchange this will have a significant impact on the market. Therefore, these alternative trading venues, also referred to as dark pools were formed. You cannot see the orders in the dark pool. Only when orders are matched, the trades are displayed publicly.

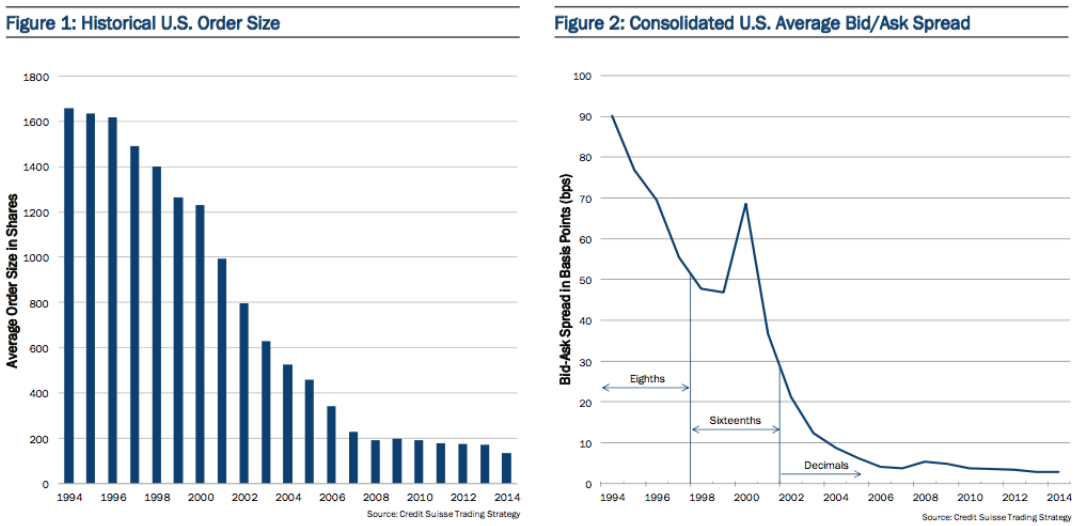


Figure 3: Order Spread Impact [91]

Figure 3 shows how after 1997 the order sizes decrease and the spread between bids and offers also decreased. This was because traders did not want to display large orders and to be competitive they had to use small orders with narrower spreads. This type of trading is better suited to computers than humans.

2.6.2 Regulation National Market System (Reg NMS)

Reg NMS was introduced in 2005 in the U.S. The Trade-Through Rule, The Access Rule, The Sub-Penny Pricing Rule and The Market Data Rules and Plans Rule were implemented to assist the investors, but it had a greater impact on computer based trading [73].

- *Trade-Through Rule* enforced that the best bids and offers be displayed and a trader must trade using the best execution price. Trading systems across all trading venues have to cancel orders that did not meet the best offer rule. This has increased the use of computer systems across the exchanges.
- *Access Rule* stopped exchanges from executing trades that did not match the best execution price at another exchange. It required the use of private linkages to access quotes and set a limit on the access fees. Computer systems needed to send more data between exchanges to update their quotes.
- *Sub-Penny Rule* stopped traders from gaining an advantage in priority by changing the order by an insignificant amount *i.e.* less than a penny. This rule did not apply to stocks less than a penny. This rule did not have a significant impact on trading systems.
- *Market Data Rule and Plans Rule* changed the formula to allocated revenue generated from market data feeds. Market Data feed revenue would go to

the market centers that provide the best prices and largest orders. The rule allowed market centers to provide their own quotes without fees. The aim of the rule was to stop wash sales and shredding manipulation. A wash sale is to sell a security at a loss and buy it back to create revenue from the reports. Shredding breaks a large transaction into smaller ones to generate revenue from market data reports. This rule increased the use of network activity amongst the trading venues.

2.6.3 Markets in Financial Instruments Directive (MiFID)

Europe has done similar work and implemented MiFID to stop market manipulation. MiFID I was put into effect in 2007 and its purpose was to improve investor protection, increase competition, and assist in the creation of a single market for financial services in the European Union. However the benefits did not flow to all market participants and the benefits did not pass on to the investor. MiFID II was established to regulate the market, make it more transparent and pay more attention to over the counter products [89].

Reg NMS and MiFID I/II placed a greater demand on trading systems and network activity. Systems needed to scale to handle the high volumes of data and analyze the data. These rules affected software and hardware design.

2.7 Technology Impact

The markets have changed over time and technology has had a major role in these changes. Computing power and software evolved; the need to reduce transaction costs increased has led to important changes in market structure and market architecture [4]. These changes have supported the rise of electronic trading and high frequency trading.

2.7.1 Hardware

Brokerage firms combine hardware and software to get as close to zero latency as possible. The speed of light is seen as a constraint to traders. A fiber optic network cable introduces 4.9 microseconds per kilometer [7]. Firms use co-location to reduce the distance and latency between their servers and the exchange. Traders optimize the hardware architecture and use different technologies to reduce the latency.

2.7.1.1 Field Programmable Gate Arrays (FPGA)

FPGAs were developed for the defense and telecommunications industry. The chip contains programmable logic blocks and interconnects that can be configured using a hardware definition language. This allows the chip to perform functions that would have been done in software [67]. FPGAs are used to process data as it arrives or to execute algorithms. These chips are faster than CPUs as you can program the chip to perform the exact function you want and the number of processing units are higher than on a CPU. FPGAs have a high throughput and low latency, however they are very difficult to program. This hardware is suitable for message gateways.

2.7.1.2 Graphical Processing Units (GPU)

GPUs were originally developed for game rendering but are now used in various other sectors such as high frequency trading. A CPU has only a few cores while a GPU has 100s of processing cores. This allows it to outperform a CPU in parallel programming. A GPU is optimized for high throughput and is suited for large scale offline calculations [62].

2.7.1.3 InfiniBand Architecture

HFT applications generally use a cluster of servers to perform computations. These servers need to pass data to each other with low latency. The standard TCP/IP protocol does not allow this as it has various layers of code that slow down the data transfer. The InfiBand Architecture was created to transfer data using a low latency link between the servers [29, 57].

2.7.2 Message Protocols

The financial industry has standardized communication between market participants by using:

- FIX [24]
- FAST [33]
- ITCH [60]
- OUCH [61]
- Native protocols [41]

2.7.2.1 FIX (Financial Information eXchange)

FIX was invented by Salomon Brothers and Fidelity Investments in the 1990s to communicate security orders and their executions [33]. It has become the most widely used protocol by trading organizations. FIX is an open standard non-propriety protocol that is used by buy and sell firms, trading platforms and regulators to transmit trade data. The protocol allows organizations to easily communicate domestically and internationally with each other. FIX was originally developed for equity trading in the pre-trade environment but now it is also used in the foreign exchange, fixed income and derivatives markets. FIX is maintained by the FIX Community Trading member firms which maintain the various message specifications [24].

2.7.2.2 FAST (FIX Adapted for Streaming)

The popularity of FIX allowed more firms to trade and this increased the volume of the market data. This increased the network latency and market participants did not receive updates in an acceptable time. The Market Data Optimization Working group created the FAST protocol to address this issue. The FAST protocol reduces latency by encoding and compressing the data before transmission [33].

2.7.2.3 ITCH and OUCH

The ITCH and OUCH protocol was developed by The Nasdaq OMX Group. ITCH is used to publish market data only. OUCH is used to place, amend and cancel orders. The messages are fixed length without delimiters or tags. Prices are represented as integers and not text [60, 61].

2.7.2.4 Native

Native protocols are custom built protocols. The simulator will use the JSE's native protocol specification as a base to exchange information with clients.

2.8 Johannesburg Stock Exchange

The Johannesburg Stock Exchange (JSE) was founded in 1887. The open outcry system was used to trade stocks. The Johannesburg Equities Trading (JET) system replaced the opened outcry system in 1996. SETS replaced JET in 2002. In 2007 TradeElect replaced SETS. This system was licensed from the London Stock Exchange (LSE). The matching engine ran in London. In 2012 the matching engine moved to Johannesburg which made high frequency trading possible at the JSE[40] [8].

2.9 Conclusion

Studying market microstructure is challenging due to the various changes in the market, regulation and technology. The current literature focuses on analyzing existing exchanges and the building of agent based models. A low latency high throughput matching engine does not exist for academics to further their understanding in this field. The rest of this dissertation describes the requirements, design and testing of a matching engine.

3 Requirements

3.1 High Level Overview

The simulator needs 3 main components:

1. The Trading Gateway
2. The Matching Engine
3. Market Data Gateway

Clients will send order events to the trading gateway. The trading gateway receives the client request, validates the request and then sends it to the matching component to be processed. The trading gateway sends updates to clients to indicate the status of the requests. The market data gateway sends market data updates to all connected clients. A website is required to monitor and configure the stocks and clients.

3.2 Scope

The matching engine requirements were obtained from the JSE documentation on their website i.e. `https://www.jse.co.za/services/technologies.` FIX and ITCH gateways are out of scope. The documents used were:

1. Volume 00 - Trading and Information Overview v2.03 [39].
2. Volume 01 - Native Trading Gateway v2.02 [42].
3. Volume 05 - Market Data Feed (ITCH - UDP) v2.04 [43].

3.3 Components

3.3.1 Clients

The clients are computer algorithms that send order events to the simulator. The software also listens for market data updates from the simulator.

3.3.2 The Trading Gateway

The Trading Gateway receives order events from clients. It forwards the request to the matching engine component if it passes the initial validation. It sends updates to the client to indicate if the status of the event.

3.3.3 Matching Engine

The Matching Engine component process the events from the Trading Gateway. It manages one or more limit order books. If there is an update to the LOB, it sends updates to the Market Data Gateway

3.3.4 The Market Data Gateway

The Market Data Gateway receives updates from the Matching Engine component. It sends updates to all connected clients.

3.3.5 The Website

The website receives updates from the Market Data Gateway. It displays the LOBs for each security and allows the user to configure the stocks and clients. The testing framework is configured and started from the website.

3.4 Functional Requirements - Trading Gateway

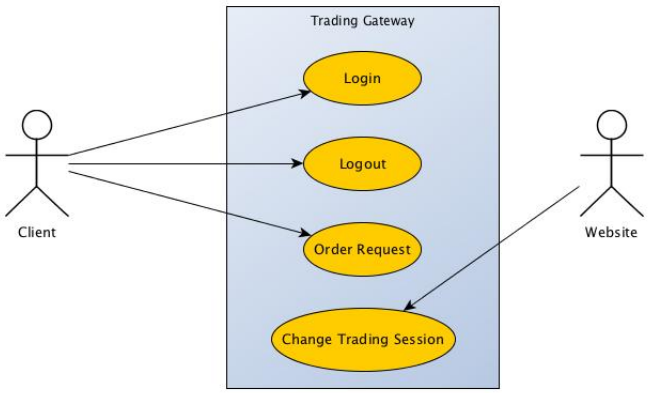


Figure 4: Trading Gateway Use Case

3.4.1 Login

1. The client sends a login request message to the Trading Gateway.
2. The Trading Gateway validates the client.
   (a) If the client is already logged in, the gateway responds with a ConcurrentLoginLimitReached Reject Code message.
   (b) If the username or password is invalid, the gateway responds with a InvalidCompIDOrPassword Reject Code message.

3.4.2 Logout

1. The client sends a logout request message to the Trading Gateway.
   (a) The Trading Gateway removes the client from its list of connected clients.
   (b) The Trading Gateway sends a response to the client to indicate if the logout was successful.
2. The Trading Gateway logouts the client when it shuts down.

3.4.3 Order Request

1. A client sends an Order Message to Trading Gateway.
2. The Trading Gateway validates the message.
3. If the message is valid, it sends the message to get matched.

3.4.4 Change Trading Session

1. When the trading session is required to change, the Website component sends a message to the Trading Gateway.
2. The Trading Gateway sends the message to the Matching Engine.

3.5 Functional Requirements - Matching Engine

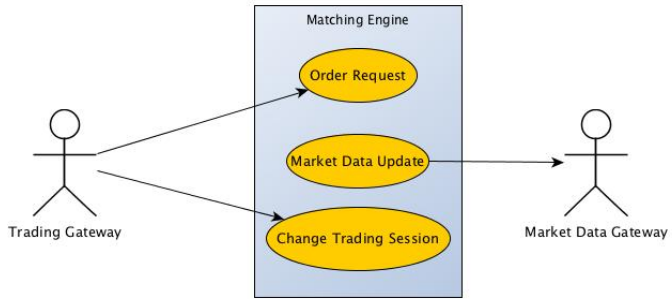


Figure 5: Matching Engine Use Case

The Trading Gateway sends order requests to the Matching Engine. Valid order types are: market, limit, hidden limit, stop and stop limit. Each order has a Time In Force (TIF) value that determines how long an order is active until it is executed, deleted or expired. The TIF value of an order cannot be amended. The engine matches the orders using matching algorithms based on the trading session. Market data updates are sent to the Market Data Gateway to indicate changes in the LOB. See Appendix B for test cases of the Matching Engine.

3.5.1 Trading Sessions

Figure 6 shows the trading sessions from the JSE. The sessions are active during different times of the day.



Figure 6: Trading Sessions [39]

3.5.2 Order Types

3.5.2.1 *Market Order (MO)*

1. A market order contains the quantity of shares to trade, but not the price.
2. It executes against existing orders in the LOB.
3. If the order is not filled, the remaining quantity expires.
4. Market orders submitted during an auction call session will remain in the LOB until the uncrossing is done. Orders that are not filled will expire.

3.5.2.2 *Limit Order (LO)*

1. A limit order displays a quantity and the limit price.
2. A limit order may execute at prices equal to or better than its limit price. If the order is not filled, the remainder will be added to the order book or expired based on the TIF.

3.5.2.3 *Hidden Limit Order (HO)*

1. A hidden limit order is a limit order that has a price and quantity that is not visible to other market participants.
2. It can execute against other visible and hidden orders in the order book.
3. Minimum Reserve Size (MRS)
   (a) MRS is the minimum order quantity for orders to qualify as hidden limit orders.
   (b) All hidden limit orders are validated against this parameter.
4. Minimum Execution Size (MES)
   (a) Minimum Execution Size is the minimum quantity of the hidden limit order which is permitted to execute.
   (b) Every hidden limit order must contain a MES.
   (c) It is a multiple of the instruments lot size.
   (d) It is greater than or equal to the MRS.
   (e) It is ignored during an auction call session.
   (f) It can be amended on existing/unexecuted orders.
   (g) If a hidden limit order aggresses the order book and has a MES that cannot be satisfied, then the order will be added to the order book or expired based on the TIF.
5. Hidden limit orders submitted during an auction call session are rejected.
6. During execution:
   (a) If the quantity remaining on the order is $<$ Minimum Reserve Size then the order will be expired.
   (b) If the quantity remaining on the order is $\geq$ Minimum Reserve Size but $<$ MES then the order will be expired.
   (c) If the quantity remaining on the order is $\geq$ Minimum Reserve Size and$\geq$ MES than the order will remain in the book or expired based on the TIF.

3.5.2.4 *Stop Order (SO)*

1. A stop order is a market order with a stop price.
2. Stop orders do not enter the order book until their stop price is reached.
3. When the stop price is reached, the stop order becomes a market order.

3.5.2.5 *Stop Limit Order (SL)*

1. A stop limit order is a limit order with a stop price.
2. Stop limit orders do not enter the order book until their stop price is reached.
3. When the stop price is reached, the stop limit order becomes a limit order.

3.5.2.6 *Stop Order and Stop Limit Order Election Rules*

1. Stop and stop limit buy orders will be elected if the last traded price is equal to or greater than the stop price.
2. Stop and stop limit sell orders will be elected if the last traded price is equal or less than the stop price.
3. Incoming stop and stop limit orders will be elected if the stop price is already reached.
4. Incoming stop and stop limit orders will be parked if the last traded price does not exist.
5. A stop and stop limit order election occurs only after an aggressing order has completed its execution.

3.5.2.7 *Stop Order and Stop Limit Order Priority Rules*

1. Buy or sell orders with the greatest difference between its stop price and the auction price will be elected first.
2. If multiple orders are at the same difference (buy and sell), the oldest order will be elected first.

3.5.3 Time In Force (TIF)

3.5.3.1 *At the Opening (OPG)*

1. OPG orders will only be accepted during the Opening Auction.
2. OPG orders that are not filled during the uncrossing will expire at the end of the Opening Auction.

3.5.3.2 *Good For Auction (GPA)*

1. GFA orders submitted are parked until the next auction.
2. GFA orders are injected into the order book at the start of the auction.
3. GFA orders that are not filled during the uncrossing will be parked for the next auction.
4. GFA orders are removed when filled or cancelled.

3.5.3.3 *Good For Intraday Auction (GFX)*

1. GFX orders submitted are parked until the next Intraday auction.
2. GFX orders are injected into the order book at the start of the Intraday auction.
3. GFX orders that are not filled during the uncrossing will expire at the end of the Intraday Auction.

3.5.3.4 *At the Close (ATC)*

1. ATC orders submitted are parked until the next Closing auction.
2. ATC orders are injected into the order book at the start of the Closing auction.
3. ATC orders that are not filled during the uncrossing will expire at the end of the Closing Auction.

3.5.3.5 *Day (DAY)*

1. DAY orders will expire at the end of the trading day.
2. If an order does not specify a TIF, it will default to DAY.

3.5.3.6 *Immediate or Cancel (IOC)*

1. IOC orders can be partially or fully filled.
2. The IOC order will be expired if there is a remaining quantity.
3. IOC orders will be rejected during auction call sessions.
4. Stop orders or stop limits orders with IOC time in force will be parked until the stop price is reached.

3.5.3.7 *Fill or Kill (FOK)*

1. FOK orders are either fully filled or expired. Partial filled orders are not allowed.
2. Stop orders or stop limits orders with FOK time in force will be parked until the stop price is reached.

3.5.3.8 *Good till Cancel (GTC)*

1. GTC orders can remain in the order book for a maximum of 90 calendar days.
2. GTC orders remain in the order book until the order is filled, cancelled or expiration date is reached.
3. Expiration date is calculated from the date the order is submitted.
4. If the order is amended, the expiration date is not re-calculated.

3.5.3.9 *Good till Date (GTD)*

1. GTD orders remain in the order book until the order is filled, cancelled or the expiration date specified is reached.
2. If an expiry time is specified for a GTD order, the order will be rejected.
3. GTD orders will expire when the 90 calendar day rule is reached.

3.5.3.10 *Good till Time (GTT)*

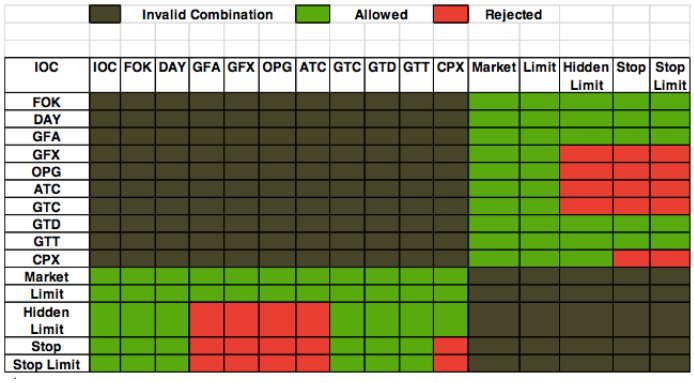
1. GTT orders remain in the order book until the specified expiry time is reached for the trading day.
2. GTT orders that have not expired will expire at the start of the Post Close Session.
3. GTT orders that have an expiry time during an auction ,will not expire. These orders will expire after the uncrossing.

3.5.3.11 *Closing Price Cross (CPX)*

1. CPX orders submitted are parked until the Closing Price Cross session.
2. Unexecuted CPX orders are expired at the end of the Closing Price Cross session.
3. Stop Order and Stop Limit Order with a TIF of CPX are rejected.

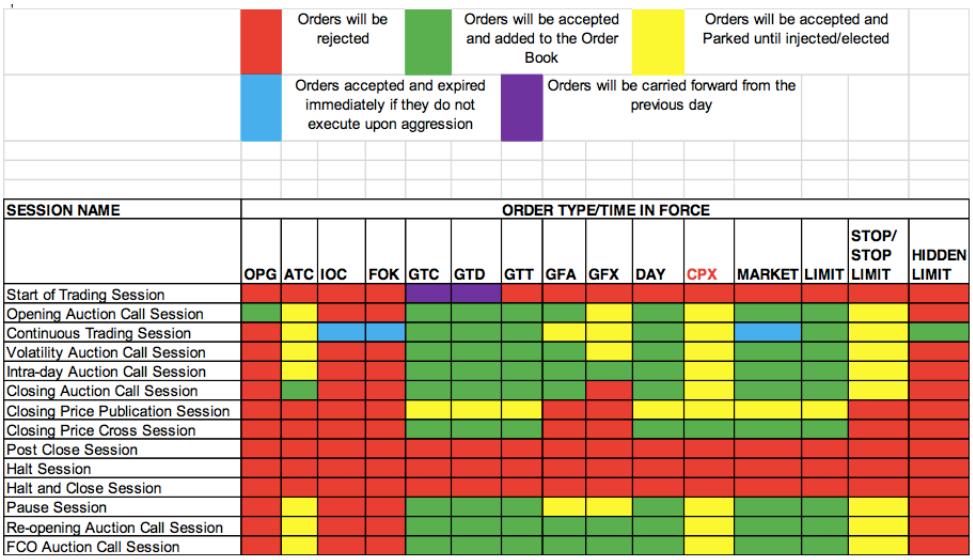
3.5.4 Valid Combinations

Figures 7 and 8 show the valid combinations of order types, time in force values and trading sessions.



IOC	IOC	FOK	DAY	GFA	GFX	OPG	ATC	GTC	GTD	GTT	CPX	Market	Limit	Hidden Limit	Stop	Stop Limit
FOK																
DAY																
GFA																
GFX																
OPG																
ATC																
GTC																
GTD																
GTT																
CPX																
Market																
Limit																
Hidden Limit																
Stop																
Stop Limit																

Figure 7: TIF Order Type Valid Combinations [39]



SESSION NAME	ORDER TYPE/TIME IN FORCE														
	OPG	ATC	IOC	FOK	GTC	GTD	GTT	GFA	GFX	DAY	CPX	MARKET	LIMIT	STOP/ STOP LIMIT	HIDDEN LIMIT
Start of Trading Session															
Opening Auction Call Session															
Continuous Trading Session															
Volatility Auction Call Session															
Intra-day Auction Call Session															
Closing Auction Call Session															
Closing Price Publication Session															
Closing Price Cross Session															
Post Close Session															
Halt Session															
Halt and Close Session															
Pause Session															
Re-opening Auction Call Session															
FCO Auction Call Session															

Figure 8: Trading Session TIF Order Type Valid Combinations [39]

3.5.5 Price-Time Priority Algorithm

The Price-Time Priority algorithm is used during the Continuous Trading Session. Each order type is processed differently based on the state of the LOB and the TIF. The requirements listed for each order request type has a corresponding test case in Appendix B that was used to evaluate the functionality. For example, the test case B.1.1 in Appendix B implements the requirement 3.5.5.2 item 1.

3.5.5.1 *Order is matched to an existing order*

1. A new order is matched to an existing order in the LOB.
2. The existing order is removed from the LOB.

3. The Matching Engine sends an update to the Trading Gateway to indicate the status of the new order.
4. The Matching Engine sends an update to the MarketData Gateway to indicate the trades that have been created.
5. The Matching Engine sends an update to the MarketData Gateway to indicate the change in price.
6. If the new order is filled, then it is not added to the LOB.
7. If the new order is partially filled, then it is added to the LOB.

3.5.5.2 *Market Order Request*

1. Add bid market message to an empty LOB. Order expires.
2. Add offer market message to an empty LOB. Order expires.
3. Add bid market message to a non-empty LOB. The order is matched to an existing order. The order is filled.
4. Add offer market message. The order is matched to an existing order. The order is filled.
5. Add bid market message. Order is partially filled. Remaining order quantity is expired.
6. Add offer market message. Order is partially filled. Remaining order quantity is expired.

3.5.5.3 *Limit Order Request*

1. Add bid limit message to an empty LOB. The order is inserted into the LOB.
2. Add bid limit message with different bid price. The order is inserted into the LOB.
3. Add bid limit message with existing bid price. The order is inserted into the LOB.
4. Add bid limit message with existing offer price. The order is not matched. The order is inserted into the LOB.
5. Add bid limit message with existing offer price. The order is matched.
6. Bid limit order matches offer limit order. Time-priority determines the matching orders .
7. Add offer limit message to an empty lob. The order is inserted into the LOB.

8. Add offer limit message with different offer price. The order is inserted into the LOB.
9. Add offer limit message with existing offer price. The order is inserted into the LOB.
10. Add offer limit message with existing bid price. The order is not matched. The order is inserted into the LOB.
11. Add offer limit message with existing bid price. The order is matched. .
12. Offer LO matches Bid LO. Time-priority determines the matching orders.

3.5.5.4 *Hidden Order Request*

1. Incoming offer LO matches on contra side including HO. Visible LO takes precedence over HO at same price point.MRS = 700
2. Incoming bid LO matches on contra side including HO. Visible LO takes precedence over HO at same price point.MRS = 700
3. Incoming offer LO matches on contra side including HO. MES is used to execute HO.MRS = 700
4. Incoming offer LO matches on contra side including HO. HO is skipped due to MES constraint.MRS = 700
5. Incoming offer HO matches on contra side including HO. Passive HO is skipped due to MES constraint.MRS = 700
6. Executing a Sell Limit message with price improvement
7. Executing a Sell Order at aggressing message's limit price
8. Not Executing a Sell Order due to aggressing message's limit price breach
9. Executing a Sell Order based on Price-Visibility-Time Execution Priority
10. Executing a Buy Limit message with price improvement
11. Executing a Buy Order at the aggressing message's limit price
12. Not Executing a Buy Order due to aggressing message's limit price breach
13. Executing a Buy Order based on Price-Visibility-Time Execution Priority
14. Executing a Buy Order stepping over a Hidden Limit message due to a MES constraint
15. Executing a Buy Market message with price improvement
16. Executing a Sell Market message with price improvement

17. Executing a Sell Market message which creates sufficient quantity for the message at the visible best offer to execute against a Hidden Limit message with a MES constraint.

3.5.5.5 *Stop Order Request*

1. Add buy stop order to an empty lob
2. Add sell stop order to an empty lob
3. Buy stop order added. Last trade price does not exist. No executions
4. Sell stop order added. Last trade price does not exist. No executions
5. Buy order aggresses order book with buy stop order. Stop order executed
6. Sell order aggresses order book with sell stop order. Stop order executed
7. Buy stop order added. Last trade price exists. Stop order executed
8. Sell stop order added. Last trade price exists. Stop order executed
9. Buy stop orders with greatest difference between its stop price and the last traded price will be elected first
10. Sell stop orders with greatest difference between its stop price and the last traded price will be elected first
11. Multiple buy stop orders with the same difference between its stop price and the last traded price. Oldest executed first
12. Multiple sell stop orders with the same difference between its stop price and the last traded price. Oldest executed first

3.5.5.6 *Stop Limit Order Request*

1. Add buy stop limit message to an empty lob
2. Add sell stop limit message to an empty lob
3. Buy stop limit message added. Last trade price does not exist. No executions
4. Sell stop limit message added. Last trade price does not exist. No executions
5. Buy message aggresses message book with buy stop limit message. Stop message executed
6. Sell message aggresses message book with sell stop limit message. Stop message executed
7. Buy stop limit message added. Last trade price exists. Stop message executed

8. Sell stop limit message added. Last trade price exists. Stop message executed
9. Buy stop limit orders with greatest difference between its stop price and the last traded price will be elected first
10. Sell stop limit orders with greatest difference between its stop price and the last traded price will be elected first
11. Multiple buy stop limit orders with the same difference between its stop price and the last traded price. Oldest executed first
12. Multiple sell stop limit orders with the same difference between its stop price and the last traded price. Oldest executed first

3.5.5.7 *Cancel Order Request*

1. Cancel limit order bid in empty order book. Cancel request rejected
2. Cancel limit order offer in empty order book. Cancel request rejected
3. Cancel limit order bid. Bid removed
4. Cancel limit order offer. Offer removed
5. Cancel stop order bid. Bid removed
6. Cancel stop order offer. Offer removed
7. Cancel stop limit order bid. Bid removed
8. Cancel stop limit order offer. Offer removed
9. Cancel hidden order bid. Bid removed
10. Cancel hidden limit order offer. Offer removed

3.5.5.8 *Replace Order Request*

1. Replace limit order bid in empty order book. Replace request rejected
2. Replace limit order offer in empty order book. Replace request rejected
3. Replace limit order bid quantity. Quantity replaced
4. Replace limit order bid GTD. GTD replaced
5. Replace limit order bid GTT. GTT replaced
6. Replace limit order offer quantity. Quantity replaced
7. Replace limit order offer GTD. GTD replaced

8. Replace limit order offer GTT. GTT replaced
9. Replace limit order bid quantity. Quantity > existing quantity. Order re-aggress the order book
10. Replace limit order offer quantity. Quantity < existing quantity. Order re-aggress the order book
11. Replace limit order bid price. Order re-aggress the order book
12. Replace limit order offer price. Order re-aggress the order book
13. Replace HO order bid quantity. Quantity replaced
14. Replace HO order bid GTD. GTD replaced
15. Replace HO order bid GTT. GTT replaced
16. Replace HO order bid MES. MES replaced
17. Replace HO order offer quantity. Quantity replaced
18. Replace HO order offer GTD. GTD replaced
19. Replace HO order offer GTT. GTT replaced
20. Replace HO order offer MES. MES replaced
21. Replace HO order bid quantity. Quantity > existing quantity. Order re-aggress the order book
22. Replace HO order offer quantity. Quantity < existing quantity. Order re-aggress the order book
23. Replace HO order bid price. Order re-aggress the order book
24. Replace HO order offer price. Order re-aggress the order book
25. Replace Stop order bid limit price. Limit price replaced
26. Replace Stop order offer limit price. Limit price replaced
27. Replace Stop order bid quantity. Quantity replaced
28. Replace Stop order bid GTD. GTD replaced
29. Replace Stop order bid GTT. GTT replaced
30. Replace Stop order offer quantity. Quantity replaced
31. Replace Stop order offer GTD. GTD replaced
32. Replace Stop order offer GTT. GTT replaced

33. Replace Stop order bid quantity. Quantity > existing quantity. Order re-aggress the order book
34. Replace Stop order offer quantity. Quantity < existing quantity. Order re-aggress the order book
35. Replace Stop order bid stop price. Order re-aggress the order book
36. Replace Stop order offer stop price. Order re-aggress the order book

3.5.6 Passive Price Improvement Algorithm

The Passive Price Improvement Algorithm executes each time an aggressive order matches an existing order. The execution price is calculated using algorithm 2.

Algorithm 2 Passive Price Improvement Algorithm

```
1: if existing order is a buy order then
2:   if execution price < best visible bid then
3:     execution Price ← best visible bid + 0.5
4:   else
5:     execution price ← existing order price
6:   end if
7: else
8:   if execution price < best visible offer then
9:     execution price ← best visible offer - 0.5
10:  else
11:    execution price ← existing order price
12:  end if
13: end if
```

3.5.7 Filter and Uncross Algorithm

The Filter and Uncross algorithm runs each time the Best Bid Offer (BBO) changes or every 30 seconds. The algorithm is split into 2 parts. A heuristic search (Hill Climber Algorithm) is run to find the optimal volume of hidden limit orders that can be executed. The Hill Climber algorithm is an optimization technique that iteratively searches for the solution to a problem by changing one element in each iteration [14]. The algorithm stops when one of the following criteria are met:

- No more improvements can be made.
- A fixed number of iterations have been completed.
- A goal point is attained.

The search will filter out hidden limit orders with MES constraints that are not eligible. After the filtering, rules are used to select the orders and price to executed in the crossed region.

3.5.7.1 Filter Algorithm (Hill Climber Algorithm)

Algorithm 3 Filter Algorithm (Hill Climber Algorithm) - Step 1

- Calculate temporary executable volume for each order in the crossed region.
- The executable volume for a hidden order is the order quantity.
- The executable volume for a visible order is the order quantity.

Example 1.

Ex. Vol.	ID.	Type	MES.	Bid Size	Price	Type	MES.	Offer Size	ID.	Ex. Vol
					68	LO		1 000	5	
					67	LO		500	6	
					66	HO	1 000	3 000	7	
30 000	**12**	**HO**	**25 000**	**30 000**	**65**	**HO**	**7 000**	**12 000**	**8**	**12 000**
					64	**HO**	**5 000**	**10 000**	**9**	**10 000**
10 000	**11**	**HO**	**4 000**	**10 000**	**63**	**LO**		**3 500**	**10 + 13**	**3 500**
	4	HO	6 000	15 000	62					
	3	HO	1 000	3 000	61					
	2	LO		500	60					
	1	LO		1 000	59					

Algorithm 4 Filter Algorithm (Hill Climber Algorithm) - Step 2

1. Calculate Total Bid Executable Volume.
2. Calculate Total Offer Executable Volume.
3. Start on the side that contains the maximum executable volume in the crossed region.

Example 2.

- Total Bid Executable Volume= $30000 + 10000 = 40000$
- Total Offer Executable Volume= $12000 + 10000 + 3500 = 25500$
- Start on Bid side.

Algorithm 5 Filter Algorithm (Hill Climber Algorithm) - Step 3

1. Start with the highest priority hidden limit order and go down the list *i.e.* Bid side: highest price to lowest price. Offer side: lowest price to highest price.
2. Calculate the Volume Available and Volume Ahead for each order.
   (a) Volume Available← the executable volume of eligible orders on the other side of the book (including visible orders).
   (b) Volume Ahead← the executable volume of eligible orders on the same side of the book (including Visible Orders) at a higher priority.
3. If (Volume Available – Volume Ahead) $\geq$ MES then the executable volume for the order = MIN((Volume Available – Volume Ahead), HO Order Quantity).
4. If (Volume Available – Volume Ahead) $\leq$ MES then the executable volume is set to zero.

Example 3.

Ex. Vol.	ID.	Type	MES.	Bid Size	Price	Type	MES.	Offer Size	ID.	Ex. Vol.
					68	LO		1 000	5	
					67	LO		500	6	
					66	HO	1 000	3 000	7	
25 500	**12**	**HO**	**25 000**	**30 000**	**65**	**HO**	**7 000**	**12 000**	**8**	**12 000**
					64	**HO**	**5 000**	**10 000**	**9**	**10 000**
0	**11**	**HO**	**4 000**	**10 000**	**63**	**LO**		**3 500**	**10 + 13**	**3 500**
	4	HO	6 000	15 000	62					
	3	HO	1 000	3 000	61					
	2	LO		500	60					
	1	LO		1 000	59					

Algorithm 6 Filter Algorithm (Hill Climber Algorithm) - Step 4

1. Process all hidden orders on the selected side of the order book.
2. Process the contra side of the order book when done with the selected side.

Algorithm 7 Filter Algorithm (Hill Climber Algorithm) - Step 5

1. After the contra side of the order book is processed, check if the executable volumes were updated.
2. If the executable volumes were not updated, then the filtering algorithm is complete and the process moves to the uncrossing algorithm.
3. If the executable volumes were updated, then the process starts again from Step 2.

3.5.7.2 Uncrossing Algorithm

Algorithm 8 Uncrossing Algorithm - Calculate the target price

```
1: if BestVisibleBid ≠ 0 and BestVisibleOffer ≠ 0 then
2:    targetPrice ← MID(BestVisibleBid, BestVisibleOffer)
3: else if BestVisibleBid ≠ 0 and BestVisibleOffer = 0 then
4:    targetPrice ← BestVisibleBid
5: else if BestVisibleBid= 0 and BestVisibleOffer ≠ 0 then
6:    targetPrice ← BestVisibleOffer
7: else if lastAutomatedTradePrice ≠ 0 then
8:    targetPrice ← lastAutomatedTradePrice
9: else if lastAutomatedTradePrice = 0 and previousDayClosingPrice ≠ 0 then
10:   targetPrice ← previousDayClosingPrice
11: else
12:   targetPrice ← referencePrice
13: end if
```

Algorithm 9 Uncrossing Algorithm - Calculate maximum executable volumes at each price level in the crossed region

```
1: for each price level in crossed region do
2:    if bid executable volume > offer executable volume then
3:       maximum executable volume ← bid executable volume
4:    else
5:       maximum executable volume ← offer executable volume
6:    end if
7: end for
```

Example 4.

Price	Executable Volume
65	25 500
64	13 500
63	3 500

Algorithm 10 Uncrossing Algorithm - Calculate the target trade price that maximizes the executable volume

```
Set MaxPV ← Maximum Volume Maximizing Price
Set MinPV ← Minimum Volume Maximizing Price

if targetPrice ≥ MaxPV then
    targetPrice ← MaxPV
else if targetPrice ≤ MinPV then
    targetPrice ← MinPV
else
    targetPrice ← targetPrice
end if
```

Example 5.

Mid of visible BBO(60 to 63)	61.50
Max. Volume Maximizing Price	65
Min. Volume Maximizing Price	65
Target Trade Price	**65**

Algorithm 11 Uncrossing Algorithm - Agress Order Book

```
if buy side executable volume < sell side executable volume then
    buy side agresses the sell side using calculated target trade price
else if sell side executable volume < buy side executable volume then
    sell side agresses the buy side using calculated target trade price
else
    buy side agresses the sell side using calculated target trade price
end if
```

Example 6.

Quantity	**Price**	**Orders**
1 000	65	Orders 12 and 10
2 500	65	Orders 12 and 13
10 000	65	Orders 12 and 9
12 000	65	Orders 12 and 8

3.5.8 Auction Algorithm

The Volume Maximizing Auction Algorithm is used for all auctions. It calculates the auction price for each stock at which the largest number or shares can be executed. It finds the price at which the volume is maximized. Algorithm 12 shows the 4 rules. It moves to the next rule if it is unable to obtain an auction price. The examples shown are from [53].

Algorithm 12 Auction Algorithm

1. **Maximum Execution**: The highest executable volume for each possible price.
2. **Minimum Surplus:** The lowest surplus for each possible price.
3. **Market Pressure:**
   (a) If the buy executable volume > sell executable volume then the auction price is the highest price.
   (b) If the sell executable volume > buy executable volume then the auction price is the lowest price.
4. **Reference Price:** The auction price will be the possible price closest to the reference price.

Example 7. Initial limit order book

Buy	
Size	**Price**
10 000	105.5
5 600	104.5
200	104

Sell	
Size	**Price**
2 500	MO
6 900	103
1 000	104.5
200	106

Example 8. Calculate the aggregated volume for each side.

Buy		**Price**	**Sell**	
Aggregate Vol.	**Vol. at price**		**Vol. at price**	**Aggregate Vol.**
	0	MO	2 500	
0	0	106	200	10 600
10 000	10 000	105.5	0	10 400
10 000	0	105	0	10 400
15 600	5 600	104.5	1 000	10 400
15 800	200	104	0	9 400
15 800	0	103.5	0	9 400
15 800	0	103	6 900	9 400

Example 9. Maximum Execution. Auction Price = 104.5

Price	**Agg. Vol. (Buy)**	**Agg. Vol. (Sell)**	**Auction Vol.**
106	0	10 600	0
105.5	10 000	10 400	10 000
105	10 000	10 400	10 000
104.5	**15 600**	**10 400**	**10 400**
104	15 800	9 400	9 400
103.5	15 800	9 400	9 400
103	15 800	9 400	9 400

Example 10. Minimum Surplus. If there are multiple maximum executable volumes, then the minimum surplus rule is used. The auction price = 104.5

Price	**Agg. Vol. (Buy)**	**Agg. Vol. (Sell)**	**Auction Vol.**	**Auction Surplus**
106	0	10 600	0	-10 600
105.5	10 000	10 400	10 000	-400
105	10 000	10 400	10 000	-400
104.5	**15 600**	**10 400**	**10 400**	**5 200**
104	**15 800**	**10 400**	**10 400**	**5 400**
103.5	15 800	9 400	9 400	6 400
103	15 800	9 400	9 400	6 400

Example 11. Market Pressure. If there are multiple surpluses, then the market pressure rule is used. Auction Price = 105

Price	**Agg. Vol. (Buy)**	**Agg. Vol. (Sell)**	**Auction Vol.**	**Auction Surplus**
106	0	10 600	0	-10 600
105.5	10 000	10 400	10 000	-400
105	**15 600**	**10 400**	**10 400**	**5 200**
104.5	**15 600**	**10 400**	**10 400**	**5 200**
104	**16 800**	**10 400**	**10 400**	**6 200**
103.5	16 600	9 400	9 400	7 200
103	16 600	9 400	9 400	7 200

Example 12. Reference Price. Auction Price = Price closest to last trade price (104.5 or 104)

Price	**Agg. Vol. (Buy)**	**Agg. Vol. (Sell)**	**Auction Vol.**	**Auction Surplus**
105.5	0	14 000	0	-14 000
105	6 000	8 000	6 000	-2 000
104.5	**7 000**	**8 000**	**7 000**	**-1 000**
104	**8 000**	**7 000**	**7 000**	**1 000**
103.5	14 000	6 000	6 000	8 000
103	14 000	6 000	6 000	8 000

3.5.9 Static And Dynamic Price

Static Reference Price(SPR): At the beginning of the day the Static Reference Price for an instrument will be its previous day's closing price. The Static Reference Price will be updated after each auction (Opening, Re-Opening, Intraday or Volatility)

Dynamic Reference Price(DPR): The Dynamic reference price applicable to an instrument is updated continuously during the course of a day and serves as a reference point when calculating the price of a share during either an Auction or Continuous Trading Session. At the start of the day, the Dynamic Reference Price for an instrument will be the previous day's closing price.

3.6 Functional Requirements - Market Data Gateway

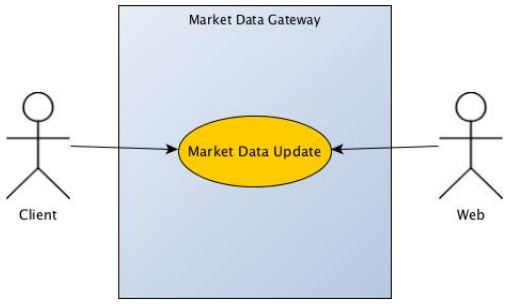


Figure 9: Market Data Gateway Use Case

3.6.1 Market Data Update

1. All clients receive market data updates from the gateway.
2. The website receives market data updates and internal messages to monitor the application.

3.7 Functional Requirements - Website

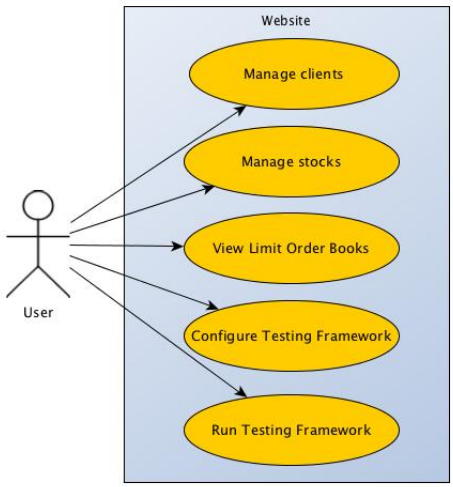


Figure 10: Website Use Case

3.7.1 Manage clients

A user can create, update and delete clients. Clients not configured will not be able to send orders to the matching engine.

3.7.2 Manage stocks

A user can create, update and delete stocks.

3.7.3 View Limit Order Books

A user can view the limit order book for each stock. A bar chart will be generated to display the active orders. Tables will show the details of the bids, offers, trades and all submitted orders.

3.7.4 Configure Testing Framework

A user can configure the parameters of the testing framework.

3.7.5 Run Testing Framework

A user can start and stop the testing framework.

3.8 Deployment Requirements

Each component must be able to run as a separate process. The software must be able to be deployed and run on different operating systems. The components must be able to be deployed on a single server or multiple servers.

4 Design

4.1 Introduction

The previous chapter listed the requirements for the simulator. The goal of this chapter describes the design decisions made to implement the different components. It covers the following:

1. The architecture.
2. The implementation programming language.
3. The message and communication protocols.
4. The limit order book data structure.
5. The open source libraries selected.
6. The design patterns selected to implement the business logic.
7. The saving and monitoring of events.

4.2 Architecture Overview

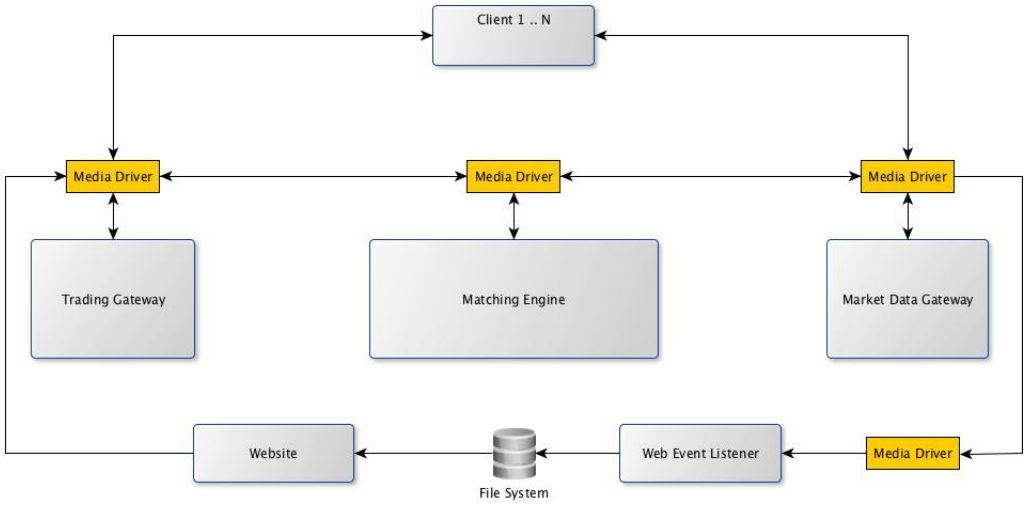


Figure 11: High Level Architecture Diagram

Flow

1. The order event will be created using the Simple Binary Encoding (SBE)[82] library at the client.

2. The Media Driver is a component of the Aeron library [84]. It is used to communicate between all components. Each component has its own media driver to shovel events to and from the component.
3. The SBE message will be sent to the Trading Gateway.
4. The Trading Gateway will send a response SBE message back to the client if the message fails validation.
5. The Trading Gateway will forward the SBE message to the Matching Engine if the message passes validation.
6. The Matching Engine will process the message. It will send a message back to the Trading Gateway to indicate the status of the message. The Trading Gateway will forward the message to the client.
7. The Matching Component will send an update to the Market Data Gateway if there was a change in the limit order book.
8. The Market Data Gateway will forward the updates to all connected clients. It will forward some of these events to the Web Event Listener.
9. The Web Event Listener saves each event to the file system using MapDB [45].
10. The Website reads and displays the data from the filesystem.

Alternate Flows

- The Website can send admin events to the Trading Gateway to control the behaviour of the application *i.e.* start and stop a trading session. These events flow through the different components.
- The clients and Website can send events to the Gateways to query the limit order books.

4.3 Implementation Language

C++ [35] and Java [69] were considered for this project and Java was selected as the implementation language. The decision was based on 4 key factors: performance, portability, development time and open source libraries.

4.3.1 Performance

The performance gap between Java and C++ is closing [55]. The matching engine needs the capability to implement changes with predicable behavior across different hardware. Java's "write once, run anywhere" capability out weighs the benefit of C++ performance gains.

4.3.2 Portability

The matching engine and/or it's components will be deployed using different hardware configurations to determine the optimal setup. The code will not need to be recompiled each time and will give similar behavior using Java.

4.3.3 Development time

The software needs the capability to change the business rules to test various assumptions. The turn around time of each implementation needs to be quick. Java will allow developers to make these changes quicker than in C++. An application spends 90% of its time in 10% of the code and Java will allow the developers to focus on this 10% [50].

4.3.4 Open source libraries

There are various Java open source libraries that can be used to implement the software.

4.4 Message Protocol

The JSE uses text messages between the clients and their matching engine. This is inefficient as characters take up more memory and they are slower to transmit across the network. These messages do not need to be human readable and should be in binary format. Java serialization, Google Protocol Buffers (GPB) and Simple Binary Encoding (SBE) were tested and SBE was selected as the message protocol.

Java serialization was compared to GPB and GPB was fastest in serialization and deserialization. The size of the GPB object was smaller than the Java object. The test was executed 10 times and the averages are shown in table 5. The test was run 1 million times to test the speed of the 2 approaches (Appendix A.1).

	Java	**Google Protocol Buffers**
Size of 1 object (bytes)	460	45
Serialize Time (ms)	4846	121.8
Deserialize Time (ms)	27578	309.2

Table 5: Java vs Google Protocol Buffers

GPB was better than standard Java serialization, however it does not use the design decisions of SBE. The test in table 6 from Martin Thompson **[82]** shows that SBE has approximately 16-25 times greater throughput than GPB.

Test	Protocol Buffers (msg/ms)	SBE (msg/ms)	Ratio
Car Encode	619.467	10436.476	16.85
Car Decode	433.711	11657.190	26.88
Market Data Encode	2088.998	34078.646	16.31
Market Data Decode	1316.123	29193.600	22.18

Table 6: SBE vs GPB [82]

The simulator uses the Simple Binary Encoding (SBE) library to encode and decode messages. *"SBE is an OSI layer 6 presentation for encoding/decoding messages in binary format to support low-latency applications"* [82]. The library is being used to implement a new FIX technical standard [25].

SBE design principles are:

1. *Copy-Free*: It does not use an intermediate buffer when encoding or decoding messages.
2. *Native Type Mapping*: It maps the data to native types in the underlying buffer.
3. *Allocation Free*: The flyweight pattern is used to avoid unnecessary allocation.
4. *Streaming Access*: Best performance is achieved by reading a message only forward.
5. *Word Aligned Access*: The message is framed with 8 byte boundaries. Fields in the messages are sorted by type and size in descending order.
6. *Backwards Compatible*: The messages are versioned and are backward compatible.

4.5 Communication

Transmission Control Protocol (TCP) and User Datagram Protocol(UDP) are two common Internet Protocol(IP) that is used to transmit data over a network.

4.5.1 TCP

TCP [70] requires a connection to be set up between two applications before data can be transmitted. TCP is called a connection-orientated protocol. TCP is reliable because if a message is not received, it will try multiple times to deliver the message. A message can be retransmitted if requested. TCP will drop the connection if there are multiple timeouts. The messages that are received will always be in the order that it is sent. TCP is heavyweight protocol because it requires three packets to set up a connection. Data is read as a byte stream [43].

4.5.2 UDP

UDP [70] does not require a connection to be setup before data is transmitted. UDP is called a connectionless protocol. Data is transmitted irrespective if the receiver is ready to receive the message. UDP is unreliable because the sender does not know if the message was delivered. The messages that are received may not be in the same order that it is sent. UDP is a lightweight protocol because it does not check the connection or order of messages. Packets that are sent have boundaries and are checked for integrity if received [43].

4.5.3 JSE Communication

The JSE uses TCP for their Trading Gateway and UDP for their Market Data Gateway. The performance requirement of the simulator requires a low latency high throughput socket library. The technology to communicate between the components needs to support different transport protocols and easily adapt to a change in infrastructure. ZeroMQ[32] and Aeron libraries were tested and Aeron was selected as the communication protocol.

4.5.4 ZeroMQ

ZeroMQ is a message-oriented middleware library. The library is written in C but it has bindings for various other languages including Java (using JNI). There is a pure Java implementation called JeroMQ [37]. It gives you access to a low level socket API and supports various transports like in-process, interprocess, TCP and multicast. It implements different communication patterns e.g. request-reply, publish-subscribe, workload distribution, etc. Changing between the different communication protocols is straight forward. The library is fully documented on its website. The library was originally created for stock trading systems and therefore it is built for speed. Figure 12 shows the results of the tests conducted by CERN that compared ZeroMQ to several other MOM libraries and scored ZeroMQ the highest [19].

	patterns	QoS	resources	performance	user friendly	community	score
CORBA	✗	✓	✗	✓	✗	✗	2
Ice	✓	✓	✗	✓	✓	✓	5
Thrift	✗	✗	✓	✓	✗	✗	2
ZeroMQ	✓	✓	✓	✓	✓	✓	6
YAMI4	✓	✓	✓	✗	✓	✗	4
RTI	✗	✓	✗	✓	✗	✓	3
Qpid	✗	✓	✗	✗	✓	✓	3

Figure 12: Cern MOM Libraries

ZeroMQ is implemented in C++ and it needs to be wrapped in a JNI layer to be used in Java code. It has to be recompiled for each operating system type which went against my design principles. It also had internal queues to store messages in case the subscriber could not keep up with the publisher. If the system went down, these messages would have been lost.

4.5.5 Aeron

Most trading systems use TCP because it has built in reliability. However this reliability comes at a cost, as it slow. Aeron is an efficient reliable UDP unicast, multicast, and IPC message transport. Performance is the key focus. Aeron is designed to be the highest throughput with the lowest and most predictable latency possible of any messaging system.

Aeron's design principles [83]:

1. *Garbage free*: Object allocation is upfront and it does not contribute to GC pauses.
2. *Smart batching*: When there is a burst of traffic, they use an algorithm to send the messages in batches and fill network packets to the optimal size.
3. *Lock free algorithms*: No threads are blocked.

4. *Non-Blocking I/O*: I/O is not blocked.
5. *No exceptional cases*: The code paths are simple and predictable for the processor.
6. *Singe writer principle*: Only 1 writer will write to a resource.
7. *Prefer unshared state*: State is not shared between threads. Each thread will have a copy of the state and messages are used between threads to update the state. This works with the single writer principle.
8. *Avoid unnecessary data copies*: Data is only copied when required.

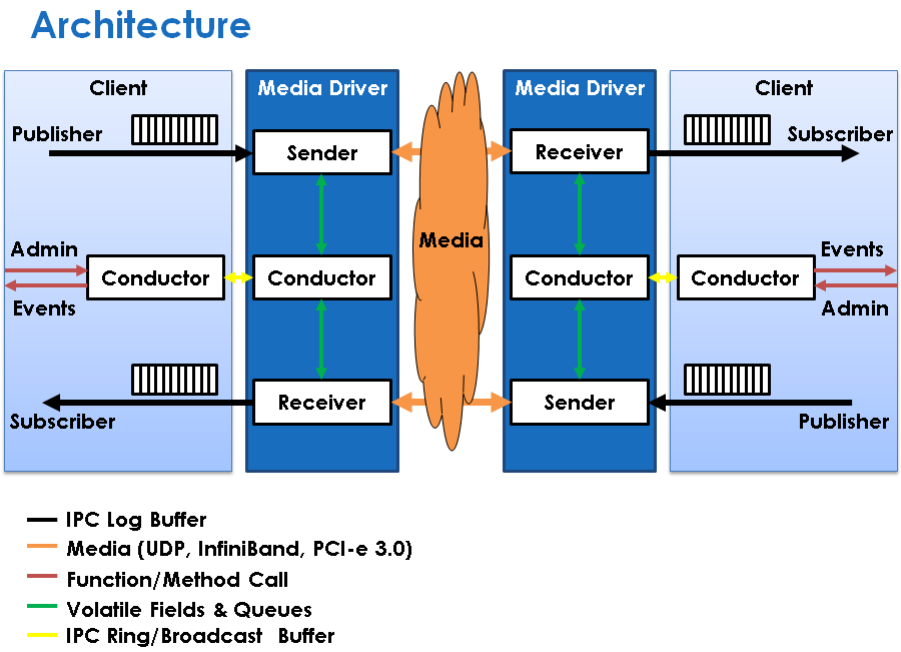


Figure 13: Aeron Architecture [81]

Figure 13 shows the architecture of Aeron with 2 clients. For further information refer to [81]. The simulator uses a separate media driver for each of the components. This will allow the media driver threads to focus on sending and receiving messages and the components to execute it's logic. If the client(s) are on the same server it can use the same media driver. The performance test done used Aeron's performance testing tools. The buffer size was set to the same size of my order messages which is 120 bytes and I sent 500 million messages between a publisher and subscriber on my laptop Mac Air (1.7 GHz Intel Core i7, 8 GB 1600 MHz DDR3). The publisher sent messages at a faster rate and there was only 20% CPU usage and only 1 GC event recorded.

Latency is important as it measures how fast a message is processed. However reliable low latency is also required. A High Dynamic Range (HDR) Histogram library was used to record and graph the latency. *"HDR is a Histogram that supports recording and analyzing sampled data value counts across a configurable integer value range with configurable value precision within the range. HDR Histogram is designed for recoding histograms of value measurements in latency and performance sensitive applications. Measurements show value recording times as low as 3-6 nanoseconds on modern (circa 2014) Intel CPUs" [79]*

The Histogram graph in figure 14 shows that the latency is 755 ns at the 99% percentile.

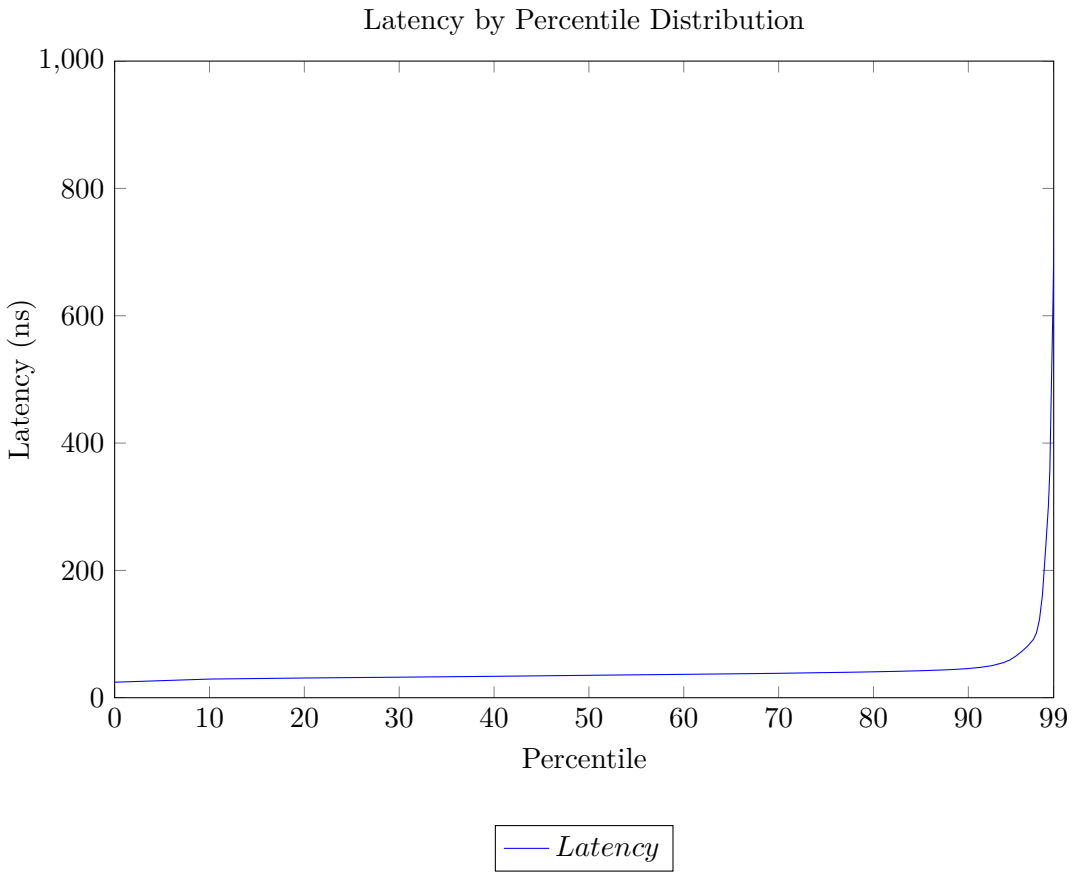


Figure 14: *A High Dynamic Range (HDR) Histogram graph. It shows the latency percentile of Aeron's PingPong test.*

4.6 Limit Order Book Data Structure

The matching engine component stores the active orders in memory. The active orders are not stored on disk as this would increase the I/O and reduce performance.

The LOB data structure was implemented using a B+Tree. There are 4 trees:

- Bid and offer trees for the active orders.
- Parked bid and parked offer trees for the parked orders.

The ObjectLayout library implementation of the B+Tree is used. The key nodes represent the prices and the leaf nodes represent the orders at each price. The leaf nodes are a custom list data structure.

The data structure was selected based on 3 requirements:

1. Execution and storage requirements.
2. Efficient use of CPU cache.
3. Memory access patterns.

4.6.1 Execution and Storage Requirements

The graph in figure 15 shows the daily totals of orders entered, amended, deleted and executed on the JSE from April 2007 to June 2012 [34]. The orders stored in the limit order book was always higher than the executions. The orders entered have increased significantly when compared to the number of executions. The number of executions have remained in the same range from 2007 to 2012. This indicates that more orders will be stored in the LOB without being executed. Therefore the LOB data structure needs to have a low memory overhead. The data structure needs to be efficient in searching, updating and deleting orders.

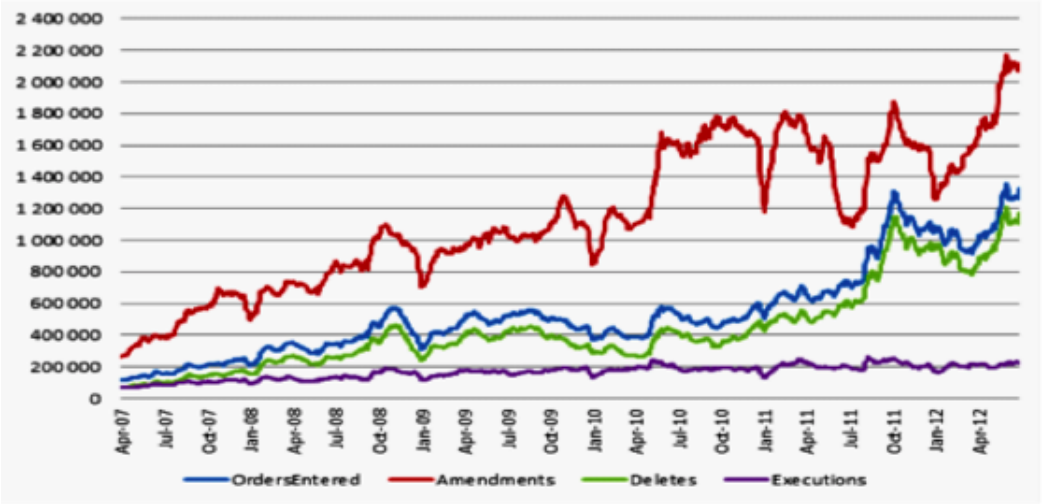


Figure 15: JSE orders [34]

The graph in figure 16 is a snapshot of the Apple Limit Order Book on the Nasdaq at 8:43. The largest number of orders are 5000 at a single price point. The data structure needs to store different quantity of orders at each price point.

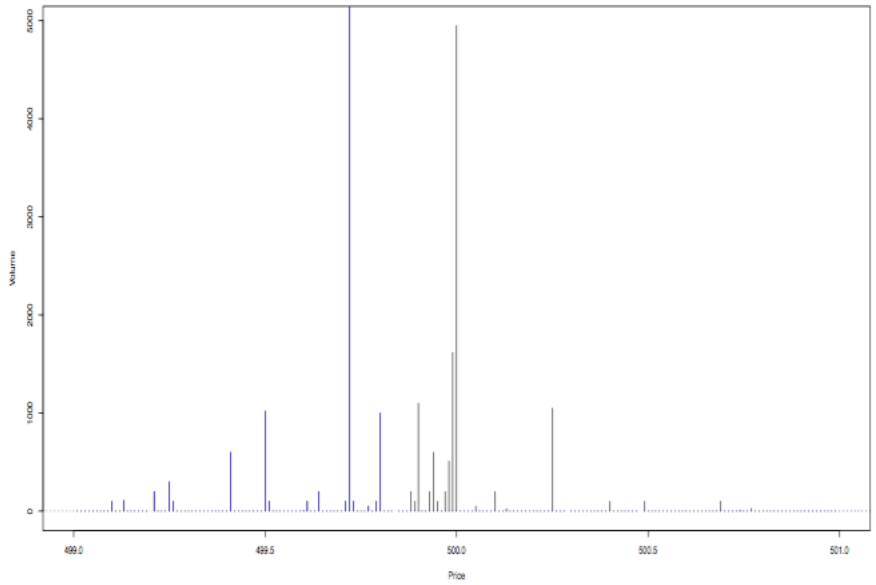


Figure 16: Snapshot of Apple LOB on NASDAQ at 8:43[12]

4.6.2 Efficient use of CPU cache

A software program runs in the operating system's main memory [6, 18]. The operating system increases the available main memory by using virtual memory. Therefore the size of memory available appears more than there is actually available. The virtual memory is space on the hard disk. If a program requires data that is not in the main memory, it fetches it from the virtual memory into the main memory and moves data from the main memory to virtual memory according to the policy configured. The time to fetch data from virtual memory is very slow. The simulator uses only main memory to achieve its low latency.

The latency is reduced between main memory and the CPU by using caches [46]. The caches contain copies of frequently used data from main memory. A cache contains cache lines. When a cache fetches data from main memory it fetches data to fill the entire line. If the CPU finds data in the cache, this is called a cache hit.

When it does not find data, this is called a cache miss. Reducing the cache misses will reduce the latency.

Figure 17 shows the memory layout, cache sizes and the time to retrieve data from each cache. There are 3 caches. An L1, L2 and L3 cache [44]. The L1 cache contains a data cache and an instruction cache. The usual size of the L1, L2 and L3 caches are 64kKB 512KB and 2MB respectively. The time to fetch data from the L1 cache is aprrox. 1 ns while going to main memory is 65 ns.

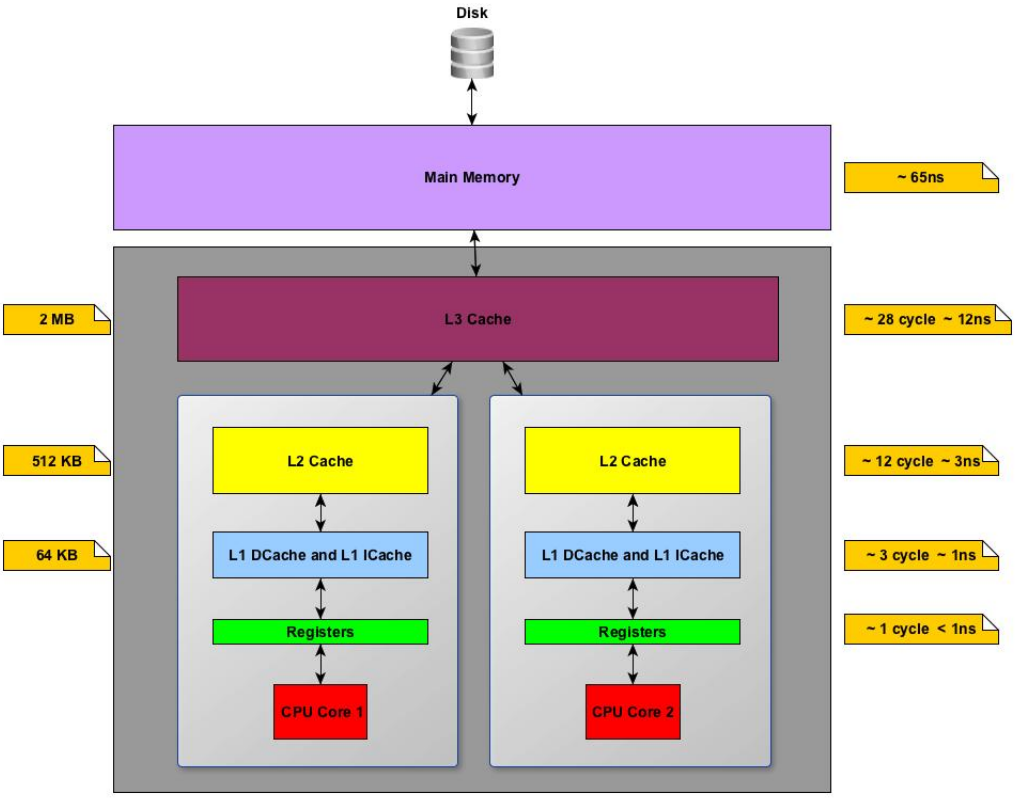


Figure 17: CPU cache

A data structure can be cache aware or cache oblivious [11, 46]. A cache aware data structure is designed to fit the sizes of the cache on the machine it is running. If the application runs on a different machine, it needs to be tuned for that machine. A cache oblivious data structure works on any type of hardware to efficiently use each cache. The simulator is cache aware. Listings 1 and Listing 2 show the cache sizes on the server that the software was tested on.

```
-bash-4.1$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             4
NUMA node(s):          4
Vendor ID:             AuthenticAMD
CPU family:            16
Model:                 2
Stepping:              3
CPU MHz:               2300.058
BogoMIPS:              4601.33
Virtualization:        AMD-V
L1d cache:             64K
L1i cache:             64K
L2 cache:              512K
L3 cache:              2048K
NUMA node0 CPU(s):     0-3
NUMA node1 CPU(s):     4-7
NUMA node2 CPU(s):     8-11
NUMA node3 CPU(s):     12-15
```

Listing 1: CPU Cache on Server

```
-bash-4.1$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE                 65536
LEVEL1_ICACHE_ASSOC                2
LEVEL1_ICACHE_LINESIZE             64
LEVEL1_DCACHE_SIZE                 65536
LEVEL1_DCACHE_ASSOC                2
LEVEL1_DCACHE_LINESIZE             64
LEVEL2_CACHE_SIZE                  524288
LEVEL2_CACHE_ASSOC                 16
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  2097152
LEVEL3_CACHE_ASSOC                 32
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC                 0
LEVEL4_CACHE_LINESIZE              0
-bash-4.1$
```

Listing 2: CPU Cache on Server

4.6.3 Memory Access Patterns

To reduce cache misses, memory in the caches need to be used more efficiently. The higher the cache misses, the more time a program will spend fetching data from main memory. Caches reduce latency by the following 3 properties [44]:

1. *Temporal:* Memory accessed recently will likely be required again soon.
2. *Spatial:* Adjacent memory is likely to be required soon.
3. *Striding:* Memory access is likely to follow a predictable pattern.

The data structure should utilize these 3 properties to be more efficient. Temporal and Spatial are referred to as locality.

4.6.4 Data Structure

Lists, Hash Tables and Trees [48] were considers to represent the limit order book. An array was not considered as this is bounded.

4.6.4.1 Lists

Lists have many pointers and large lists do not generally fit into the cache and result in a large number of cache misses [74].

4.6.4.2 Hash Tables

Hash Tables do not have good locality because accessing keys that are logically next to each other, will usually not be next to each other in memory [74].

4.6.4.3 Trees

Trees have good locality, but only for the top elements. The remaining elements result in a large number of cache misses [74].

4.6.4.4 B-Tree

A B-Tree is the best data structure that has good locality and uses the cache most efficiently. A B-Tree stores the elements in the tree and this would result in not being able to store many elements in the cache [16, 72].

4.6.4.5 B+Tree

A B+Tree only stores the keys in the root or internal nodes, and the leaf nodes store the actual data. Therefore more keys can be stored in the cache. The B+Tree can also be cache friendly if its key size is the same size as the cache line [16, 72]. For a more detailed discussion, refer to [56].

Definition 13. B+Tree [75, 56]

- B+Tree of order n
  - The leaves store all the data items
  - The root node is either a leaf node or has between 2 and n children
  - All internal nodes store up to $n - 1$ keys
  - All internal nodes have between $[n/2]$ and n children
  - All leaf nodes have the same depth
  - All leaf nodes have between $[l/2]$ and l data items
- O(log n) - insert, delete, find
- O(n) - space

The ObjectLayout library B+Tree is used for LOB [80]. It was enhanced to iterate over the elements more efficiently. This library created new data structures that were designed to optimally use the memory layout and to match the speed benefits of similar data structures in most C-style languages. Figure 18 shows the B+Tree with the leaf nodes storing the orders of the LOB at each price point.

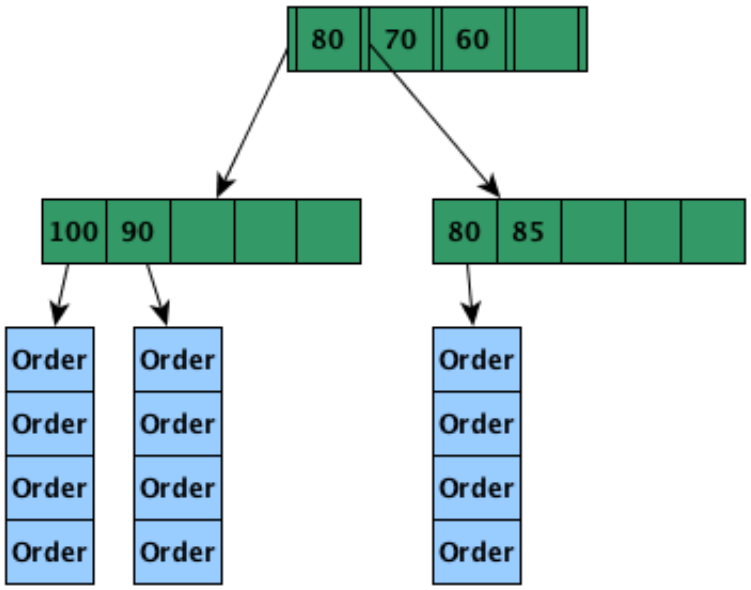


Figure 18: Data Structure

The leaf nodes contain a custom list class called OrderList. The OrderList class contains OrderEntry objects. The OrderEntry objects are created and managed outside the heap. The OrderEntry is implemented using the Java unsafe package. The time spent garbage collecting had to be reduced. Therefore the unsafe package was used as it allows memory to be managed outside the Java heap. The OrderList data structure creates a list of objects in memory and I used the flyweight pattern to access each object. The unsafe package is planned to be removed from Java and be replaced with variable handles in future versions [20]. Most applications use the unsafe package to achieve their performance. The prices are stored as cents and not floating point numbers. Figure 19 shows the OrderEntry objects and it's access via a flyweight object.

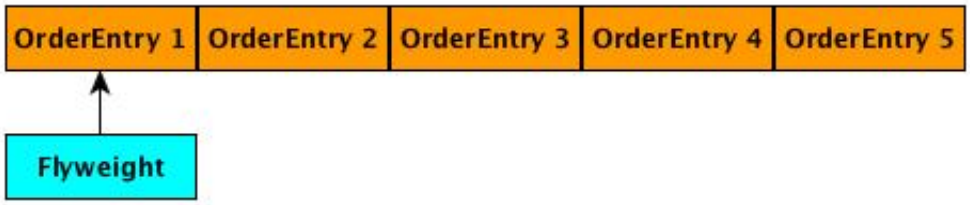


Figure 19: Order List

4.7 Business Rules

When clients send events to the matching engine, the event needs to validated and/or manipulated before applying the matching logic. The event may trigger further logic after the event is matched or added to the limit order book. The Intercepting Filter design pattern was chosen to implement the business rules. The Intercepting Filter intercepts and manipulates a request and a response before and after the request is processed [2]. This pattern was selected because the complex matching and trading session rules could be separated into it's own classes. This will allow for easier unit testing and adding or removing logic.

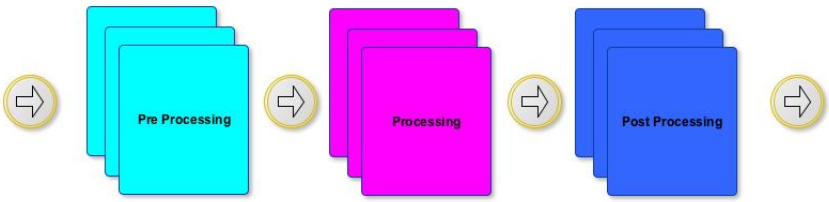


Figure 20: Business Rules

4.8 Website

Managing and monitoring the software is as important as implementing the matching logic. The trading session execution times and monitoring was not implemented in the components as this would reduce the throughput and increase the latency which is against my design goals. A website was developed to implement this type of logic. It reads all events from the file system. The trading session scheduled times generate events which are sent to the simulator. Users can view the limit order book and configure the clients and stocks in the system. The website was developed using Spring Boot and Apache Wicket.

"Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration" ***[78].***

"Apache Wicket is a component oriented framework that differs from classic web frameworks in that it builds a model of requested page on the server side and the HTML sent back to the client is generated according to this model. With this kind of framework the web pages and their HTML components (forms, input controls, links, etc…), are pure class instances. Since pages are class instances they live inside the JVM heap and it is handled like any other Java class" [90]

4.9 Web Event Listener

The original design had the event listener and website in the same Java process. When the website was used or paused because of garbage collection, it affected the receiving and saving of events. Therefore this logic was split into a separate component.

The listener could keep the received events in memory or save it to the file system. Saving the data in memory would be fast, but would require an unknown maximum memory setting. Therefore the data needs to be saved to the file system. The Web Event Listener would have read/write access and the website would have only read access. The single writer principle would improve performance of both components.

I used an off heap hashmap to save the events to memory mapped files. An off heap hashmap stores data outside the Java heap space and is not affected by garbage collection. Off heap data is suited for storing data larger than the current memory and allows sharing of data between JVMs [51]. Memory mapped files allow Java programs to read and write files using only memory while the operating system reads and writes to the file system. This significantly improves performance. The entire file or a part of the file can be loaded into memory. The values in memory will still be written to the file system even if the JVM crashes [71].

I used the MapDB library to store the events. *"MapDB is an open-source (Apache 2.0 licensed), embedded Java database engine and collection framework. It provides Maps, Sets, Lists, Queues, Bitmaps with range queries, expiration, compression, off-heap storage and streaming*" [45]. MapDB can store data off heap in memory or using memory mapped files.

The Web Event Listener receives events faster than it can save it to file. I used the Disruptor to resolve this problem. One thread receives the events and stores in the the Disruptor and another thread saves it to the file system.

4.10 Conclusion

Creating a matching engine requires a detailed understanding of a broad range of topics to achieve its low latency and high throughput. This chapter highlighted the design decisions for each area. The next chapter shows the software that was created.

5 CoinTossX

5.1 Introduction

The requirements and design decisions discussed in the previous chapters were implemented to create CoinTossX. This is a low latency high throughput exchange that allows clients to send orders and receive market data updates. This chapter gives a brief overview of CoinTossX.

5.2 Technical Documentation

The technical documentation that describes the architecture, Java modules, class, use case and deployment diagrams is covered in the Operational Concept Description document [76]. This document also describes how to use and maintain the software.

5.3 Website Screenshots

5.3.1 Splash Screen

Users can access the CoinTossX website to view and maintain the configuration of the software. The splash screen is the first page the users will see. It is split across three images.

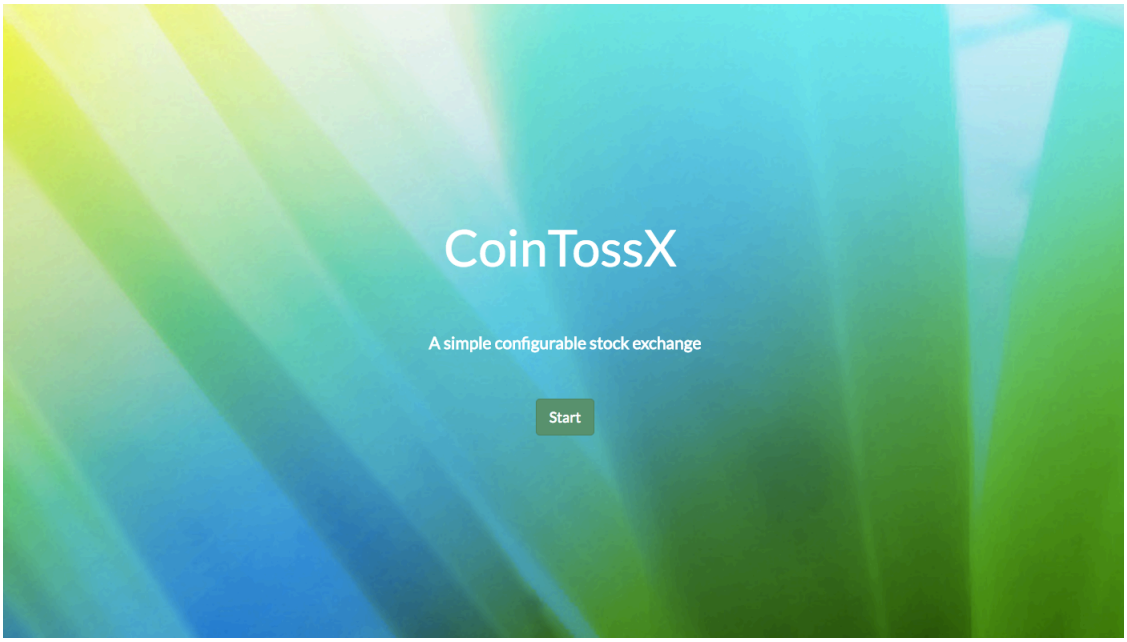
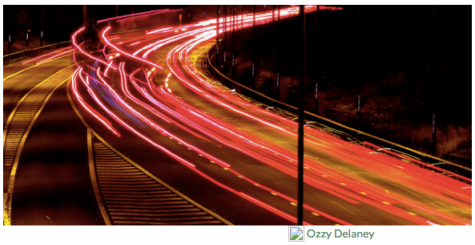


Figure 21: Splash Screen 1



Performance

Low Latency High Throughput

A low latency stock exchange using Aeron and Simple Binary Protocol (SBE) to send and receive messages from clients.

The Matching Engine stores the limit order book off heap to reduce the GC pauses.

Simulation

Hawkes Simulation

We propose a feasible scheme for the simulation of an asynchronous stream of limit order book events. An n-variate mutually-exciting Hawkes process is used to govern the times of coupled liquidity demand and supply events, while trade and quote prices and volumes are generated consistent with the event type. This provides a exible framework to simulate a market data feed with varying throughput, with full control over the trade and quote conditional intensities.

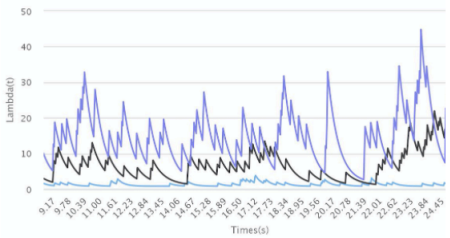


Figure 22: Splash Screen 2

Witwatersrand University

Research

CoinTossX was created in collaboration with the Advanced Mathematics in Finance, the QuERI Lab Research Group and Mathematical Science Support services at the University of the Witwatersrand, Johannesburg. Find more information about the QuERI Lab Research group at Blogspot

Figure 23: Splash Screen 3

5.3.2 Stocks

The stock screen shows the stocks configured, the trading session that is active for each stock and a button to view the limit order book of the stock.

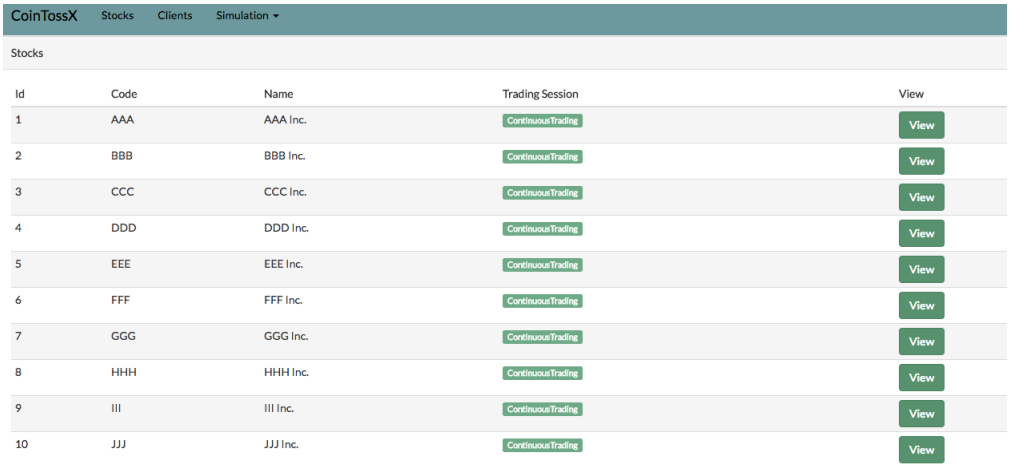


Figure 24: Stocks

5.3.3 Clients

The client screen shows the clients that are configured. It allows users to add or edit clients.

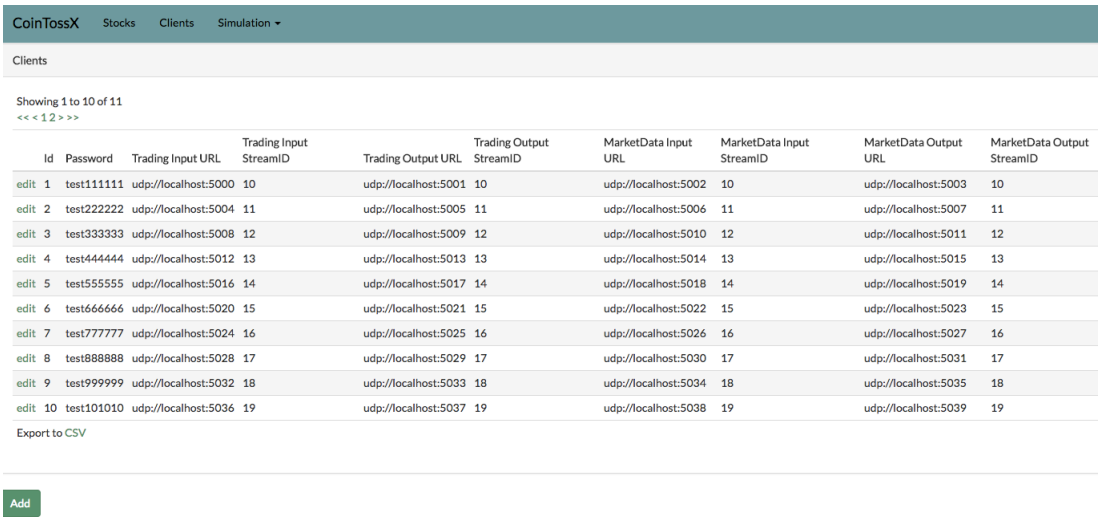


Figure 25: Clients

5.3.4 Limit Order Book

The limit order book screen shows the graph of the bid and offers in the LOB. It lists the details of the bids, offers, orders submitted and trades in separate tables. The data in the tables can be exported.

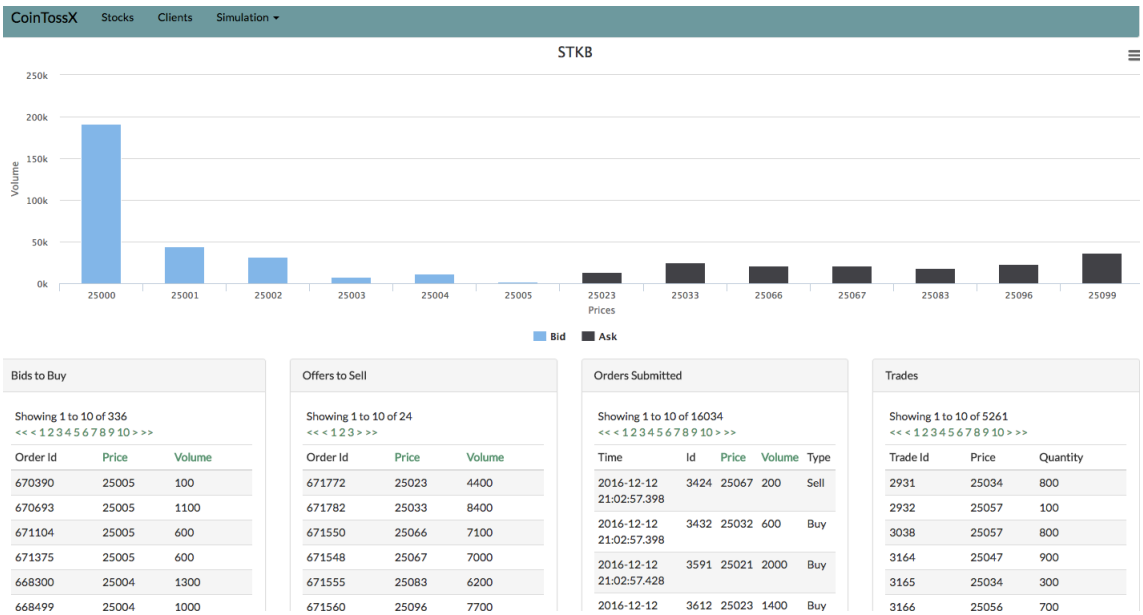


Figure 26: Limit Order Book

5.3.5 Hawkes Configuration

The Hawkes configuration page allows the user to change the values of the Hawkes input data before running the simulation testing.

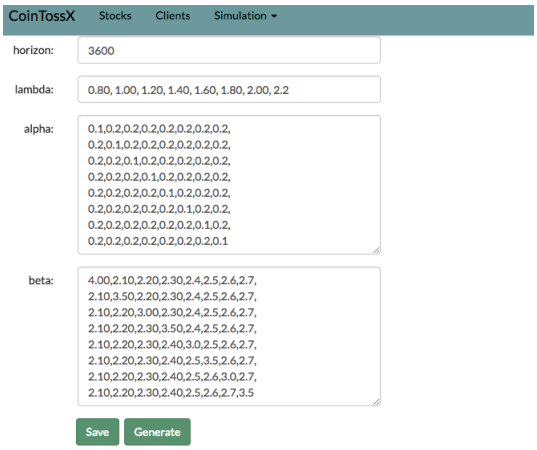


Figure 27: Hawkes Configuration

5.3.6 Run Simulation

The Simulation page allows the user to stop and start the warmup process and the Hawkes simulation. It shows the status of each client and the active trading session.

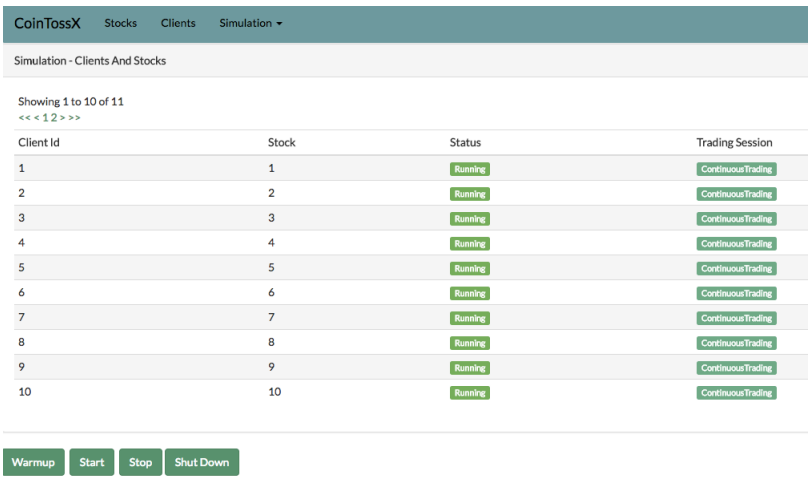


Figure 28: Hawkes Simulation

5.4 Conclusion

CoinTossX was built to process millions of orders and store data to be analyzed by the users. The next chapter describes the Hawkes testing methodology used to test the software.

6 Testing Methodology

6.1 Introduction

To test the software's performance and functionality, orders need to be sent to the Trading Gateway and the Market Data Gateway needs to publish updates from the exchange. A Hawkes simulation is used to generate the orders. This chapter gives a review of Hawkes processes and its use in modeling financial data.

6.2 Hawkes Process

Hawkes [31] introduced a new point process model in 1971 to model the frequency of earthquakes called the Hawkes process. He considers a linear "self-exciting" process with an exponential decay kernel. This means that as events arrive the rate of future events increase for a specific time period. The rate at which this effect decays is defined by the decay function. A Hawkes process can be defined using a Poisson cluster process with a particular branching structure or as a conditional intensity function. It can be further defined as univariate or multivariate and marked or unmarked. A mark is some additional value associated with each point. This counting process was first used by Ogata [65] in seismology and since then it has been used across many fields such as eismology, epidemiology, neurophysiology and network modeling [17]. Toke [87, 86] provides the following definitions:

Definition 14. Point Process [87, 86]

Let $(t_i)_{i\epsilon\mathbb{N}^*}$ *be a sequence of non-negative random variables on some probability space* $(\Omega, \mathcal{F}, \mathcal{P})$ *such that* $\forall i\epsilon\mathbb{N}^*, t_i < t_i + 1$. *The sequence* $(t_i)_{i\epsilon\mathbb{N}^*}$ *is called a simple point process on* $\mathbb{R}_+$.

In this research the point process will represent the arrival of orders to the limit order book.

Definition 15. Linear self-exciting process [87, 86]

Let $N(t)$ *be a point process with a Filtration* $\digamma_t, t \geq 0$. *Then a Linear self exciting process has the intensity*

$$\lambda(t) = \lambda_0(t) + \int_{\infty}^{t} \nu(t-s)dN_s = \lambda_0(t) + \sum_{t_i<t} \nu(t-t_i) \tag{1}$$

where:

1. $\lambda_0 : \mathbb{R} \to \mathbb{R}_+$ *is the base intensity*
2. $\nu : \mathbb{R} \to \mathbb{R}_+$ *expresses the positive influence of the past events* t_i *on the current value of the intensity process.*
3. dN_s *represents the difference in arrival times of the events.*

This self excitation process generates a new intensity based on the past intensity.

Definition 16. Mutually Exciting Hawkes Process [87, 86]

Let $\left\{(t_i^m)_i\right\}_{m=1...M}$ *be a M-dimensional point process and* $\left(N_t^1...N_t^M\right)$ *be the associated counting process. Then a linear multi-dimensional Hawkes process has intensities* $\lambda^m, m = 1, ..., M$ *given by*

$$\lambda^m(t) = \lambda_0^m(t) + \sum_{n=1}^{M} \int_0^t \sum_{j=1}^{P} \alpha_j^{mn} e^{-\beta_j^{mn(t-s)}} dN_s^n \tag{2}$$

where:

1. λ_0 *is the deterministic base intensity.*
2. $\alpha_j^{mn} e^{-\beta_j^{mn(t-s)}}$ *is the exponential kernel proposed by Hawkes.*
3. α *increases the intensity of arrival of events.*
4. β *is the rate at which the intensity decreases back to the base intensity.*
5. P *allows for multiple kernels to be included but in this paper, we assume the simplest form,* $P = 1$*.*
6. M *allows for different types of events within a point process with each event having its own base intensity.*

This mutually exciting process generates a new intensity based on the past intensity of all events.

6.3 Hawkes Process In Financial Modeling

Hawkes process have become popular in high frequency finance [3]. Bowsher[10] used the Hawkes model to develop a continuous time econometric modeling framework for multivariate market event data. The Hawkes model was also used to model arrival times of orders and clustering of events in a limit order book [49, 87]. More applications of the Hawkes processes in finance can be found in [3] in which they describe how it has been used in estimating the volatility at the transaction data level, estimating the stability of the market, understanding systemic risk models the contagion effect, creating optimal execution strategies or recording the behaviour of the order book. Toke and Pomponio [88] show how a bivariate Hawkes process fit their empirical observations of trades-through *i.e.* the transactions that were processed extended to the 2nd level of the order book.

6.3.1 Modeling A Stream Of Orders

As mention earlier, Large [49] streamed orders into a limit order book. He measured the resiliency of the LOB. A LOB is resilient if it can go back to its initial state or replenish itself after a large order has removed liquidity from it. He measure the time it takes for orders to be added back into the LOB after the execution of the large order. He identified 10 different types of orders:

6.3.1.1 Type 1

An aggressive market buy order that affects the best ask level of the LOB. It changes the best ask price of the LOB by removing the price from the LOB. This may affect other ask levels.

6.3.1.2 Type 2

An aggressive market sell order that affects the best bid level of the LOB. It changes the best bid price of the LOB by removing the price from the LOB. This may affect other bid levels.

6.3.1.3 Type 3

An aggressive limit buy order that affects the best bid level of the LOB. It increases the best bid price of the LOB by adding a new price to the LOB.

6.3.1.4 Type 4

An aggressive limit sell order that affects the best ask level of the LOB. It decreases the best ask price of the LOB by adding a new price to the LOB.

6.3.1.5 Type 5

A passive market buy order that affects the best ask level of the LOB. It adds an order to the LOB that is at the best ask price. The best ask price does not change as the volume is not consumed.

6.3.1.6 Type 6

A passive market sell order that affects the best bid level of the LOB. It adds an order to the LOB that is at the best bid price. The best bid price does not change as the volume is not consumed.

6.3.1.7 Type 7

A passive limit buy order that does not affect the best bid level of the LOB. It adds an order to the LOB below the best bid price.

6.3.1.8 Type 8

A passive limit sell order that does not affect the best ask level of the LOB. It adds an order to the LOB above the best ask price

6.3.1.9 Type 9

A cancel order that cancels a bid order.

6.3.1.10 Type 10

A cancel order that cancels an ask order.

The types of orders mentioned above have a relationship with each other. As one type of order is sent, it affects the the occurrence of the next order. Toke [87] made the following conclusions about the relationship of the orders:

- Passive limit orders can be seen as “background noise”
- Aggressive limit orders are first influenced by aggressive market orders, then by passive market orders and lastly by aggressive limit orders.
- Aggressive market orders are first influenced by passive market orders, then by aggressive market orders and lastly by aggressive limit orders.
- Passive market orders are influenced by passive and aggressive market orders. These orders are not influenced by limit orders

6.4 Simulating A Hawkes Process

The most popular way to simulate a Hawkes process is to use the intensity-based thinning method that was introduced by Lewis [52] and later modified by Ogata [64]. The Ogata modified thinning algorithm simulates a homogeneous Poisson process using a conditional intensity function. The algorithm generates high intensities and then thins out the points.

Theorem 17. *The Basic Thinning Theorem [52]*

Consider a one-dimensional non-homogeneous Poisson process $N^*(t)_{t\geq 0}$ *with rate function* $\lambda^*(t)$*, so that the number of points* $N^*(T_0)$ *in a fixed interval* $(0, T_0]$ *has a Poisson distribution with parameter* $\mu_0^* = \int_0^{T_0} \lambda^*(s)ds$*. Let* $t_1^*, t_2^*, ..., t_{N^*(T_0)}^*$ *be the points of the process in the interval* $(0, T_0]$*. Suppose that for* $0 \leq t \leq T_0, \lambda(t) \leq \lambda^*(t)$*.*

For $i = 1, 2, ..., N^*(T_0)$*, delete the points* t_i^* *with probability* $1 - \frac{\lambda(t_i^*)}{\lambda^*(t_i^*)}$ *. Then the remaining points form a non-homogeneous Poisson process* $N(t)_{t\geq 0}$ *with rate function* $\lambda(t)$ *in the interval* $(0, T_0]$*.*

Toke [87] defined the thinning algorithm as:

Let $U_{[0,1]}$ denote the uniform distribution on the interval $[0, 1]$ and $[0, T]$ the time interval on which the Hawkes process defined by equation 2 is to be simulated. We define $I^K(t) = \sum_{n=1}^{K} \lambda^n(t)$ the sum of the intensities of the first K components of the multivariate process. The algorithm is written as follows.

Algorithm 13 Step 1 - Initialization

1: **Set** $i \leftarrow 1, i^1 \leftarrow 1, \ldots, i^M \leftarrow 1$ and
2: $I* \leftarrow I^M(0) = \sum_{n=i}^{M} \lambda_0^i(0)$

Algorithm 14 Step 2 - First event

1: **Generate** $U \sim U_{[0,1]}$ and set $s \leftarrow -\frac{1}{\lambda*} \ln U$
2:
3: **if** s > T **then**
4: go to step 4
5: **end if**
6:
7: **Attribution Test:** Generate $D \sim U_{[0,1]}$ and
8: set $t_1^{n_0} \leftarrow s$ where n_0 is such that $\frac{I^{n_0-1}(0)}{I*} < D \leq \frac{I^{n_0}(0)}{I*}$
9:
10: Set $t_1 \leftarrow t_1^{n_0}$

Algorithm 15 Step 3 - General routine

Set $i^{n_0} \leftarrow i^{n_0} + 1$ and $i \leftarrow i + 1$
Update maximum intensity: Set $I* \leftarrow I^M(t_{i-1}) + \sum_{n=1}^{M} \sum_{j=1}^{P} \alpha_j^{nn_0}$

New event: Generate $U \sim U_{[0,1]}$ and $s \leftarrow s - \frac{1}{I*} \ln U$

if s > T **then**
  go to step 4
end if

Attribution-Rejection test: Generate $D \sim U_{[0,1]}$
if $D \leq \frac{I^M(s)}{I*}$ **then**
  set $t_{i^{n_0}}^{n_0} \leftarrow s$ where n_0 is such that $\frac{I^{n_0-1}(s)}{I*} < D \leq \frac{I^{n_0}(s)}{I*}$, and $t_i \leftarrow t_{i^{n_0}}^{n_0}$ and go through the general routine again
else
  update $I^* \leftarrow I^M(s)$ and try a new date at step (b) of the general routine
end if

Algorithm 16 Step 4 - Output

Retrieve the simulated process $(\{t_i^n\}\, i)_{n=1,\ldots,M}$ on $[0, T]$

Nchab [63] used the above algorithm to simulate an 8-variate mutually-exciting Hawkes process. He generated the first 8 order types mentioned by Large with the corresponding arrival times.

6.5 Conclusion

The literature shows that the Hawkes model has been used to simulate a realistic order flow using different order types. These orders have not been sent in real time to an exchange. This dissertation is able to use CoinTossx to configure the Hawkes simulation and process the orders. For a more detailed discussion on this simulation refer to [77]. The next chapter describes the algorithm used to simulate orders to CoinTossX.

7 Hawkes Simulation

A simulation was developed to test a synchronous stream of limit order book events. An 8-variate mutually-exciting Hawkes process is used to govern the times of coupled liquidity demand and supply events, while trade and quote prices and volumes are generated consistent with the event type. This provides a flexible framework to simulate a market data feed with varying throughput, with full control over the trade and quote conditional intensities.

7.1 Hawkes Algorithm

The Hawkes algorithm was ported from the Hawkes R package to Java [92]. This allows the simulation to be portable across different hardware. The code connected to the Trading and Market Data Gateways. The Hawkes simulation generated the times to send the orders. To simulate the intensity, the generated times was used to delay the current thread. The shorter the delay, the higher the intensity. The input to the Hawkes process was used from [63].

$$u = \begin{bmatrix} 0.80 & 1.00 & 1.20 & 1.40 & 1.60 & 1.80 & 2.00 & 2.2 \end{bmatrix} \tag{3}$$

$$\alpha = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.1 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.1 \end{bmatrix} \tag{4}$$

$$\beta = \begin{bmatrix} 4.00 & 2.10 & 2.20 & 2.30 & 2.4 & 2.5 & 2.6 & 2.7 \\ 2.10 & 3.50 & 2.20 & 2.30 & 2.4 & 2.5 & 2.6 & 2.7 \\ 2.10 & 2.20 & 3.00 & 2.30 & 2.4 & 2.5 & 2.6 & 2.7 \\ 2.10 & 2.20 & 2.30 & 3.50 & 2.4 & 2.5 & 2.6 & 2.7 \\ 2.10 & 2.20 & 2.30 & 2.40 & 3.0 & 2.5 & 2.6 & 2.7 \\ 2.10 & 2.20 & 2.30 & 2.40 & 2.5 & 3.5 & 2.6 & 2.7 \\ 2.10 & 2.20 & 2.30 & 2.40 & 2.5 & 2.6 & 3.0 & 2.7 \\ 2.10 & 2.20 & 2.30 & 2.40 & 2.5 & 2.6 & 2.7 & 3.5 \end{bmatrix} \tag{5}$$

7.2 Algorithm for Price and Volume Generator

A client application connects to the Trading and MarketData Gateways of the simulator. Orders are sent to the Trading Gateway and the client subscribes to market data events from the MarketData Gateway. The generated Hawkes times are used to pause the application to simulated the intensity or orders sent. A new order is sent to the simulator when it receives a market data update. A mutex is used by the client application and it's market data subscriber guarantee this behavior. The order type, price and volume is generated based on the type of event.

1. Let $L(t)$ represent all active orders in a LOB at time t
2. Let $b(t)$ and $a(t)$ be the highest bid and lowest ask price at time t
3. Let $bv(t)$ and $av(t)$ be the volume at the highest bid and lowest ask price at time t
4. Let $x = (p_x, t_x, v_x)$ be an order with price p_x at time $t_x$with volume v_x
5. Let $N_{[\mu,\sigma]}$ denote the normal distribution over the interval [mean,std]
6. Let M be the maximum LOB depth level permitted.

7.3 Initialization

1. M = 10
2. Subscribe to Trading Gateway (TG).
3. Subscribe to Market Data Gateway (MDG).
4. Login to Trading Gateway.
5. Initialize the LOB by sending a limit buy and sell order to the simulator.
   (a) $I_b \leftarrow 25034$ where I_b is the initial limit buy order price
   (b) $I_s \leftarrow 25057$ where I_s is the initial limit sell order price
   (c) $L_b \leftarrow 25000$ where L_b is the lowest buy price that will be generated by the simulator
   (d) $H_s \leftarrow 25057$ where H_s is the highest sell price that will be generated by the simulator
6. Wait for a bid and ask update from the MDG.

7.4 General Routine - Client

Algorithm 17 General Routine - Client

```
timeArray = hawkes simulation time array
for i = 0 to size(timeArray) -1 do
    acquire mutex
    pause for duration(time[i+1] - time[i])
    generate order x
    send order x to TG
    if order x is not sent to TG then
        release mutex
    end if
end for
```

7.5 General Routine - Client MarketData Subscriber

Algorithm 18 General Routine - Client MarketData Subscriber

```
subscribe to market data events
if event arrives then
    update b(t), bv(t), a(t) and av(t)
    release mutex
end if
```

7.6 VWAP

Algorithm 19 Volume Weighted Average Price (VWAP)

$VWAP \leftarrow \frac{\sum_{i=1}^{k} Price_i * Volume_i}{\sum_{i=1}^{k} Volume_i}$ where k is the highest level affected by the aggressive trade

Volume Weighted Average Price (VWAP) is used to calculate the price for an aggressive buy/sell trade.

7.7 Aggressive Buy Trade (Type 1)

Algorithm 20 Aggressive Buy Trade (Type 1)

1: $p(t) \leftarrow VWAP$
2: $v(t) \sim N_{[av(t),1000]}$ with $v(t) \geq av(t)$ and $v(t) \,\%\, 100 = 0$
3:
4: **if** $p(t) > 0 \, and \, v(t) > 0$ **then**
5: **return** A buy market order with price $p(t)$ and volume $v(t)$ at time t
6: **else**
7: **return** null {Order is not sent to TG}
8: **end if**

An aggressive buy trade is an order submitted that affects the best ask level of the LOB. It changes the best ask price of the LOB by removing the price from the LOB. The generated order has a price obtained using VWAP and a volume that is $\geq av(t)$. The volume generated is divisible by 100 to create orders with large volumes.

7.8 Aggressive Sell Trade (Type 2)

Algorithm 21 Aggressive Sell Trade (Type 2)

1: $p(t) \leftarrow VWAP$
2: $v(t) \sim N_{[bv(t),1000]}$ with $v(t) \geq bv(t)$ and $v(t) \,\%\, 100 = 0$
3:
4: **if** $p(t) > 0 \, and \, v(t) > 0$ **then**
5: **return** A sell market order with price $p(t)$ and volume $v(t)$ at time t
6: **else**
7: **return** null {Order is not sent to TG}
8: **end if**

An aggressive sell trade is an order submitted that affects the best bid level of the LOB. It changes the best bid price of the LOB by removing the price from the LOB. The generated order has a price that is obtained using VWAP and a volume that is $\geq bv(t)$. The volume generated is divisible by 100 to create orders with large volumes.

7.9 Aggressive Buy Quotes (Type 3)

Algorithm 22 Aggressive Buy Quotes (Type 3)

if $b(t) \neq 0 \, and \, a(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $a(t) \geq p(t) > b(t)$ and $p(t) \geq L_b$

else if $b(t) \doteq 0 \, and \, a(t) \neq 0$ **then**
  $p(t) \sim N_{[a(t),200]}$ where $a(t) \geq p(t) > b(t)$ and $p(t) \geq L_b$

else if $b(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $p(t) > b(t)$ and $p(t) \geq L_b$

else
  $p(t) \sim N_{[I_b,200]}$ where $p(t) > b(t)$ and $p(t) \geq L_b$

end if

$v(t) \sim N_{[bv(t),1000]}$ with $v(t) \geq bv(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0 \, and \, v(t) > 0$ **then**
  return A buy limit order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

An aggressive buy quote is an order submitted that affects the best bid level of the LOB. It increases the best bid price of the LOB by adding a new price to the LOB. If the new price increases the LOB depth to be greater than N, than the new price is not added to the LOB. The price is generated based on $b(t)$ and $a(t)$. If the LOB is empty, then the price is generated using I_b.

7.10 Aggressive Sell Quotes (Type 4)

Algorithm 23 Aggressive Sell Quotes (Type 4)

if $b(t) \neq 0$ *and* $a(t) \neq 0$ **then**
  $p(t) \sim N_{[a(t),200]}$ where $a(t) > p(t) \geq b(t)$ and $p(t) \leq H_s$

else if $b(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $p(t) \geq b(t)$ and $p(t) \leq H_s$

else
  $p(t) \sim N_{[I_s,200]}$ where $p(t) \geq b(t)$ and $p(t) \leq H_s$

end if

$v(t) \sim N_{[av(t),1000]}$ with $v(t) \geq bv(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0$ *and* $v(t) > 0$ **then**
  return A sell limit order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

An aggressive sell quote is an order submitted that affects the best ask level of the LOB. It decreases the best ask price of the LOB by adding a new price to the LOB. If the new price increases the LOB depth to be greater than N, than the new price is not added to the LOB. The price is generated based on $b(t)$ and $a(t)$. If the LOB is empty, then the price is generated using I_s.

7.11 Passive Buy Trade (Type 5)

Algorithm 24 Passive Buy Trade (Type 5)

$p(t) \leftarrow a(t)$
$v(t) \sim N_{[av(t),1000]}$ with $0 < v(t) < av(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0$ *and* $v(t) > 0$ **then**
  return A buy market order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

A passive buy trade is an order submitted that affects the best ask level of the LOB. It adds an order to the LOB that is at the best ask price. The generated order has a price that matches $a(t)$ and a volume that is $< av(t)$. The volume generated is divisible by 100 to create orders with large volumes.

7.12 Passive Sell Trade (Type 6)

Algorithm 25 Passive Sell Trade (Type 6)

$p(t) \leftarrow b(t)$
$v(t) \sim N_{[bv(t),1000]}$ with $v(t) < bv(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0 \, and \, v(t) > 0$ **then**
  return A sell market order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

A passive sell trade is an order submitted that affects the best bid level of the LOB. It adds an order to the LOB that is at the best bid price.. The generated order has a price that matches $b(t)$ and a volume that is $< bv(t)$. The volume generated is divisible by 100 to create orders with large volumes.

7.13 Passive Buy Quotes (Type 7)

Algorithm 26 Passive Buy Quotes (Type 7)

if $b(t) \neq 0 \, and \, a(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $a(t) \geq p(t) > b(t)$ and $p(t) \geq L_b$

else if $b(t) \doteq 0 \, and \, a(t) \neq 0$ **then**
  $p(t) \sim N_{[a(t),200]}$ where $a(t) \geq p(t)$ and $p(t) \geq L_b$

else if $b(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $p(t) < b(t)$ and $p(t) \geq L_b$

else
  $p(t) \sim N_{[I_b,200]}$ where $H_s \geq p(t) \geq L_b$

end if

$v(t) \sim N_{[bv(t),1000]}$ with $v(t) \geq bv(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0 \, and \, v(t) > 0$ **then**
  return A buy limit order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

A passive buy quote is an order submitted that does not affect the best bid level of the LOB. It adds an order to the LOB below the best bid price. If the new price

increases the LOB depth to be greater than N, than the new price is not added to the LOB. The price is generated based on $b(t)$ and $a(t)$. If the LOB is empty, then the price is generated using I_b.

7.14 Passive Sell Quotes (Type 8)

Algorithm 27 Passive Sell Quotes (Type 8)

if $b(t) \neq 0 \, and \, a(t) \neq 0$ **then**
  $p(t) \sim N_{[a(t),200]}$ where $p(t) > a(t)$ and $p(t) \leq H_s$

else if $b(t) \neq 0$ **then**
  $p(t) \sim N_{[b(t),200]}$ where $p(t) > b(t)$ and $p(t) \leq H_s$

else
  $p(t) \sim N_{[I_s,200]}$ where $L_b \leq p(t) \leq H_s$

end if

$v(t) \sim N_{[av(t),1000]}$ with $v(t) \geq bv(t)$ and $v(t) \,\%\, 100 = 0$

if $p(t) > 0 \, and \, v(t) > 0$ **then**
  return A sell limit order with price $p(t)$ and volume $v(t)$ at time t
else
  return null {Order is not sent to TG}
end if

A passive sell quote is an order submitted that does not affect the best ask level of the LOB. It adds an order to the LOB above the best ask price. If the new price increases the LOB depth to be greater than N, than the new price is not added to the LOB. The price is generated based on $b(t)$ and $a(t)$. If the LOB is empty, then the price is generated using I_s.

7.15 Intensity Charts

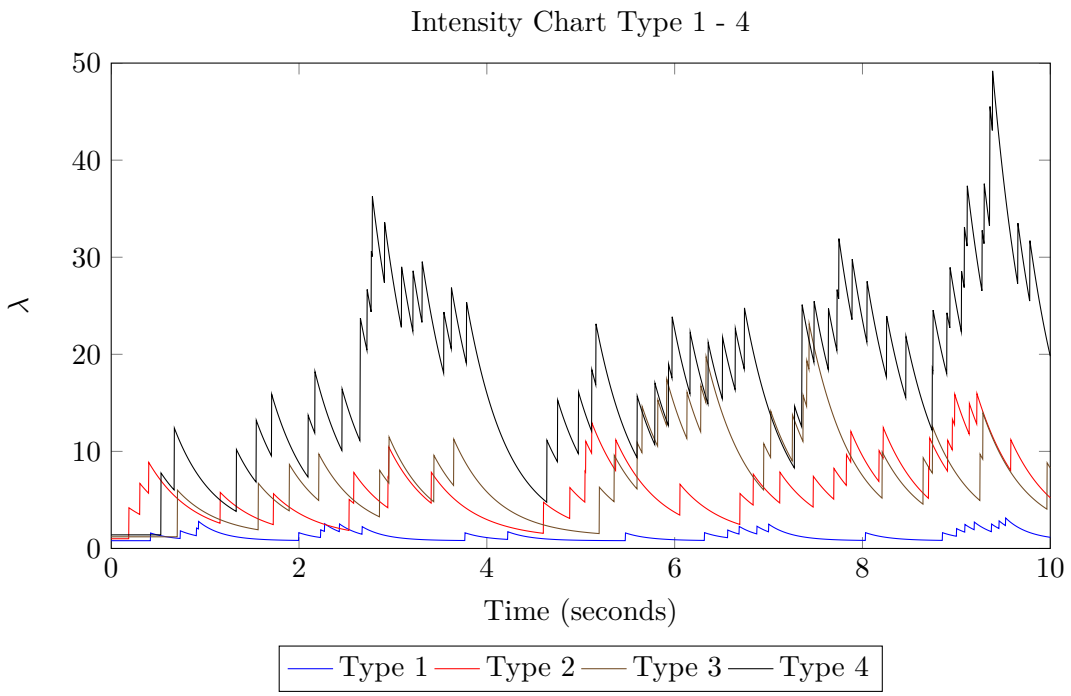


Figure 29: *Intenisty of Hawkes process for type 1 - 4 event types and using the paramters from equations 3,4,5*

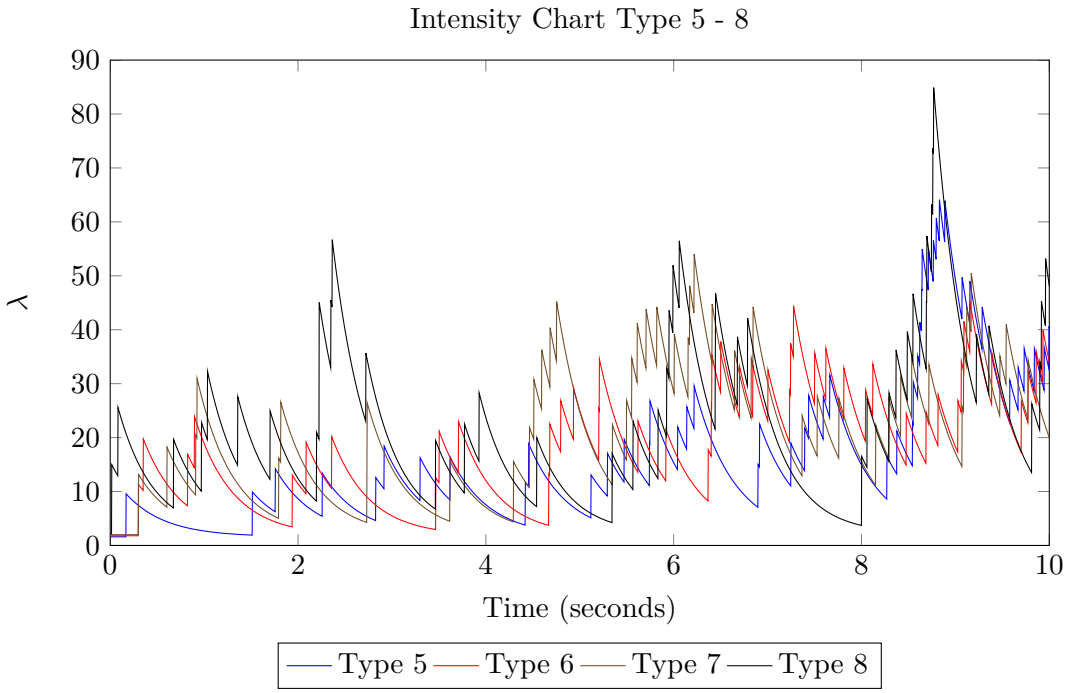


Figure 30: *Intenisty of Hawkes process for type 5 - 8 event types and using the paramters from equations 3,4,5*

8 Test Analysis

8.1 Introduction

This chapter describes the functional and performance tests that were run to evaluate the software. It covers the unit tests and the Hawkes testing scenarios using the algorithms from the previous chapter.

8.2 Unit Tests

Unit tests were created for most of the classes in the software. The unit tests for the matching engine is cover in Appendix B. These are detailed test cases for each of the requirements in Chapter 3. The remaining units tests can be accessed from the source code. These tests show the impact on the LOB when an order is processed. It lists the following details:

1. The stock configuration
2. The initial state of the LOB
3. The aggressive order
4. The final state of the LOB
5. The trades executed

8.3 Java Microbenchmark Harness (JMH)

Performance testing of individual methods is required to implement efficient algorithms and logic. This type of performance testing is difficult to do as the method requires to be warmed up before the actual test can be run and it needs to cater for the behaviour of the JVM [36]. The software used the Java Microbenchmark Harness (JMH) to test methods that had performance problems. *"JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targetting the JVM"*[68].

8.4 Performance Tests

The matching engine has been designed for multiple clients and stocks. The matching algorithms are determined by the trading sessions. Test scenarios were created to test the performance of the software by combing these features.

1. **Scenario A**: This scenario tests the impact of multiple clients and stocks. Each client sends orders to a single stock. Each run increased the number of clients and stocks. The Continuous Trading session was the only session used during the test.

2. **Scenario B**: This scenario tests the impact of the different trading sessions. A one minute Intraday Auction Trading session was executed during the test. The Continuous Trading session was used before and after the Intraday Auction Trading session.

8.4.1 Scenario A: Throughput Testing

Table 7 shows the throughput per second (TPS) test results. Each client waits for market data updates before calculating the next order to send. The client also waits a few nano seconds as part of the Hawkes simulation. These delays reduced the throughput of orders sent. As the number of clients and stocks increase, the throughput decreases. The most significant decrease in throughput is when more than 1 client and stock is used.

	Client	**Stock**	**Start Time**	**End Time**	**Duration**	**Orders**	**TPS**
A1	1	1	01:25:00.516	01:25:49.336	00:00:48.820	111646	2287
A2	2	2	01:30:32.741	01:34:56.722	00:04:23.981	224562	850
A3	4	4	04:12:43.498	04:24:45.702	00:12:02.204	448774	621
A4	6	6	04:31:57.914	04:47:10.300	00:15:12.386	669331	733
A5	8	8	04:56:01.288	05:16:28.298	00:20:27.010	895080	729
A6	10	10	05:21:48.718	05:40:41.007	00:18:52.289	1120514	989

Table 7: Scenario A: Throughput

8.4.2 Scenario A: Latency Testing

The latency was tested using the HdrHistogram library. Figure 31 shows the latency using the test runs from table 7. The different lines show the percentile latency of each run. The latency increases as the number of clients and stocks increase. The min and max latency is 99 ns and 353 ns at the 90th percentile. This means that the matching engine keeps a low latency below 360 ns as the number of orders submitted increases.

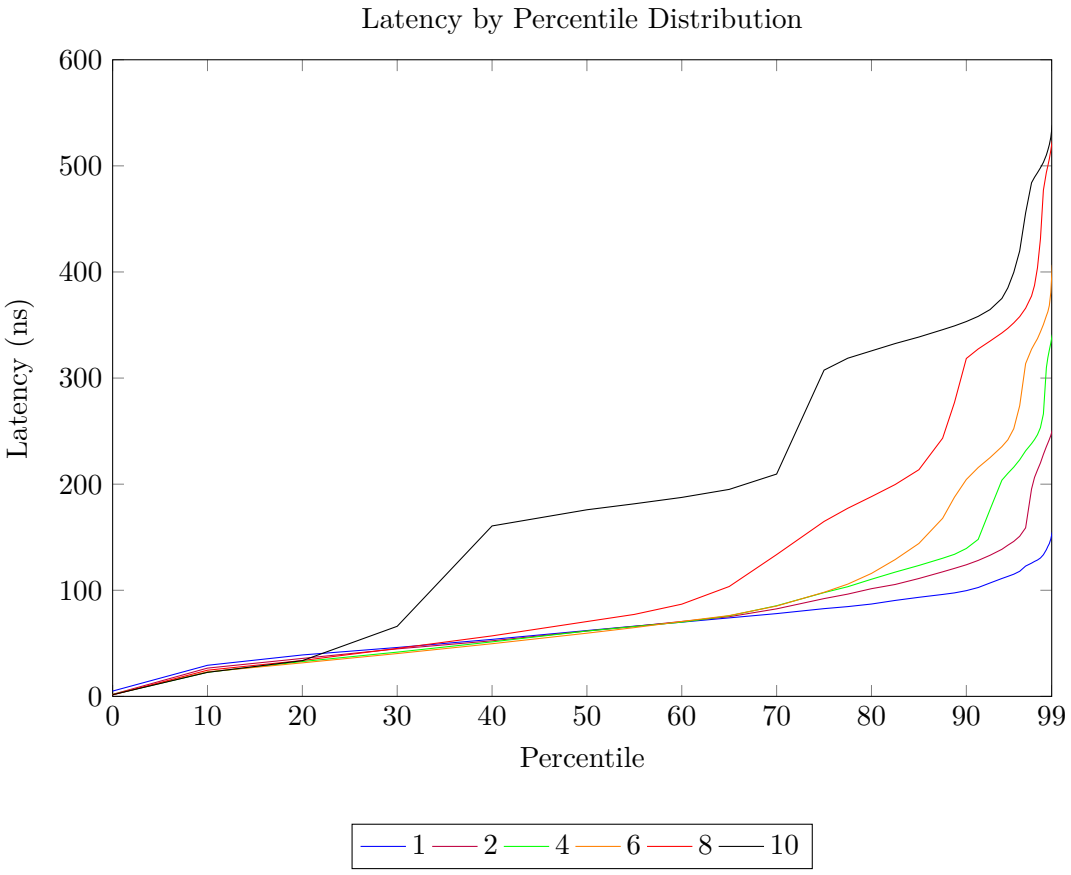


Figure 31: *A High Dynamic Range (HDR) Histogram graph. It shows the latency percentile of the matching engine. It compares the latency across mutiple test runs using a different number of clients and stocks.*

8.4.3 Scenario A: Limit Order Book Storage Testing

The figures shown in 32 and 33 show the view of the STKJ stock from Test A6. The limit order book is able to store thousands of orders at each price point. The design of the LOB is allows orders to be added and removed easily.

Figure 34 shows the number of trades executed in 1 minute buckets. The matching engine is able to hold and publish thousands of trades to the Market Data Gateway.

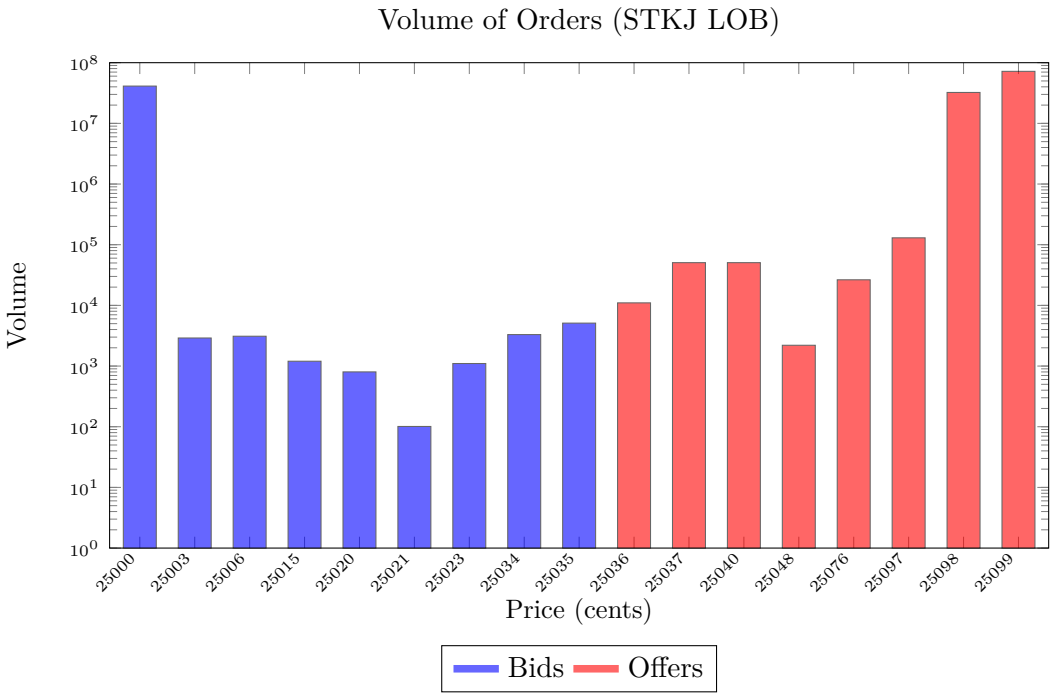


Figure 32: *Limit Order Book of stock STKJ. It shows the volume of the bids to buy and offeres to sell at each price point*

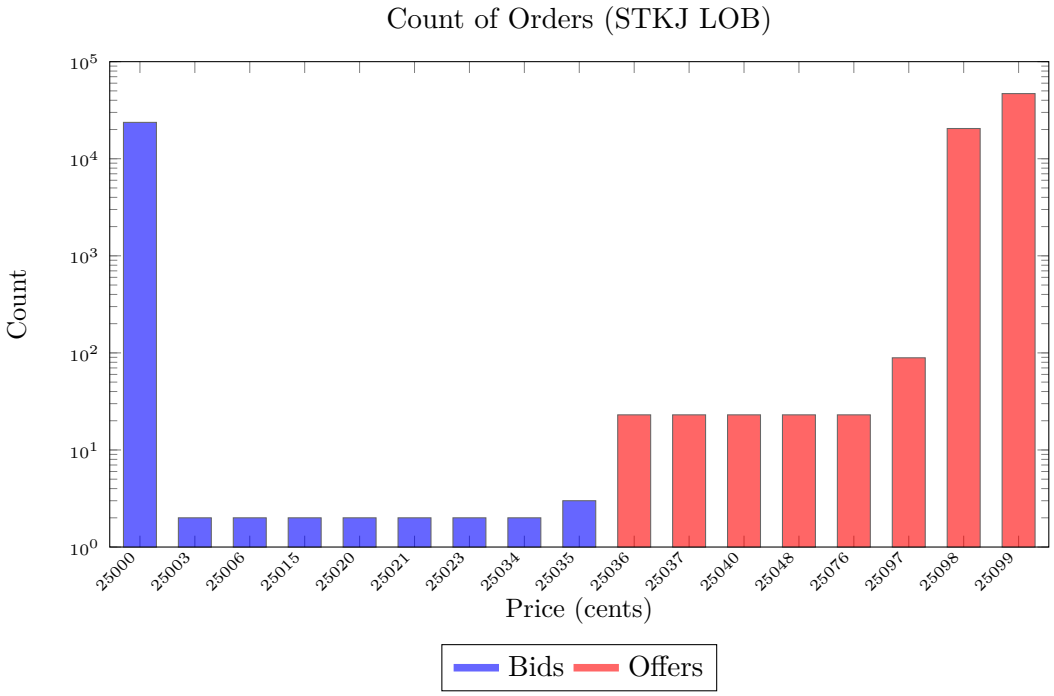


Figure 33: *Limit Order Book of stock STKJ. It shows the count of orders stored at each price point*

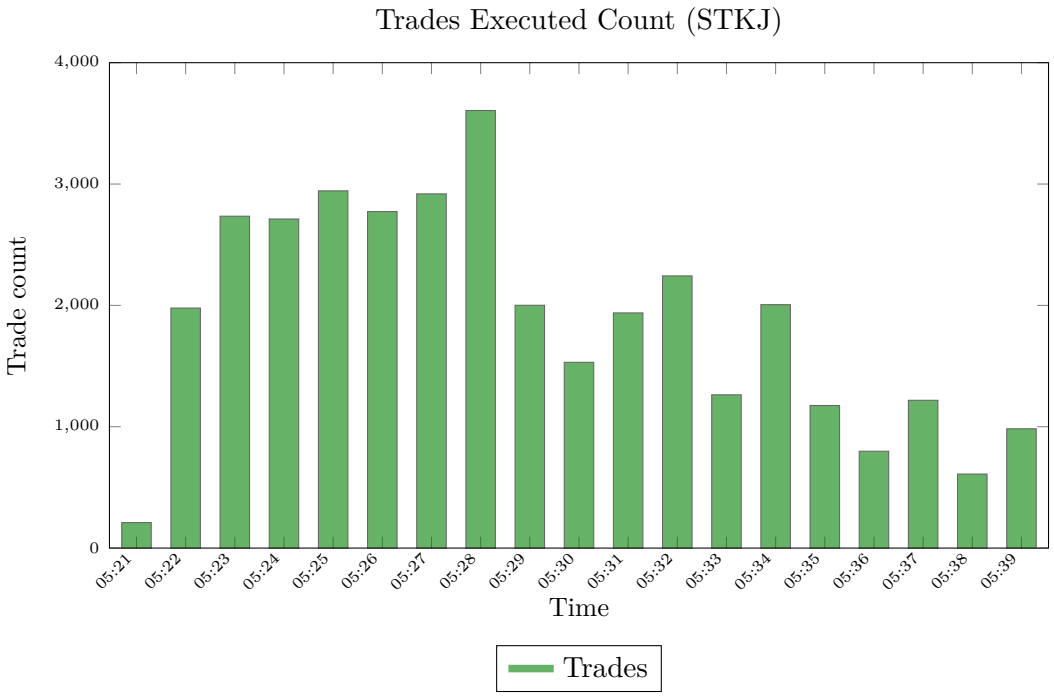


Figure 34: *Count of trades executed for stock STKJ. It shows the count of trades executed per minute during the test run*

8.4.4 Scenario B: Throughput Testing

Table 8 shows the throughput per second test results using an Intraday Auction. As in Scenario A the throughput is reduced by the delays in the Hawkes simulator. The throughput is further reduced by the one minute Intraday Auction.

	Client	**Stock**	**Start Time**	**End Time**	**Duration**	**Orders**	**TPS**
B1	10	10	08:20:25.424	08:45:06.178	00:24:40.754	1119864	756

Table 8: Scenario B: Intraday Auction Throughput

8.4.5 Scenario B: Latency Testing

Figure 35 shows the latency using the data from test B1. The latency is 406 ns at the 90% percentile. This increase was due to the one minute Intraday Auction. Figure 36 shows the number of trades executed in 1 minute intervals for a single stock. The missing bar in the graph represents the one minute auction. The trades are created at the end of the auction when the order book is crossed.

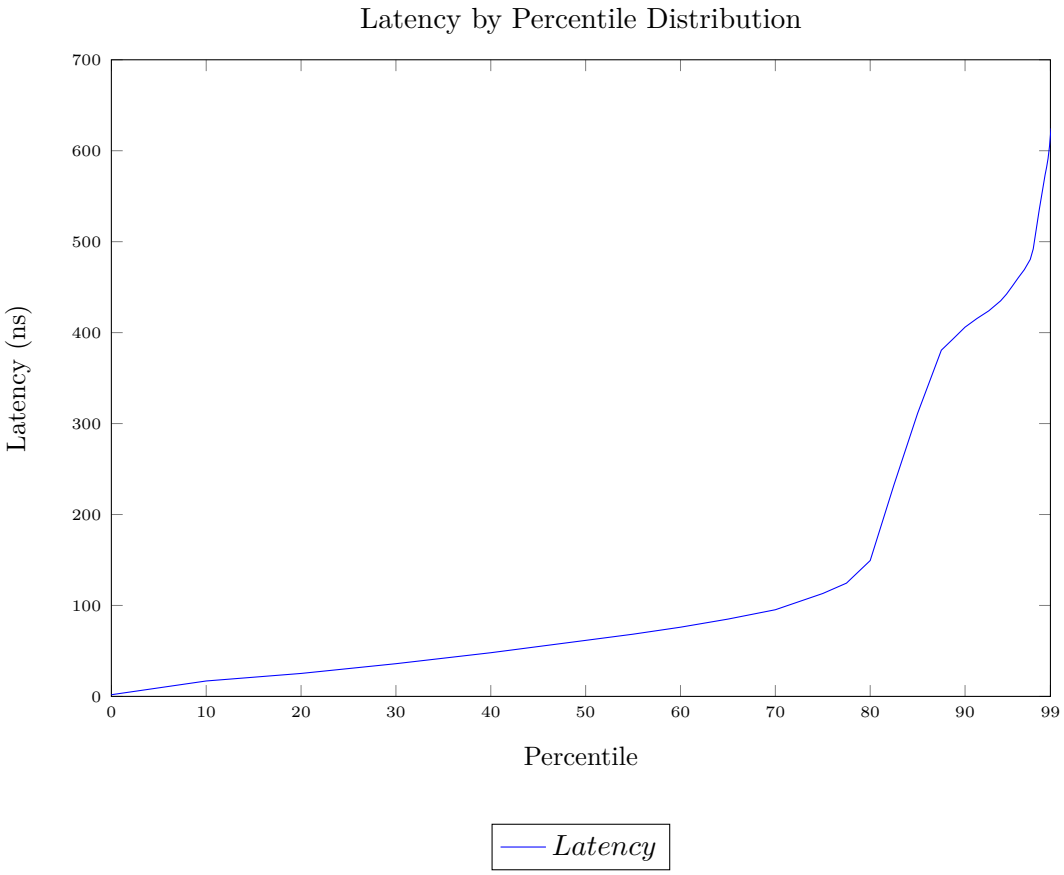


Figure 35: *A High Dynamic Range (HDR) Histogram graph. It shows the latency percentile of the matching engine with an Intraday Auction.*

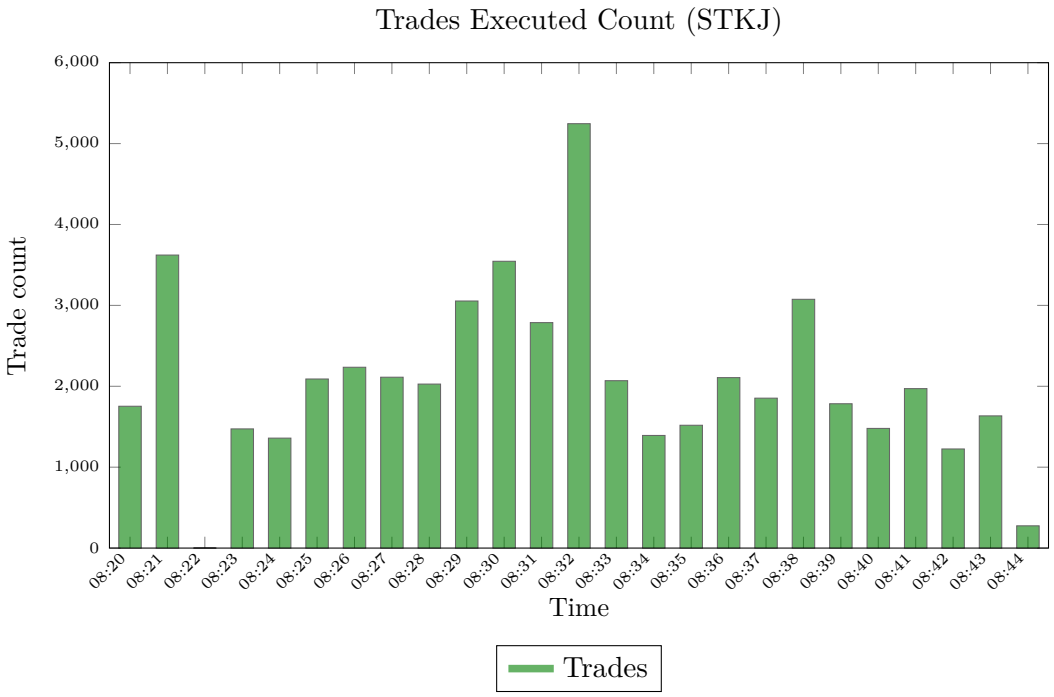


Figure 36: *Count of trades executed for stock STKJ. It shows the count of trades executed per minute during the test run*

8.5 Analysis

The unit tests cover the testing of the functional requirements of the software. These test provide a safety net to allow changes to be made to the code without breaking existing functionality. The software has unit tests that cover a majority of the application. The tests assisted when developing the logic of the matching algorithms. JMH testing was used to test the performance of individual methods.

The performance testing of the software showed that it is has a low latency. The JSE has a latency of 50 microseconds. The tests show that the latency is between 99-353 ns at the 90th percentile. This is a consistent low latency. The latency increased when the Volatility Auction executed. This is expected behavior as the matching engine performs more work at the end of the auction.

The throughput was affected by the delays in the Hawkes clients. The Hawkes testing provides a more realistic simulation but does not push the system. A streaming client test case would give the true value of the throughput of the software.

The software is able to store all active orders in the order book across multiple stocks. The order book depth was limited to 10 for bids to buy and offers to

sell. This limitation is required for the Hawkes process to be more accurate but restricts the orders stored in the order book. Removing the limitation would test the software's true limit order book capacity.

The software was deployed and tested on a single server. This may have had an impact on the performance. Further testing is required to test on multiple servers and using different deployment configurations *i.e.* splitting the stocks across multiple matching engines.

9 Conclusion

9.1 Discussion

CoinTossX is a low latency high throughput stock exchange. It is configurable and allows users to view the limit order book in real time. The software allows multiple clients to connect and send orders to the exchange. The exchange supports multiple stocks and a variety of different trading session logic *i.e.* Continuous Trading, Intraday Auction Trading and various combinations of these, including rolling session auctions. The Hawkes processes provides a realistic client simulator for the exchange.

Creating the software required designing and testing each class and component. The software was designed and developed in iterations. Unit and performance testing was done in each iteration and had a significant impact on the architecture. The architecture was designed so that all components worked together optimally with high software cohesion and relatively low coupling. If a single process was faster or slower, this would have an impact on the overall software. The main challenges were working on the storage of the data in the limit order book, maintaining the correct flow control of messages and ensuring the hardware and software were configured correctly. If the deployment or hardware changes, the performance needs to be tested again.

This dissertation describes the design, implementation and testing required to build a matching engine for high frequency trading. As of this writing, such a software does not exist for traders, organizations and academic institutions to test their agent based models and further their understanding of market microstructure. The software will be open sourced and available at `https://bitbucket.org/dharmeshsing/jsematchingengine`

The software and Hawkes client processes were deployed on one server which affected the performance of all components. The logic for the different trading sessions were implemented, but not all trading sessions were verified. The DAY Time In Force was only used for the Hawkes testing.

9.2 Further Work

The software was designed such that different matching logic algorithms are in separate Java classes. This allows the logic to be changed to test any variations of the matching logic. Further work can be done on deployments to different hardware architecture layouts. The CoinTossX website can be enhanced to analyze the data that is processed. The exchange will provide a platform for agent based models to be tested using software that is closer to reality than any simple academic matching engine. Additional work can be done to change the matching rules on the engine to

test its impact on the limit order book. For example, introducing new order types and simulating their impact. In addition the design of the LOB could be refactored to support new technologies such as the blockchain. Also, since the components are de-coupled, their implementation language can be changed to support the latest frameworks.

References

[1] Abergel, F., Chakraborti, A., Anane, M., Jedidi, A., and Toke, I. M. (2016). *Limit order books*. Cambridge University Press.

[2] Alur, D., Crupi, J., and Malks, D. (2003). *Core J2EE Patterns*. Prentice Hall PTR.

[3] Bacry, E., Mastromatteo, I., and Muzy, J.-F. (2015). Hawkes processes in finance. arXiv preprint arXiv:1502.04592.

[4] Baker, H. K. and Kiymaz, H. (2013). Trends in Market Microstructure. Retrieved from `http://www.europeanfinancialreview.com/?p=827`.

[5] Beck, K. (2003). *Test Driven Development. By Example*. Addison Wesley.

[6] Bhatt, P. C. P. (2014). *An Introduction to Operating Systems: Concepts and Practice (GNU/LINUX)*. Prentice Hall of India.

[7] Bobrovs, V., Spolitis, S., and Ivanovs, G. (2013). Latency causes and reduction in optical metro networks. In *SPIE OPTO*, pages 90080C–90080C. International Society for Optics and Photonics.

[8] Borkum, H. (2012). JSE prepares for transition to new technology. Retrieved from `https://www.jse.co.za/news/jse-prepares-for-transition-to-new-technology`.

[9] Bouchaud, J.-P., Mézard, M., Potters, M., et al. (2002). Statistical properties of stock order books: empirical results and models. *Quantitative Finance*, 2(4):251–256.

[10] Bowsher, C. G. (2007). Modelling Security Market Events in Continuous Time: Intensity Based, Multivariate Point Process Models. *Journal of Econometrics*, 141(2):876–912.

[11] Brodal, G. S. (2004). Cache-Oblivious Algorithms and Data Structures. In *Algorithm Theory - SWAT 2004*. Springer.

[12] Carmona, R. (2013). Limit Order Books. Retrieved from `https://www.princeton.edu/~rcarmona/download/short_courses/Princeton_June2013/RTG1.pdf`.

[13] Cartea, Á., Jaimungal, S., and Penalva, J. (2015). *Algorithmic and high-frequency trading*. Cambridge University Press.

[14] Castro, L. N. D. (2006). *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*. Chapman & Hall/CRC.

[15] CFTC (2012). Sub-Committee on Automated and High Frequency Trading Working Group 1. Retrieved from `http://www.cftc.gov/ucm/groups/public/@newsroom/documents/file/tac103012_wg1.pdf`.

[16] Comer, D. (1979). Ubiquitous B-Tree. *Computing Surveys*, 11(2):121–137.

[17] Daley, D. and Vere-Jones, D. (2003). *An Introduction to the Theory of Point Processes*. Springer-Verlag, 2nd edition.

[18] Drepper, U. (2007). What Every Programmer Should Know About Memory. Retrieved from `https://people.freebsd.org/~lstewart/articles/cpumemory.pdf`.

[19] Dworak, A., Charrue, P., Ehm, F., Sliwinski, W., and Sobczak, M. (2011). Middleware trends and market leaders 2011. volume 111010, No. CERN-ATS-2011-196, p. FRBHMULT05, pages 1334–1338.

[20] Engelbert, C. (2015). A Post-Apocalyptic sun.misc.Unsafe World. Retrieved from `https://www.infoq.com/articles/A-Post-Apocalyptic-sun.misc.Unsafe-World`.

[21] Farley, D. (2015). Mechanical Sympathy: Understanding the Hardware Makes You a Better Developer. Retrieved from `https://dzone.com/articles/mechanical-sympathy`.

[22] Farley, D. and Thompson, M. (2012). Lmax Disruptor: 100K TPS at Less than 1ms Latency. Retrieved from `http://www.infoq.com/presentations/LMAX-Disruptor-100K-TPS-at-Less-than-1ms-Latency`.

[23] Farmer, J. D., Patelli, P., and Zovko, I. I. (2005). The predictive power of zero intelligence in financial markets. *Proceedings of the National Academy of Sciences*, 102(6):2254–2259.

[24] FIX Trading Community (2014). What is FIX? Retrieved from `http://www.fixtradingcommunity.org/pg/main/what-is-fix`.

[25] FIX Trading Community (2015). FIX Trading Community announces new open technical resource to facilitate software development - Press Release. Retrieved from `http://www.fixtradingcommunity.org/pg/blog/fplpo/read/2891183/`.

[26] Gomber, P., Arndt, B., Lutat, M., and Uhle, T. E. (2011). High-Frequency Trading. *SSRN Electronic Journal*. Available from: `http://ssrn.com/abstract=1858626`.

[27] Gorham, M. and Singh, N. (2009). *Electronic Exchanges: The Global Transformation from Pits to Bits*. Elsevier and IIT Stuart Center for Financial Markets Press. Elsevier Science.

[28] Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., and Howison, S. D. (2013). Limit Order Books. *Quantitative Finance*, 13(11):1709–1742.

[29] Grant, R. E., Balaji, P., and Afsahi, A. (2010). A study of hardware assisted IP over InfiniBand and its impact on enterprise data center performance. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 144–153. IEEE.

[30] Hasbrouck, J. (2007). *Empirical Market Microstructure: The Institutions, Economics, and Econometrics of Securities Trading*. Oxford University Press.

[31] Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90.

[32] Hintjens, P. (2014). OMQ - The Guide. Retrieved from `http://zguide.zeromq.org/page:all`.

[33] Houstoun, K. (2006). FIX Adapted for STreaming - FAST Protocol Technical Overview. Retrieved from `http://www.fixtradingcommunity.org/mod/file/view.php?file_guid=42876`.

[34] Hughes, L. (2012). JSE High Frequency Trading A discussion document to entice market engagement and consultation. Technical report, Johannesburg Stock Exchange Pty (Ltd).

[35] ISO/IEC (2014). ISO International Standard ISO/IEC 14882:2014(E) - Programming Language C++. Retrieved from `https://isocpp.org/std/the-standard`.

[36] Jenkov, J. (2015). JMH - Java Microbenchmark Harness. Retrieved from `http://tutorials.jenkov.com/java-performance/jmh.html`.

[37] JeroMQ (2016). Pure Java ZeroMQ. Retrieved from `https://github.com/zeromq/jeromq`.

[38] JSE (2012). JSE's new Equity Trading Platform in SA after Decade in London. Retrieved from `http://ir.jse.co.za/phoenix.zhtml?c=198120&p=irol-newsArticle_print&ID=1713058`.

[39] JSE (2013). New Equity Market Trading and Information Solution JSE Specification Document Volume 00 Trading and Information Overview. 2.03. Retrieved from `https://www.jse.co.za/services/technologies/equity-market-trading-and-information-technology-change`.

[40] JSE (2014a). The lowest-latency connection to JSE markets. Technical report. Available from: `https://www.jse.co.za/content/JSETechnologyDocumentItems/3.%20JSE%20Colocation%20Brochure%202015.pdf`.

[41] JSE (2014b). New Equity Market Trading and Information Solution JSE Specification Document Native Trading Gateway. Retrieved from `http://ir.jse.co.za/phoenix.zhtml?c=198120&p=irol-newsArticle_print&ID=1713058`.

[42] JSE (2014c). New Equity Market Trading and Information Solution JSE Specification Document Volume 01 Native Trading Gateway. 2.02. Retrieved from https://www.jse.co.za/services/technologies/equity-market-trading-and-information-technology-change.

[43] JSE (2014d). New Equity Market Trading and Information Solution JSE Specification Document Volume 05 Market Data Gateway (ITCH UDP). 2.04. Retrieved from https://www.jse.co.za/services/technologies/equity-market-trading-and-information-technology-change.

[44] Khan, M. A. (2009). Optimization Study for Multicores. Master's thesis, Uppsala University, Department of Information Technology.

[45] Kotek, J. (2017). MapDB. Retrieved from https://www.gitbook.com/book/jankotek/mapdb/details.

[46] Kowarschik, M. and Weiß, C. (2003). An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In *Algorithms for Memory Hierarchies*, chapter 10, pages 213–232. Springer Nature.

[47] Kukanov, A. (2013). *Stochastic Models of Limit Order Markets*. PhD thesis, Columbia University.

[48] Lafore, R. (2002). *Data Structures and Algorithms in Java*. Pearson Education (US).

[49] Large, J. (2007). Measuring the resiliency of an electronic limit order book. *Journal of Financial Markets*, 10(1):1–25.

[50] Lawrey, P. (2013). Writing and Testing High Frequency Trading System. Retrieved from http://java.dzone.com/articles/writing-and-testing-high.

[51] Lawrey, P. (2015). On Heap vs Off Heap Memory Usage. Retrieved from https://dzone.com/articles/heap-vs-heap-memory-usage.

[52] Lewis, P. A. W. and Shedler, G. S. (1979). Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413.

[53] LSE (2000). London Stock Exchange market enhancements release 3.1 worked examples. Retrieved from https://www.londonstockexchange.com/products-and-services/technical-library/technical-guidance-notes/technicalguidancenotesarchive/release.pdf.

[54] Madhavan, A. (2000). Market microstructure: A survey. *Journal of Financial Markets*, 3:205–258.

[55] Mangione, C. (1998). Performance tests show Java as fast as C++. Retrieved from http://www.javaworld.com/article/2076593/performance-tests-show-java-as-fast-as-c--.html.

[56] Mehta, D. P. and Sahni, S. (2004). *Handbook of Data Structures and Applications*. Taylor & Francis Inc.

[57] Morgan, T. P. (2014). Wall Street Wants Tech To Trade Smarter And Faster. Retrieved from `https://www.enterprisetech.com/2014/04/09/wall-street-wants-tech-trade-smarter-faster/`.

[58] Naes, R. and Skjeltorp, J. (2006). Is the market microstructure of stock markets important? *Norges Bank Economic Bulletin*, 77:123.

[59] Nair, P. (2014). *Agent based modelling of a single-stock market on the JSE*. Msc thesis, University of the Witwatersrand.

[60] NasdaqTrader (2014). NASDAQ TotalView-ITCH 4.0. Retrieved from `http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/tvitch-v4.pdf`.

[61] NasdaqTrader (2016). Ouch 4.2. Retrieved from `http://www.nasdaqtrader.com/content/technicalsupport/specifications/TradingProducts/OUCH4.2.pdf`.

[62] Navarro, C. A., Hitschfeld-Kahler, N., and Mateu, L. (2014). A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures. *Communications in Computational Physics*, 15(02):285–329.

[63] Nchaba, L. P. (2015). Simulation of Realistic Market using the Hawkes Process. University of the Witwatersrand.

[64] Ogata, Y. (1981). On Lewis' Simulation Method for Point Processes. *IEEE Transactions On Information Theory*, 27:23–31.

[65] Ogata, Y. (1988). Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association*, 83:9–27.

[66] O'Hara, M. (1995). *Market Microstructure Theory*. Blackwell Publishers.

[67] O'Hara, M. (2012). FPGA & Hardware Accelerated Trading, Part One - Who, What, Where and Why? Retrieved from `http://www.thetradingmesh.com/pg/blog/mike/read/55950/fpga-hardware-accelerated-trading-part-one-who-what-where-and-why`.

[68] Oracle (2016). Code Tools: jmh. Retrieved from `http://openjdk.java.net/projects/code-tools/jmh/`.

[69] Oracle (2017). Retrieved from `www.oracle.com/technetwork/java/index.html`.

[70] Parziale, L., Britt, D. T., Davis, C., Forrester, J., Liu, W., Matthews, C., and Rosselot, N. (2006). *TCP/IP Tutorial and Technical Overview*. Available from: `http://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf`.

[71] Paul, J. (2012). Why use Memory Mapped File or MapppedByteBuffer in Java. Retrieved from http://javarevisited.blogspot.com.au/2012/01/memorymapped-file-and-io-in-java.html.

[72] Rao, J. and Ross, K. A. (2000). Making B+- trees cache conscious in main memory. *ACM SIGMOD Record*, 29(2):475–486.

[73] SEC (2004). Regulation NMS. (File No. S7-10-04). Available from: http://www.sec.gov/rules/proposed/34-50870.pdf.

[74] Shasha, D. and Bonnet, P. (2002). *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Morgan Kaufmann Publishers.

[75] Silberschatz, A., Korth, H. F., and Sudarshan, S. (2010). *Database System Concepts*. McGraw-Hill Higher Education.

[76] Sing, D. (2017). Operational Concept Description for JSE Matching Engine Simulator. *Working Technical Document*. University of the Witwatersrand.

[77] Sing, D., Hendricks, D., and Gebbie, T. (2017). The simulation of a realistic market data feed using mutually-exciting hawkes processes. *Working Paper*. University of the Witwatersrand.

[78] Spring (2016). Spring Boot. Retrieved from https://projects.spring.io/spring-boot/.

[79] Tene, G. (2016). HdrHistogram: A High Dynamic Range Histogram. Retrieved from https://hdrhistogram.github.io/HdrHistogram/.

[80] Tene, G. and Thompson, M. (2017). org.ObjectLayout: A layout-optimized Java data structure package. Retrieved from http://objectlayout.github.io/ObjectLayout/.

[81] Thompson, M. (2014a). Design Overview. Retrieved from https://github.com/real-logic/Aeron/wiki/Design-Overview.

[82] Thompson, M. (2014b). Simple Binary Encoding. Retrieved from http://mechanical-sympathy.blogspot.com.au/2014/05/simple-binary-encoding.html.

[83] Thompson, M. (2015). Design principles. Retrieved from: https://github.com/real-logic/Aeron/wiki/Design-Principles.

[84] Thompson, M. (2016). Aeron: The Next Generation in High-performance Messaging. Retrieved from https://www.infoq.com/presentations/aeron.

[85] Thompson, M., Farley, D., Barker, M., Gee, P., and Stewart, A. (2011). Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads. Retrieved from http://lmax-exchange.github.io/disruptor/files/Disruptor-1.0.pdf.

[86] Toke, I. M. (2011a). An Introduction to Hawkes Processes with Applications to Finance. *Lectures Notes from Ecole Centrale Paris, BNP Paribas Chair of Quantitative Finance*. Retrieved from `http://lamp.ecp.fr/MAS/fiQuant/ioane_files/HawkesCourseSlides.pdf`.

[87] Toke, I. M. (2011b). Some Applications of Hawkes Processes for Order Book Modelling. *First Unconventional Workshop on Quantitative Finance and Economics*. Available from: `http://lamp.ecp.fr/MAS/fiQuant/ioane_files/20110223-QFWTokyoSlides.pdf`.

[88] Toke, I. M. and Pomponio, F. (2012). Modelling Trades-Through in a Limit Order Book Using Hawkes Processes. *Economics discussion paper*, 6.

[89] Union, E. (2004). Directive 2004/39/EC of the European Parliament and of the Council of 21 April 2004 on markets in financial instruments amending Council Directives 85/611/EEC and 93/6/EEC and Directive 2000/12/EC of the European Parliament and of the Council and repealing Council Directive 93/22/EEC. *Official Journal of the European Union*, 47. Available from: `http://eur-lex.europa.eu/eli/dir/2004/39/2011-01-04`.

[90] Wicket (2016). Wicket 8.x Reference Guide. Retrieved from `https://ci.apache.org/projects/wicket/guide/8.x/single.html`.

[91] Yasenchak, R. and Arendell, S. (2014). The Trading Series Part 1:The Evolution of Trading - From Quarters to Pennies and Beyond. Available from: `https://www.intechjanus.com/intech/insight-and-research`.

[92] Zaatour, R. (2015). *Hawkes process simulation and calibration toolkit*. Retrieved from `https://cran.r-project.org/web/packages/hawkes/hawkes.pdf`.

[93] Zhu, H. (2013). Do Dark Pools Harm Price Discovery? *SSRN Electronic Journal*.

A Appendix

A.1 Java vs Google Protocol Buffers Test

This test simulates an order message and captures the size, serialization and deserialization times using Java and the Google Protocol Buffer library. A warmup phase is run first before the actual test to ensure that the JVM has optimized the code. The warmup phase runs 10 times, serializing and deserializing 10 000 objects. The actual test runs 10 times, serializing and deserializing 1 million objects. The tables below show the results of the tests.

Run	Serialize (ms)	Deserialize (ms)	Size (bytes)
1	4789	28972	460
2	4903	28178	460
3	4924	28001	460
4	4926	27580	460
5	4879	26760	460
6	4710	27034	460
7	4773	26154	460
8	4820	29223	460
9	5136	27333	460
10	4657	26545	460
Average	**4846**	**27578**	**460**

Table 9: Java Test

Run	Serialize (ms)	Deserialize (ms)	Size (bytes)
1	126	323	45
2	122	349	45
3	118	290	45
4	119	304	45
5	121	293	45
6	115	307	45
7	117	296	45
8	128	339	45
9	128	339	45
10	124	252	45
Average	**121.8**	**309.2**	**45**

Table 10: Google Protocol Buffer Test

A.2 Aeron Performance Tests

A.2.1 Aeron ThroughPut Performance Test

```
java
-cp
aeron-samples/build/libs/samples.jar
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
-XX:BiasedLockingStartupDelay=0
-Daeron.mtu.length=16384
-Daeron.socket.so_sndbuf=2097152
-Daeron.socket.so_rcvbuf=2097152
-Daeron.rcv.buffer.length=16384
-Daeron.rcv.initial.window.length=2097152
-Dagrona.disable.bounds.checks=true
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
uk.co.real_logic.aeron.samples.LowLatencyMediaDriver

java
-cp
aeron-samples/build/libs/samples.jar
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
-Daeron.sample.messageLength=120
-Daeron.sample.messages=500000000
-Dagrona.disable.bounds.checks=true
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
uk.co.real_logic.aeron.samples.StreamingPublisher


java
-cp
aeron-samples/build/libs/samples.jar
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
-Dagrona.disable.bounds.checks=true
-Daeron.sample.frameCountLimit=256
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
uk.co.real_logic.aeron.samples.RateSubscriber
```

Listing 3: Aeron Performance Test

- Subscriber
  - 1.3e+06 msgs/sec, 1.6e+08 bytes/sec, totals 500 000 000 messages 57 220 MB
- Publisher
  - 7.9e+05 msgs/sec, 9.4e+07 bytes/sec, totals 500 000 000 messages 57 220 MB
- 20% CPU Usage
- 1 GC Event

A.2.2 Aeron Latency Performance Test

```
java
-cp
aeron-samples/build/libs/samples.jar
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
-Daeron.sample.messages=100000
-Daeron.sample.messageLength=120
-Dagrona.disable.bounds.checks=true
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
 uk.co.real_logic.aeron.samples.Ping


java
-cp
aeron-samples/build/libs/samples.jar
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
-Dagrona.disable.bounds.checks=true
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
uk.co.real_logic.aeron.samples.Pong
```

Listing 4: Aeron Latency Test

B Appendix

B.1 Matching Engine Test Cases

This section lists the matching engine unit test cases that was used to test the functionality of the matching logic. The test cases are separated into the following sections:

1. Market Order Test Case
2. Limit Order Test Case
3. Hidden Order Test Case
4. Stop Order Test Case
5. Stop Limit Order Test Case
6. Filter And Uncross Test Case
7. Auction Test Case
8. Cancel Order Test Case
9. Replace Order Test Case

Each test case shows the changes to the limit order book as an aggressive order is processed *i.e.* the initial and final state of the lob. The aggressive order and trades executed are also shown. The Auction test case does not have aggressive orders.

Abbreviations Used:

- MO - Market Order
- LO - Limit Order
- HO - Hidden Order
- SO - Stop Order
- SL - Stop Limit Order
- MES - Minimum Execution Size
- MRS - Minimum Reserve Size

The default MRS is 15000.

B.1.1 Market Order Test Case - Test 1

Add bid market message to an empty lob. Order expires

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	MO	0	10:00	1000	0.0					

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.2 Market Order Test Case - Test 2

Add offer market message to an empty lob. Order expires

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	10:00	0	MO	1

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.3 Market Order Test Case - Test 3

Add bid market message. Order is filled

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					50.0	200	11:10	0	LO	3
					80.0	500	11:00	0	LO	2
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	MO	0	12:00	1000	0.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	700	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	50.0	200
2	80.0	500
3	100.0	300

Trades

B.1.4 Market Order Test Case - Test 4

Add offer market message. Order is filled

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	11:10	200	50.0					
2	LO	0	11:00	500	80.0					
1	LO	0	10:00	1000	100.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	12:00	0	MO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	11:10	200	50.0					
2	LO	0	11:00	500	80.0					

Final State

Trade Id	**Price**	**Quanity**
1	100.0	1000

Trades

B.1.5 Market Order Test Case - Test 5

Add bid market message. Order is partially filled. Order is expired

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	500	11:00	0	LO	2
					100.0	500	10:00	0	LO	1
					110.0	2000	11:10	1000	HO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	MO	0	12:00	1200	0.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					110.0	2000	11:10	1000	HO	3

Final State

Trade Id	**Price**	**Quanity**
1	80.0	500
2	100.0	500

Trades

B.1.6 Market Order Test Case - Test 6

Add offer market message. Order is partially filled. Order is expired

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	HO	1000	11:00	2000	70.0					
2	LO	0	10:10	500	80.0					
1	LO	0	10:00	500	100.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1200	12:00	0	MO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	HO	1000	11:00	2000	70.0					

Final State

Trade Id	**Price**	**Quanity**
1	100.0	500
2	80.0	500

Trades

B.1.7 Limit Order Test Case - Test 1

Add bid limit message to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.8 Limit Order Test Case - Test 2

Add bid limit message with different bid price

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	500	200.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	500	200.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.9 Limit Order Test Case - Test 3

Add bid limit message with existing bid price

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
1	LO	0	10:00	1000	100.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.10 Limit Order Test Case - Test 4

Add bid limit message with existing offer price. No Match

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	50.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	50.0					
					100.0	1000	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.11 Limit Order Test Case - Test 5

Add bid limit message with existing offer price. Match

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					50.0	200	11:10	0	LO	3
					80.0	500	11:00	0	LO	2
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	1000	80.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	300	80.0					
					100.0	1000	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	50.0	200
2	80.0	500

Trades

B.1.12 Limit Order Test Case - Test 6

Bid LO matches Offer LO. Time-priority determines the matching orders

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					20.0	200	12:00	0	LO	4
					50.0	300	11:00	0	LO	2
					50.0	300	11:10	0	LO	3
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
5	LO	0	12:00	500	50.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					50.0	300	11:10	0	LO	3
					100.0	1000	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	20.0	200
2	50.0	300

Trades

B.1.13 Limit Order Test Case - Test 7

Add offer limit message to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.14 Limit Order Test Case - Test 8

Add offer limit message with different offer price

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					200.0	500	11:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					200.0	500	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.15 Limit Order Test Case - Test 9

Add offer limit message with existing offer price

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	11:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					100.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.16 Limit Order Test Case - Test 10

Add offer limit message with existing bid price. No Match

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	50.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	11:00	0	LO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	50.0					
					100.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.17 Limit Order Test Case - Test 11

Add offer limit message with existing bid price. Match

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	11:10	200	50.0					
2	LO	0	11:00	200	80.0					
1	LO	0	10:00	500	100.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	1000	12:00	0	LO	4

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	11:10	200	50.0					
					80.0	300	12:00	0	LO	4

Final State

Trade Id	**Price**	**Quanity**
1	100.0	500
2	80.0	200

Trades

B.1.18 Limit Order Test Case - Test 12

Offer LO matches Bid LO. Time-priority determines the matching orders

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
2	LO	0	11:00	300	50.0					
3	LO	0	11:10	300	50.0					
1	LO	0	10:00	700	100.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					50.0	1000	12:00	0	LO	5

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	300	50.0					

Final State

Trade Id	**Price**	**Quanity**
1	100.0	700
2	50.0	300

Trades

B.1.19 Hidden Order Test Case - Test 1

Incoming offer LO matches on contra side including HO. Visible LO takes precedence over HO at same price point.MRS = 700

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	HO	500	11:10	500	50.0					
6	LO	0	11:05	800	50.0					
5	LO	0	13:00	200	60.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					40.0	1000	12:00	0	LO	7

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	HO	500	11:10	500	50.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	60.0	200
2	50.0	800

Trades

B.1.20 Hidden Order Test Case - Test 2

Incoming bid LO matches on contra side including HO. Visible LO takes precedence over HO at same price point.MRS = 700

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	300	20.0					
1	LO	0	10:00	700	40.0					
					50.0	200	12:00	0	LO	4
					80.0	500	11:10	500	HO	3
					80.0	800	11:05	0	LO	6
					100.0	200	13:00	0	LO	5

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
7	LO	0	14:00	1000	90.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	300	20.0					
1	LO	0	10:00	700	40.0					
					80.0	500	11:10	500	HO	3
					100.0	200	13:00	0	LO	5

Final State

Trade Id	**Price**	**Quanity**
1	50.0	200
2	80.0	800

Trades

B.1.21 Hidden Order Test Case - Test 3

Incoming offer LO matches on contra side including HO. MES is used to execute HO.MRS = 700

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	500	45.0					
6	HO	700	11:05	1400	50.0					
5	LO	0	13:00	200	60.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					40.0	1000	12:00	0	LO	7

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	500	45.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	60.0	200
2	50.0	800

Trades

B.1.22 Hidden Order Test Case - Test 4

Incoming offer LO matches on contra side including HO. HO is skipped due to MES constraint.MRS = 700

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	500	45.0					
6	HO	700	11:05	1400	50.0					
5	LO	0	13:00	500	60.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					40.0	600	12:00	0	LO	7

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	400	45.0					
6	HO	700	11:05	1400	50.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	60.0	500
2	45.0	100

Trades

B.1.23 Hidden Order Test Case - Test 5

Incoming offer HO matches on contra side including HO. Passive HO is skipped due to MES constraint.MRS = 700

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	500	45.0					
6	HO	700	11:05	1400	50.0					
5	LO	0	13:00	500	60.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					40.0	600	12:00	100	HO	7

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	12:00	200	20.0					
3	LO	0	11:10	400	45.0					
6	HO	700	11:05	1400	50.0					
					80.0	300	11:00	0	LO	2
					100.0	700	10:00	0	LO	1

Final State

Trade Id	**Price**	**Quanity**
1	60.0	500
2	45.0	100

Trades

B.1.24 Hidden Order Test Case - Test 6

Executing a Sell Limit message with price improvement

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	12:00	0	LO	3
1	HO	15000	10:00	15000	106.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					104.0	15000	13:00	0	LO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	104.5	15000

Trades

B.1.25 Hidden Order Test Case - Test 7

Executing a Sell Order at aggressing message's limit price

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	12:00	0	LO	3
1	HO	15000	10:00	15000	106.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	15000	13:00	0	LO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	13:00	0	LO	4

Final State

Trade Id	**Price**	**Quanity**
1	105.0	1000
2	105.0	14000

Trades

B.1.26 Hidden Order Test Case - Test 8

Not Executing a Sell Order due to aggressing message's limit price breach

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	12:00	0	LO	3
1	HO	15000	10:00	15000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					106.0	15000	13:00	0	LO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					105.0	1000	12:00	0	LO	3
					106.0	15000	13:00	0	LO	4
1	HO	15000	10:00	15000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.27 Hidden Order Test Case - Test 9

Executing a Sell Order based on Price-Visibility-Time Execution Priority

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
1	HO	15000	10:00	15000	106.0					
					107.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					104.0	15000	13:00	0	LO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	106.0	15000

Trades

B.1.28 Hidden Order Test Case - Test 10

Executing a Buy Limit message with price improvement

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	15000	12:00	15000	HO	3
1	LO	0	10:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	15000	107.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	106.5	15000

Trades

B.1.29 Hidden Order Test Case - Test 11

Executing a Buy Order at the aggressing message's limit price

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	15000	12:00	15000	HO	3
1	LO	0	10:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	15000	106.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	106.0	1000
2	106.0	14000

Trades

B.1.30 Hidden Order Test Case - Test 12

Not Executing a Buy Order due to aggressing message's limit price breach

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	15000	12:00	15000	HO	3
1	LO	0	10:00	1000	107.0					
					108.0	1000	11:00	0	LO	2

Initial State

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	15000	106.0					

Aggressive Order

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	15000	12:00	15000	HO	3
4	LO	0	13:00	15000	106.0					
1	LO	0	10:00	1000	107.0					
					108.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.31 Hidden Order Test Case - Test 13

Executing a Buy Order based on Price-Visibility-Time Execution Priority

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	103.0					
					105.0	15000	12:00	15000	HO	3
					107.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	15000	106.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	103.0					
					107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	105.0	15000

Trades

B.1.32 Hidden Order Test Case - Test 14

Executing a Buy Order stepping over a Hidden Limit message due to a MES constraint

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1000	100.0					
1	HO	15000	10:00	15000	106.0					
					107.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	13:00	0	LO	4

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	HO	15000	10:00	15000	106.0					
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	100.0	1000

Trades

B.1.33 Hidden Order Test Case - Test 15

Executing a Buy Market message with price improvement

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	100000	12:00	100000	HO	3
1	LO	0	10:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	MO	0	13:00	100000	0.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	106.0					
					107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	106.5	100000

Trades

B.1.34 Hidden Order Test Case - Test 16

Executing a Sell Market message with price improvement

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	10:00	100000	100.0					
					101.0	80000	11:00	0	LO	2
3	HO	100000	12:00	100000	102.0					

Initial State

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
					0.0	100000	13:00	0	MO	4

Aggressive Order

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	10:00	100000	100.0					
					101.0	80000	11:00	0	LO	2

Final State

Trade Id	Price	Quanity
1	100.5	100000

Trades

B.1.35 Hidden Order Test Case - Test 17

Executing a Sell Market message which creates sufficient quantity for the message at the visible best offer to execute against a Hidden Limit message with a MES constraint

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	100000	100.0					
					101.0	80000	11:00	0	LO	2
3	HO	100000	12:00	100000	102.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	200000	13:00	0	MO	4

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					101.0	80000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	100.5	100000
2	100.0	100000

Trades

B.1.36 Stop Order Test Case - Test 1

Add buy stop order to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SO	0	12:00	1000	90.0					

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SO	0	12:00	1000	90.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.37 Stop Order Test Case - Test 2

Add sell stop order to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	1000	12:00	0	SO	1

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	1000	12:00	0	SO	1

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.38 Stop Order Test Case - Test 3

Buy stop order added. Last trade price does not exist. No executions

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
10	SO	0	13:00	1000	90.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
10	SO	0	13:00	1000	90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.39 Stop Order Test Case - Test 4

Sell stop order added. Last trade price does not exist. No executions

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	1000	13:00	0	SO	10

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0	1000	13:00	0	SO	10
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.40 Stop Order Test Case - Test 5

Buy order agresses order book with buy stop order. Stop order executed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
10	SO	0	13:00	1000	90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
11	LO	0	14:00	2000	93.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
14	LO	0	12:00	1000	94.0					

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000
2	92.0	1000
3	92.0	1000

Trades

B.1.41 Stop Order Test Case - Test 6

Sell order agresses order book with sell stop order. Stop order executed

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0	1000	13:00	0	SO	10

Initial State

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					91.0	2000	14:00	0	LO	11

Aggressive Order

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					

Final State

Trade Id	**Price**	**Quanity**
1	94.0	1000
2	92.0	1000
3	92.0	1000

Trades

B.1.42 Stop Order Test Case - Test 7

Buy stop order added. Last trade price exists. Stop order executed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
15	LO	0	13:00	1000	90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
16	SO	0	14:00	2000	80.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					94.0	1000	12:00	0	LO	14

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000
2	92.0	2000

Trades

B.1.43 Stop Order Test Case - Test 8

Sell stop order added. Last trade price exists. Stop order executed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0	1000	13:00	0	LO	15

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					95.0	2000	14:00	0	SO	16

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					

Final State

Trade Id	**Price**	**Quanity**
1	94.0	1000
2	92.0	2000

Trades

B.1.44 Stop Order Test Case - Test 9

Buy stop orders with greatest difference between its stop price and the last traded price will be elected first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
16	SO	0	14:10	500	70.0					
15	SO	0	14:00	1000	80.0					
					90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
17	LO	0	15:00	1000	90.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					92.0	500	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000
2	92.0	500
3	92.0	1000

Trades

B.1.45 Stop Order Test Case - Test 10

Sell stop orders with greatest difference between its stop price and the last traded price will be elected first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0					
					95.0	1000	14:00	0	SO	15
					100.0	500	14:10	0	SO	16

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					94.0	1000	15:00	0	LO	15

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	500	92.0					

Final State

Trade Id	**Price**	**Quanity**
1	94.0	1000
2	92.0	500
3	92.0	1000

Trades

B.1.46 Stop Order Test Case - Test 11

Multiple buy stop orders with the same difference between its stop price and the last traded price. Oldest executed first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
15	SO	0	14:00	1000	80.0					
16	SO	0	14:10	500	80.0					
					90.0	1000	10:00	0	LO	12
					92.0	2000	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
17	LO	0	15:00	1000	90.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					92.0	500	11:00	0	LO	13
					94.0	1000	12:00	0	LO	14

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000
2	92.0	1000
3	92.0	500

Trades

B.1.47 Stop Order Test Case - Test 12

Multiple sell stop orders with the same difference between its stop price and the last traded price. Oldest executed first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	2000	92.0					
14	LO	0	12:00	1000	94.0					
					95.0	1000	14:00	0	SO	15
					95.0	500	14:10	0	SO	16

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					94.0	1000	15:00	0	LO	15

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
12	LO	0	10:00	1000	90.0					
13	LO	0	11:00	500	92.0					

Final State

Trade Id	**Price**	**Quanity**
1	94.0	1000
2	92.0	1000
3	92.0	500

Trades

B.1.48 Stop Limit Order Test Case - Test 1

Add buy stop limit message to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	12:00	12000	106.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	12:00	12000	106.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.49 Stop Limit Order Test Case - Test 2

Add sell stop limit message to an empty lob

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					106.0	12000	12:00	0	SL	1

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					106.0	12000	12:00	0	SL	1

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.50 Stop Limit Order Test Case - Test 3

Buy stop limit message added. Last trade price does not exist. No executions

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	104.0					
					107.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	SL	0	12:00	12000	107.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	104.0					
3	SL	0	12:00	12000	107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.51 Stop Limit Order Test Case - Test 4

Sell stop limit message added. Last trade price does not exist. No executions

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	104.0					
					107.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					104.0	12000	12:00	0	SL	3

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	104.0	12000	12:00	0	SL	3
					107.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.52 Stop Limit Order Test Case - Test 5

Buy message agresses message book with buy stop limit message. Stop message executed

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	10:00	1000	105.0					
3	LO	0	12:00	15000	106.0	15000	13:00	0	SL	4
					109.0	1000	11:00	0	LO	2

Initial State

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
6	MO	0	14:00	1000	0.0					

Aggressive Order

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	10:00	1000	105.0					

Final State

Trade Id	Price	Quanity
1	109.0	1000
2	106.0	15000

Trades

B.1.53 Stop Limit Order Test Case - Test 6

Sell message agresses message book with sell stop limit message. Stop message executed

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
					108.0	15000	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2
4	SL	0	13:00	15000	110.0					

Initial State

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	14:00	0	MO	6

Aggressive Order

Buy								Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					109.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	107.0	1000
2	108.0	15000

Trades

B.1.54 Stop Limit Order Test Case - Test 7

Buy stop limit message added. Last trade price exists. Stop message executed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	MO	0	14:00	1000	0.0					
1	LO	0	10:00	1000	107.0					
3	LO	0	12:00	15000	108.0					
					109.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					108.0	15000	13:00	0	SL	5

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					

Final State

Trade Id	**Price**	**Quanity**
1	109.0	1000
2	108.0	15000

Trades

B.1.55 Stop Limit Order Test Case - Test 8

Sell stop limit message added. Last trade price exists. Stop message executed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	14:00	0	MO	4
1	LO	0	10:00	1000	107.0					
					108.0	15000	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
5	SL	0	13:00	15000	110.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					109.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	107.0	1000
2	108.0	15000

Trades

B.1.56 Stop Limit Order Test Case - Test 9

Buy stop limit orders with greatest difference between its stop price and the last traded price will be elected first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
					108.0	2000	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2
5	SL	0	13:00	1000	110.0					
5	SL	0	14:00	500	110.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	15:00	0	MO	4

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					108.0	500	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	107.0	1000
2	108.0	1000
3	108.0	500

Trades

B.1.57 Stop Limit Order Test Case - Test 10

Sell stop limit orders with greatest difference between its stop price and the last traded price will be elected first

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
3	LO	0	12:00	2000	108.0	1000	13:00	0	SL	4
					108.0	500	14:00	0	SL	5
					109.0	1000	11:00	0	LO	2

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
6	MO	0	15:00	1000	0.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
3	LO	0	12:00	500	108.0					

Final State

Trade Id	**Price**	**Quanity**
1	109.0	1000
2	108.0	1000
3	108.0	500

Trades

B.1.58 Stop Limit Order Test Case - Test 11

Multiple buy stop limit orders with the same difference between its stop price and the last traded price. Oldest executed first

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
					108.0	2000	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2
5	SL	0	13:00	1000	110.0					
5	SL	0	14:00	500	110.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	1000	15:00	0	MO	4

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					108.0	500	12:00	0	LO	3
					109.0	1000	11:00	0	LO	2

Final State

Trade Id	**Price**	**Quanity**
1	107.0	1000
2	108.0	1000
3	108.0	500

Trades

B.1.59 Stop Limit Order Test Case - Test 12

Multiple sell stop limit orders with the same difference between its stop price and the last traded price. Oldest executed first

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
3	LO	0	12:00	2000	108.0	500	13:00	0	SL	4
					108.0	1000	14:00	0	SL	5
					109.0	1000	11:00	0	LO	2

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
6	MO	0	15:00	1000	0.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	107.0					
3	LO	0	12:00	500	108.0					

Final State

Trade Id	**Price**	**Quanity**
1	109.0	1000
2	108.0	500
3	108.0	1000

Trades

B.1.60 Filter And Uncross Test Case - Test 1

Example13. Filtering Algorithm. No Executions. Order 11 MES not satisfied

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
					63.0	1000	12:30	0	LO	10
					64.0	10000	12:00	5000	HO	9
					65.0	12000	11:30	7000	HO	8
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Initial State

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
11	HO	4000	13:00	10000	63.0					

Aggressive Order

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
11	HO	4000	13:00	10000	63.0	1000	12:30	0	LO	10
					64.0	10000	12:00	5000	HO	9
					65.0	12000	11:30	7000	HO	8
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Final State

Trade Id	Price	Quanity
Zero trades executed		

Trades

B.1.61 Filter And Uncross Test Case - Test 2

Example14. Filtering Algorithm. No Executions possible

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
11	HO	4000	13:00	10000	63.0	1000	12:30	0	LO	10
					64.0	10000	12:00	5000	HO	9
					65.0	12000	11:30	7000	HO	8
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Initial State

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
12	HO	25000	13:30	30000	65.0					

Aggressive Order

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
11	HO	4000	13:00	10000	63.0	1000	12:30	0	LO	10
					64.0	10000	12:00	5000	HO	9
12	HO	25000	13:30	30000	65.0	12000	11:30	7000	HO	8
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Final State

Trade Id	Price	Quanity
Zero trades executed		

Trades

B.1.62 Filter And Uncross Test Case - Test 3

Example15. Filtering and uncrossing. 4 trades created

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
11	HO	4000	13:00	10000	63.0	1000	12:30	0	LO	10
					64.0	10000	12:00	5000	HO	9
12	HO	25000	13:30	30000	65.0	12000	11:30	7000	HO	8
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					63.0	2500	14:00	0	LO	13

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	1000	09:00	3000	61.0					
4	HO	6000	09:30	15000	62.0					
11	HO	4000	13:00	10000	63.0					
					66.0	3000	11:00	1000	HO	7
					67.0	500	10:30	0	LO	6
					68.0	1000	10:00	0	LO	5

Final State

Trade Id	**Price**	**Quanity**
1	65.0	1000
2	65.0	2500
3	65.0	10000
4	65.0	12000

Trades

B.1.63 Filter And Uncross Test Case - Test 4

Example16. Filtering and uncrossing. 4 trades created

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	2500	09:00	3000	61.0	2000	11:30	0	LO	12
4	HO	4500	09:30	15000	62.0	11000	12:00	0	LO	13
11	HO	5000	10:00	10000	63.0	12000	12:30	0	LO	14
					67.0	500	11:00	0	LO	6
					68.0	1000	10:30	0	LO	5

Initial State

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
					60.0	2000	13:00	1000	HO	15

Aggressive Order

Order	Type	Buy MES	Time	Size	Price	Size	Time	Sell MES	Type	Order
1	LO	0	08:00	1000	59.0					
2	LO	0	08:30	500	60.0					
3	HO	2500	09:00	3000	61.0					
4	HO	4500	09:30	10000	62.0					
					63.0	12000	12:30	0	LO	14
					67.0	500	11:00	0	LO	6
					68.0	1000	10:30	0	LO	5

Final State

Trade Id	Price	Quanity
1	62.0	2000
2	62.0	2000
3	62.0	6000
4	62.0	5000

Trades

B.1.64 Auction Test Case - Test 1

Auction. One max volume

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	2500	11:00	0	MO	7
					103.0	6900	09:30	0	LO	4
3	LO	0	09:00	200	104.0					
2	LO	0	08:30	5600	104.5	1000	10:00	0	LO	5
1	LO	0	08:00	10000	105.5					
					106.0	200	10:30	0	LO	6

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	09:00	200	104.0					
2	LO	0	08:30	5200	104.5					
					106.0	200	10:30	0	LO	6

Final State

Trade Id	**Price**	**Quanity**
1	104.5	6900
2	104.5	2500
3	104.5	600
4	104.5	400

Trades

B.1.65 Auction Test Case - Test 2

Auction. Two max volume. Min surplus selected

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	2500	11:00	0	MO	7
					103.0	6900	09:30	0	LO	4
3	LO	0	09:00	200	104.0	1000	10:00	0	LO	5
2	LO	0	08:30	5600	104.5					
1	LO	0	08:00	10000	105.5					
					106.0	200	10:30	0	LO	6

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	09:00	200	104.0					
2	LO	0	08:30	5200	104.5					
					106.0	200	10:30	0	LO	6

Final State

Trade Id	**Price**	**Quanity**
1	104.5	6900
2	104.5	2500
3	104.5	600
4	104.5	400

Trades

B.1.66 Auction Test Case - Test 3

Auction. Two max volume. Same surplus. Max price selected

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					0.0	2500	11:00	0	MO	7
					103.0	6900	09:30	0	LO	4
3	LO	0	09:00	1000	104.0	1000	10:00	0	LO	5
2	LO	0	08:30	5600	105.0					
1	LO	0	08:00	10000	105.5					
					106.0	200	10:30	0	LO	6

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
3	LO	0	09:00	1000	104.0					
2	LO	0	08:30	5200	105.0					
					106.0	200	10:30	0	LO	6

Final State

Trade Id	**Price**	**Quanity**
1	105.0	6900
2	105.0	2500
3	105.0	600
4	105.0	400

Trades

B.1.67 Cancel Order Test Case - Test 1

Cancel limit order bid in empty order book. Cancel request rejected

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	0	100.0					

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.68 Cancel Order Test Case - Test 2

Cancel limit order offer in empty order book. Cancel request rejected

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	0	10:00	0	LO	1

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.69 Cancel Order Test Case - Test 3

Cancel limit order bid. Bid removed

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	10:00	0	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.70 Cancel Order Test Case - Test 4

Cancel limit order offer. Offer removed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	0	10:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.71 Cancel Order Test Case - Test 5

Cancel stop order bid. Bid removed

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	SO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SO	0	10:00	0	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.72 Cancel Order Test Case - Test 6

Cancel stop order offer. Offer removed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	SO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	0	10:00	0	SO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.73 Cancel Order Test Case - Test 7

Cancel stop limit order bid. Bid removed

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	SL	0	12:00	12000	106.0					
3	LO	0	13:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	10:00	0	106.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	13:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.74 Cancel Order Test Case - Test 8

Cancel stop limit order offer. Offer removed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					106.0	12000	12:00	0	SL	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					107.0	0	10:00	0	SL	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.75 Cancel Order Test Case - Test 9

Cancel hidden order bid. Bid removed

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	500	11:10	500	50.0					
1	LO	0	10:00	1000	100.0					
3	LO	0	13:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	0	10:00	0	50.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	13:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.76 Cancel Order Test Case - Test 10

Cancel hidden limit order offer. Offer removed

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					106.0	500	11:10	500	HO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					106.0	0	10:00	0	HO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.77 Replace Order Test Case - Test 1

Replace limit order bid in empty order book. Replace request rejected

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					

Aggressive Order

	Buy							Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.78 Replace Order Test Case - Test 2

Replace limit order offer in empty order book. Replace request rejected

	Buy					Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Initial State

	Buy					Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1

Aggressive Order

	Buy					Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.79 Replace Order Test Case - Test 3

Replace limit order bid quantity. Quantity replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	1500	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	1500	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.80 Replace Order Test Case - Test 4

Replace limit order bid GTD. GTD replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.81 Replace Order Test Case - Test 5

Replace limit order bid GTT. GTT replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.82 Replace Order Test Case - Test 6

Replace limit order offer quantity. Quantity replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	1500	11:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	1500	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.83 Replace Order Test Case - Test 7

Replace limit order offer GTD. GTD replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.84 Replace Order Test Case - Test 8

Replace limit order offer GTT. GTT replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	0	LO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.85 Replace Order Test Case - Test 9

Replace limit order bid quantity. Quantity > existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0	2500	13:00	2500	HO	4
3	LO	0	12:00	1000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	2500	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	1000	107.0					

Final State

Trade Id	**Price**	**Quanity**
1	105.0	2500

Trades

B.1.86 Replace Order Test Case - Test 10

Replace limit order offer quantity. Quantity < existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
4	HO	2500	13:00	2500	105.0	2000	11:00	0	LO	2
					107.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2500	11:00	0	LO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	105.0	2500

Trades

B.1.87 Replace Order Test Case - Test 11

Replace limit order bid price. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	LO	0	11:00	2000	105.0					
3	LO	0	12:00	1000	107.0					
					108.0	2000	13:00	0	LO	4

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	LO	0	11:00	2000	108.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	1000	107.0					

Final State

Trade Id	**Price**	**Quanity**
1	108.0	2000

Trades

B.1.88 Replace Order Test Case - Test 12

Replace limit order offer price. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	2000	90.0					
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	0	LO	2
					107.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	2000	11:00	0	LO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	90.0	2000

Trades

B.1.89 Replace Order Test Case - Test 13

Replace HO order bid quantity. Quantity replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	2000	11:00	4000	105.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	4000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.90 Replace Order Test Case - Test 14

Replace HO order bid GTD. GTD replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	2000	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.91 Replace Order Test Case - Test 15

Replace HO order bid GTT. GTT replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	2000	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.92 Replace Order Test Case - Test 16

Replace HO order bid MES. MES replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	1000	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	1000	11:00	2000	105.0					
3	LO	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.93 Replace Order Test Case - Test 17

Replace HO order offer quantity. Quantity replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	1000	HO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	1500	11:00	1000	HO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	1500	11:00	1000	HO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.94 Replace Order Test Case - Test 18

Replace HO order offer GTD. GTD replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	3000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	2000	HO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.95 Replace Order Test Case - Test 19

Replace HO order offer GTT. GTT replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	3000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	2000	HO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.96 Replace Order Test Case - Test 20

Replace HO order offer MES. MES replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	3000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	1000	HO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	1000	HO	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.97 Replace Order Test Case - Test 21

Replace HO order bid quantity. Quantity > existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0	2500	13:00	2500	HO	4
3	LO	0	12:00	1000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	2500	11:00	2500	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	1000	107.0					

Final State

Trade Id	**Price**	**Quanity**
1	105.0	2500

Trades

B.1.98 Replace Order Test Case - Test 22

Replace HO order offer quantity. Quantity < existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
4	HO	2500	13:00	2500	105.0	2000	11:00	2000	HO	2
					107.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2500	11:00	2500	HO	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	105.0	2500

Trades

B.1.99 Replace Order Test Case - Test 23

Replace HO order bid price. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
2	HO	2000	11:00	2000	105.0					
3	LO	0	12:00	1000	107.0					
					108.0	2000	13:00	0	LO	4

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	HO	2000	11:00	2000	108.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	LO	0	10:00	1000	100.0					
3	LO	0	12:00	1000	107.0					

Final State

Trade Id	**Price**	**Quanity**
1	108.0	2000

Trades

B.1.100 Replace Order Test Case - Test 24

Replace HO order offer price. Order re-aggress the order book

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
4	LO	0	13:00	2000	90.0					
					100.0	1000	10:00	0	LO	1
					105.0	2000	11:00	2000	HO	2
					107.0	1000	12:00	0	LO	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					90.0	2000	11:00	2000	HO	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	LO	1
					107.0	1000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
1	90.0	2000

Trades

B.1.101 Replace Order Test Case - Test 25

Replace Stop order bid limit price. Limit price replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	2000	105.0					
3	SL	0	12:00	3000	105.0					

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	107.0					

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
3	SL	0	12:00	3000	105.0					
2	SL	0	11:00	2000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.102 Replace Order Test Case - Test 26

Replace Stop order offer limit price. Limit price replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					107.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					107.0	1500	11:00	0	SL	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					107.0	1500	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.103 Replace Order Test Case - Test 27

Replace Stop order bid quantity. Quantity replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	2000	105.0					
3	SL	0	12:00	3000	105.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	1500	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	1500	105.0					
3	SL	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.104 Replace Order Test Case - Test 28

Replace Stop order bid GTD. GTD replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	105.0					
1	SL	0	10:00	1000	107.0					
3	SL	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	2000	105.0					
3	SL	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.105 Replace Order Test Case - Test 29

Replace Stop order bid GTT. GTT replaced

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	2000	105.0					
3	SL	0	12:00	3000	107.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	105.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
1	SL	0	10:00	1000	100.0					
2	SL	0	11:00	2000	105.0					
3	SL	0	12:00	3000	107.0					

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.106 Replace Order Test Case - Test 30

Replace Stop order offer quantity. Quantity replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	1500	11:00	0	SL	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	1500	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.107 Replace Order Test Case - Test 31

Replace Stop order offer GTD. GTD replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	0	SL	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	LO	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.108 Replace Order Test Case - Test 32

Replace Stop order offer GTT. GTT replaced

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Initial State

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					105.0	2000	11:00	0	SL	2

Aggressive Order

Buy						Sell				
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					100.0	1000	10:00	0	SL	1
					105.0	2000	11:00	0	SL	2
					107.0	3000	12:00	0	SL	3

Final State

Trade Id	**Price**	**Quanity**
Zero trades executed		

Trades

B.1.109 Replace Order Test Case - Test 33

Replace Stop order bid quantity. Quantity > existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	90.0	1000	13:00	0	LO	4
1	LO	0	10:00	1000	100.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	1000	90.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
1	100.0	1000
2	90.0	1000

Trades

B.1.110 Replace Order Test Case - Test 34

Replace Stop order offer quantity. Quantity < existing quantity. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	2000	11:00	0	SL	2
3	LO	0	12:00	1000	90.0	1000	10:00	0	LO	1
4	LO	0	13:00	1000	90.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	1000	11:00	0	SL	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000

Trades

B.1.111 Replace Order Test Case - Test 35

Replace Stop order bid stop price. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	90.0	1000	13:00	0	LO	4
1	LO	0	10:00	1000	100.0	1000	12:00	0	LO	3

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
2	SL	0	11:00	2000	90.0					

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
1	100.0	1000
2	90.0	1000

Trades

B.1.112 Replace Order Test Case - Test 36

Replace Stop order offer stop price. Order re-aggress the order book

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	2000	11:00	0	SL	2
3	LO	0	12:00	1000	90.0	1000	10:00	0	LO	1
4	LO	0	13:00	1000	90.0					

Initial State

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
					80.0	2000	11:00	0	SL	2

Aggressive Order

		Buy						Sell		
Order	**Type**	**MES**	**Time**	**Size**	**Price**	**Size**	**Time**	**MES**	**Type**	**Order**
Empty LOB										

Final State

Trade Id	**Price**	**Quanity**
1	90.0	1000
2	90.0	1000

Trades

C Appendix

C.1 Getting Started Guide

C.1.1 Download Java

The Java version should be version 8+. Check the Java version on the server

```
$ java -version
java version "1.8.0_60"
```

C.1.2 Clone the repository

```
git clone https://dharmeshsing@bitbucket.org/dharmeshsing/jsematchingengine.git
```

C.1.3 Build the project

```
gradle build -x test
```

C.1.4 Location of configuration files

The configuration files are located in the data directory. It contains the following files:

- **clientData.csv**: The client details.
- **hawkesData.properties**: The Hawkes input data.
- **Stock.csv**: The stock details.
- **Trader.csv**: The trader details.
- **tradingSessionCron.properties**: The cron expressions for the trading sessions.

The start and stop scripts are located in the scripts directory.

C.1.5 Location of connection files

The connection files contain the directory paths to store files and the UDP details for each module. The files are:

- **local.properties**: Connection details for testing on your laptop/desktop.

- **witsServer.properties**: Connection details for deploying on the university's Linux server.
- **windows.properties**: Connection details for deploying on the university's Windows server.

C.1.6 Deploy

```
gradle -PenvProp=local.properties -PsoftwarePath=/tmp clean installDist
    bootRepackage copyResourcesToInstallDir copyToDeploy deleteDeployZip
    deployZip deployToWitsServer
```

C.1.7 Start the Gateways on the server

Go to the deployed software path scripts directory *i.e.* /tmp/scripts.

```
./startAll.sh
```

C.1.8 Simple client

The client login details must be defined in the clientData.csv file.

```
//These values should be stored and retrevied from the clientData.csv file
String url = udp://localhost:5000
int streamId = 10
int compId = 1
String password = test111111

//Connect to the Trading Gateway
GatewayClientImpl tradingGatewayPub = new GatewayClientImpl();
tradingGatewayPub.connectInput(url,streamId);

//Login to the Trading Gateway
LogonBuilder logonBuilder = new LogonBuilder();
DirectBuffer buffer = logonBuilder.compID(compId)
.password(password.getBytes())
.newPassword(password.getBytes())
.build();

tradingGatewayPub.send(buffer);
```

C.1.9 Send a new order to the Trading Gateway

```
public DirectBuffer createNewOrder(long volume,
                                   long price,
                                   SideEnum side,
                                   OrdTypeEnum orderType){

NewOrderBuilder newOrderBuilder = new NewOrderBuilder();

DirectBuffer directBuffer = newOrderBuilder.compID(clientData.getCompID())
            .clientOrderId("1234".getBytes())
            .account("account123".getBytes())
            .capacity(CapacityEnum.Agency)
            .cancelOnDisconnect(CancelOnDisconnectEnum.DoNotCancel)
            .orderBook(OrderBookEnum.Regular)
            .securityId(securityId)
            .traderMnemonic("John".getBytes())
            .orderType(orderType)
            .timeInForce(TimeInForceEnum.Day)
            .expireTime("20150813-23:00:00".getBytes())
            .side(side)
            .orderQuantity((int) volume)
            .displayQuantity((int) volume)
            .minQuantity(0)
            .limitPrice(price)
            .stopPrice(0)
            .build();
    return directBuffer;
}

//Send new order to Trading Gateway
tradingGatewayPub.send(createNewOrder(1200, 25034, SideEnum.Buy, OrdTypeEnum.Limit
    ));
```