



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

Estimation of Clusters based on Decision Latency in High Frequency Trading

Tanmay Bagla



Supervised by Professor Khurshid Ahmad

September 2020

A DISSERTATION REPORT

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

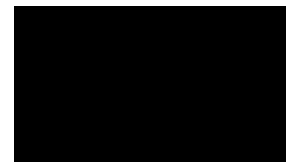
MASTER OF SCIENCE IN COMPUTER SCIENCE (DATA SCIENCE)

SCHOOL OF COMPUTER SCIENCE & STATISTICS

TRINITY COLLEGE DUBLIN, IRELAND

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.



Tanmay Bagla

September 11, 2020

Permission to Lend or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Tanmay Bagla

September 11, 2020

Acknowledgments

Firstly, I would like to articulate my sincerest gratitude to Professor Khurshid Ahmad, whose constant encouragement was essential in making this thesis a reality. His in-depth knowledge and vast experience in the field of computer science and statistics helped me solidifying my concepts and overcome difficulties faced in this research. Without his contribution results would not look as promising as they are now.

Secondly, I am grateful to Professor Simon Wilson for his time and valuable suggestions on the improvement of this research. A big credit goes to the Department of Computer Science and Statistics, Trinity College Dublin, for providing me with adequate cloud infrastructure to work on my thesis.

Lastly, I am grateful to my family for supporting me during my MSc program at Trinity College Dublin and special thanks to my colleague Mr Vishal Kumar for helping me with all the technicalities of Latex.

Tanmay Bagla

Abstract

One of the critical factors to evaluate a system or network performance is latency. Latency is how fast a system responds to the input requests. Within the financial market low decision latency means high-profit margins. Decision latency means that in a streaming data environment where people have to make decisions within a unit interval of time (by the second, minute, hour, and so on), the decision relates to the delivery of a result well before the next data arrives. The other challenge financial institutions face is scalability. With changing size of the incoming data, it is not easy for people to make real-time, accurate decisions. Scalability is the ability of the computing system to handle change in the size of the data while solving complex problems.

This dissertation aims to provide a big data architecture that focuses on providing solutions to both problems of scalability and latency. Based on the decision latency and the volume of the data, this research will provide estimations of the computing clusters. The key results states that the average reduction in decision latency on switching the computation from modeling time dependent volatility to descriptive statistics is a whopping 105.97%. This means that the time it takes to compute time dependent volatility is almost double the time it takes to compute historical volatility through descriptive statistics. In both cases, it is common to see that the speed at which the performance in execution is improved is decreasing with an increase in the size of the input dataset. This is worth noting that the solution to the problem of scalability is the solution to the problem of latency itself. As scaling the size of computing machines or clusters will handle latency issues for both storage and processing of big data.

Keywords: Decision Latency, High Frequency Trading, Big Data, AWS, Spark, Time Series, Parallel Computing, volatility, ARCH, GARCH

Word Count: 21130

Contents

List of Figures	VIII
List of Tables	XIII
1 Introduction	1
1.1 Overview	1
1.2 Research Objective	5
1.3 Research Challenges	6
1.4 Contributions	6
1.5 Research Progression	7
1.6 Thesis Structure	8
1.7 Conclusion	9
2 Motivation and Literature Review	10
2.1 Introduction	10
2.2 High Frequency Trading	11
2.2.1 Evolution of high frequency trading	11
2.2.2 The role of speed in HFT	11
2.2.3 How does latency Impact Performance	12
2.2.4 High Frequency Returns and its Stylized Facts	13
2.3 Need for big data Solutions	18
2.4 Risk Estimation	22
2.4.1 Different Interpretations to Volatility	22
2.4.2 Time Series Volatility Modelling	24
2.5 Motivation	25

3	Design and Methods	26
3.1	Introduction	26
3.2	System Architecture	26
3.3	Parallel Computing	28
3.3.1	Apache Hadoop	28
3.3.2	Apache Spark	30
3.3.3	Limitations of Hadoop MapReduce	33
3.3.4	Hadoop On-Premise vs. Hadoop on Cloud	34
3.4	AWS Cloud Solutions	36
3.4.1	Real-Time Streaming	36
3.4.2	AWS EMR (Elastic MapReduce) Big Data Platform	39
3.5	High Frequency Methods	43
3.6	Volatility	47
3.6.1	Realized Volatility (Historical Volatility)	47
3.6.2	Conditional Volatility	48
3.6.3	Time Series Volatility Models	48
3.7	Important Classes of Spark SQL and DataFrames	50
4	Implementation	51
4.1	Experiment 1	51
4.2	Experiment 2	53
4.3	Data Discovery and Acquisition	54
4.3.1	Historical Data Acquisition	55
4.3.2	Data Partitioning and File Storage Format	58
4.3.3	Real-time Data Extraction from Amazon Kinesis Streams Using Zeppelin and Spark Streaming	59
4.4	Environment Setup	62
4.4.1	Security Configuration	63
4.4.2	Setting up a Spark cluster on AWS EMR	64
4.5	Risk Estimation	67
4.5.1	Summarized Statistics	67
4.5.2	Volatility Model	68
4.6	Decision Latency	69

4.7	Conclusion	70
5	Results	71
5.1	Introduction	71
5.2	Stylized Facts for Financial Returns	72
5.2.1	First stylized fact	73
5.2.2	Second stylized Fact	75
5.2.3	Third stylized Fact	75
5.2.4	Latency in Stylized Facts	77
5.3	Experiment 1 Results	78
5.3.1	Experiment 1	78
5.3.2	Cluster Estimation based on Decision Latency	80
5.3.3	Change in Decision Latency based on Data Points	83
5.4	Experiment 2 Results	85
5.4.1	Experiment 2	85
5.4.2	Cluster Estimation based on Decision Latency	95
5.4.3	Change in Decision Latency based on Data Points	98
5.5	Combined Evaluation of Experiment 1 and 2	100
5.6	Status of Results	102
5.7	Conclusion	102
6	Conclusion and Future Work	103
6.1	Conclusion	103
6.2	Future Work	105
	References	107

List of Figures

1.1	Minute Intraday Returns of S&P 500 Mini Futures from the year 2010 to the year 2019. Total number of data points in this 1 minute dataset is nearly 3.4 million. Normally at points where returns are more than 4% or 5% above or below zero (mean) accounts for high intraday volatility. . . .	3
1.2	Five V's of Big Data.	4
2.1	Autocorrelations for intraday absolute S&P 500 returns.	18
2.2	Reference component architecture of Financial Business Cloud. Cloud Manager is the only source connected to the stock exchange and data providers. ESB (Enterprise Service Bus) is like a cluster manager in the Hadoop framework responsible for managing resources, distributing tasks, etc. It provides intelligent routing to perform tasks such as real-time analytics, run time risk management, or trade execution.	21
3.1	System Architecture Overview. Three major steps involved are data collection and aggregation, risk estimation using unconditional and conditional volatility models, analysis of decision latency.	27
3.2	Hadoop Core Components	29
3.3	The MapReduce programming model. K elements represent the keys in pairs.	29
3.4	Spark Framework - Spark Core as the foundation for the platform, Spark SQL for interactive queries, Spark streaming for real-time analytics, Spark MLlib for machine learning and GarphX for graph processing.	31
3.5	Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.	32

3.6	Direct Acyclic Graph Architecture. Each square is a task, inside which nodes are present for processing information. Arrows connecting nodes represent the data flow between nodes and the vertexes in the graph, dashed lines represent the dependencies between data blocks (cylinders) .	33
3.7	Data distribution in shards.	37
3.8	Kinesis Stream Records Components.	38
3.9	Kinesis Architecture	39
3.10	AWS EMR (Elastic MapReduce) Architecture	39
3.11	Apache SPARK Framework used for processing Big Data distributed over AWS EMR clusters. On launching a spark application, the Spark Driver breaks the program into multiple tasks and distribute them over spark executors. The Spark Driver has to request cluster manager to start the jobs over executors. The Spark Executors perform the tasks received from the Spark Driver. The Distributed Storage Layer is based on the Hadoop API and holds the distributed dataset, which is partitioned across hard drives of the Spark worker nodes	42
3.12	The shape of the S&P500 return distribution explains the fact that the observed standardized returns are approximately symmetric has a fat tail and high peak distribution, which is not the same as the normal distribution.	45
4.1	Flowchart Representation of experiment 1 – To observe decision latency in computing Descriptive Statistics and measure historical volatility by calculating standard deviation.	52
4.2	Flowchart Representation of experiment 2. To observe decision latency in computing time series volatility models namely ARCH (1) and GARCH (1,1)	53
4.3	S&P 500 Mini Futures (ES) 1-minute intraday prices from the year 2010-2019	57
4.4	AWS Data Permissions	59
4.5	AWS S3 Object Storage Data Encryption	59
4.6	Real-Time Data Extraction and Analysis using AWS Kinesis Framework	60
4.7	Data Aggregation (Historical and Real Time) at every 1-minute.	62

4.8	AWS Security Settings. SSH lets user login to EC2 master Ubuntu machine using Putty. Custom TCP port 8890 is the default port to open Zeppelin Notebook running on distributed platform.	64
4.9	Selection of the EMR type clusters. Necessary packages such as Spark, Hadoop, Zeppelin or optional Hive are installed on EC2 machines. Note EMR cluster is just collection of EC2 machines running concurrently. . .	65
4.10	Screenshot of AWS interface showing selection of clusters mentioned in Table 4.5	66
4.11	EMR clusters (2 Cores or 4 Instances) running with installed Spark framework.	67
4.12	Output Window for one of the use case in experiment 1 - Descriptive Statistics (Measuring Historical Volatility)	68
4.13	Output Window for one of the use case in experiment 2. Modelling volatility using 400k datapoints	69
5.1	Intraday Future Standardized Returns of S&P 500 Mini from 2019-01-01 to 2019-03-11. At many points, returns are deviating by more than 5 standard deviation from the mean which accounts for high volatility. . . .	72
5.2	Return Distribution. The distribution has a very long tail and normalised returns reach to maximum 15 standard deviations from the mean. The high peak describes the maximum returns are concentrated within 0.5 standard deviations (see Table 5.2)	74
5.3	Autocorrelation summary of Table 5.3 but with up to 40 lags. A plot of the autocorrelation of a time series by lag is called the Autocorrelation Function, or the acronym ACF. This plot is sometimes called a correlogram or an autocorrelation plot. Running the example creates a 2D plot showing the lag value along the x-axis and the correlation on the y-axis between -1 and 1. The greater the distance in time, the more unlikely that autocorrelation exists	77

5.4	Descriptive Statistics of 1 Minute Returns for S&P 500 Mini Future Series. The mean of the 1-minute high-frequency return dated from the year 2010-01-03 to 2013-01-02 is near zero with a standard deviation of 0.039%. The series has high Kurtosis of 99.009 and negative skewness of -0.29. Negative skewness indicates that the left tail is stronger and longer than the right tail. Also, the left tail is stronger, and the mass of the distribution is more concentrated on the right side as compared to a normal distribution. . . .	79
5.5	Graphs (A-G) describing the relationship between execution time and number of clusters. It is clearly observed that increasing the size of cluster reduces decision latency, after which the logarithm curve leads towards a saturation point post which there cannot be more improvement in performance even with increase in size of cluster. There is a little scope in improvement of decision latency when jumping from 8 clusters to 16 clusters.	82
5.6	Graphs (A, B, and C) showing relation between decision latency and data points on each given cluster. Latency is increased with an increase in size of the data.	84
5.7	ARCH Model Summary Statistics	87
5.8	GARCH (1,1) Model Output	91
5.9	GARCH (1,1) Residual Variance Forecast of 1 minute ahead	94
5.10	Actual vs Predicted Daily Volatility. Summation of squared intraday returns aggregated to the daily level, termed realised variance, as a measure of that day's variance is more accurate than 1-minute ahead intraday forecasts	95
5.11	Actual vs Predicted 1-minute volatility forecast. 1 Minute variance forecast results in noisy measurement. The reason for this is that the GARCH takes squared returns into consideration while modelling volatility. $R_t = \sigma_t \cdot z_t$ where z_t denotes an independent mean zero and t denotes latent volatility. Squared returns cause noisy measurements due to the idiosyncratic error term z_t^2	95

5.12	Graphs(A-F) showing the relation between decision latency and data points on each given size of a cluster. The logarithm fit curve shows that a saturation point will be reached as the latency reduces further as it cannot pass beyond a certain threshold level.	98
5.13	Graph(A-C) showing the relation between decision latency and data points on each given cluster. On a logarithmic graph of decision latency, even though the size of data is increasing along with size of the clusters, a point is observed where the rate of growth starts to level off when that exponential growth has stopped (see Figure 5.12).	100

List of Tables

2.1	Average Frequencies for standardized returns.	16
3.1	Difference between Hadoop MapReduce and Spark.	34
3.2	PySpark classes used in the research.	50
4.1	Spot vs. Futures Prices	55
4.2	S&P 500 Mini Futures (ES) 1-minute intraday prices from the year 2010-2019	57
4.3	Data Scaling	58
4.4	Cluster Scaling	58
4.5	Details of Instance used in the research along with its pricing.	65
5.1	Descriptive Statistics on log-returns of S&P 500 Mini Future series for nearly three months dated 2019-01-01 to 2019-03-11. Note that returns are not normally distributed, have fat tails and high peak. The mean is near zero. Skewness is slightly greater than zero, unlike in normal distribution where it is zero. High Kurtosis of 15.25 explains the heavy tails, distinct peakedness near the mean. It is much higher than the Kurtosis of a normal distribution which is 3.	73
5.2	Frequency for standardized returns of S&P 500 Mini Futures. The frequency of observations lies below $\bar{r} - 3s$ or above $\bar{r} + 3s$ is approximately six times as it of the normal distribution, corresponding to the high value of Kurtosis	74

5.3	Autocorrelation in S&P 500 Mini Futures returns from 2019-01-01 to 2019-03-11. Proving second and third stylized fact that prices are highly autocorrelated, log returns are not autocorrelated or close to zero, squared returns are positively autocorrelated, and lastly absolute returns have high positive autocorrelation than squared returns. There is a very high correlation between prices at time t and $t+1$. The correlation between log return r_t and r_{t+1} is negative and continues for a few lags after which it becomes positive proving that return reverts to mean. There is a positive dependence between absolute returns on nearby days, and likewise for squared returns.	76
5.4	Summary Statistics for Future Returns computed on 1 million data points. \bar{r} , s , b , and k are the mean, standard deviation, skewness, and Kurtosis for the time series.	79
5.5	Decision Latency for experiment 1 on different sizes of data and clusters.	81
5.6	Decision latency on a different scale of data inputs. Every time the data size is doubled execution time increases by some specific percentage	84
5.7	Decision Latency table for experiment 2 on different scales of data sizes and clusters.	96
5.8	Decision latency on a different scale of data inputs. Every time the data size is doubled execution time increases by a definite percentage.	99
5.9	Combined evaluation of execution time for both experiment 1 and experiment 2 based on Decision Latency. Example includes that with data size as 1 million or more it is not possible to compute both summary statistics and ARCH models within 1-minute interval. This might be possible on scaling cluster size from 8 to 16.	101

Code Listings

4.1	Code for Futures contracts data extraction through IEXCloud Web API from the year 2019-2020. Similarly, more data is extracted from the year 2010 – 2019	56
4.2	KPL application written in Python to send records to shards aggregated up to 100KB, with every 60 seconds and joined with ‘\n’	61
4.3	Install python libraries.sh	66

Chapter 1

Introduction

1.1 Overview

In today's fast-moving, data-driven economic environment, it is highly significant to make responsible and accurate decisions within a quick interval of time. In High Frequency Trading (HFT), the frequency of the data is so high that it is difficult for a human trader to make decisions in seconds or even in milliseconds. **High Frequency Trading** is a type of algorithmic trading that relies on cutting-edge technology infrastructure to strive in terms of fast turnover rates that exploit large volumes of financial information. (Aldridge, 2013). Speed is critical for traders because of the fundamental intrinsic uncertainty of the securities called volatility, and (De Luca, 2006) volatility plays a crucial role in decision making for risk management, asset allocation, and asset pricing. A millisecond decline in a decision latency may support an HFT firm's revenue by around 100 million each year, furthermore encourages a firm to increase incredible rivalry advantage (Shamgar, 2014). To compete for speed, which creates more and more profit, HFT firms locate computers close to the trading exchanges with a motive to reduce the latency of order submissions. Moreover, the high frequency time series allows us to learn how prices react to dynamic information.

The concept of **decision latency** alludes to the way that in a real-time streaming data environment, we need to settle on accurate financial decisions within a unit time period

(constantly, moment, hour, etc.) before the data arrives in the next interval. Decision latency intends to measure the response time engaged with a decision to trade in response to a market event that is very informative to the HFT firms (Baron et al., 2018). Typically, the decisions are based on the rapid fluctuation of data streams such as the price of stocks or commodities, temperature in a plant, brain waves, and so on. This fluctuation is an indication of risk, which can result in a huge loss of investment. In work presented here, volatility is modelled to forecast risk in a minute interval HFT environment.

Oxford dictionary defines volatility (France, 1792) as flightiness or a lack of steadiness. In the financial market, volatility is the statistical measure of price variability over a definite period of time. It can be related to a stochastic number of intraday price revisions. (Stephen J Taylor, 2007). The asset returns are one of the most important characteristics to measure market risk (Ladokhin, 2009). The asset returns are considered to be random variables. These returns fluctuate unevenly with a spread termed as volatility, which is used extensively in many financial applications such as pricing financial derivatives and portfolio risk management. With HFT more observations enable us to and forecast volatility, which benefits both the derivatives trader and portfolio managers. Quantitative models are used to forecast valuable market trends and are used widely in financial institutions. The enthusiasm for high frequency information was, to a great extent, provided by (Andersen and Bollerslev, 1998), who utilized the concept of realized variance to show that standard volatility models can be used perfectly to deliver accurate forecasts. There are immense opportunities in HFT, but in order to utilize those opportunities, intraday volatility patterns need to be modelled accurately.

Modelling intraday volatility requires storing and analyzing huge volume of data generated over minutes, seconds, or milliseconds interval. This is the reason we need big data processing systems that allow quick retrieval of real-time data, secured storage, and high-speed computation. Today with the advent of unstructured data such as news in the form of audio, video, social media posts, and articles, there is a dire need of creating a big data pipeline, which can be fed into algorithms to make accurate decisions.

Though the concept of big data is relatively new, the origin of larger datasets goes back to the 1960s and 1970s with the development of data centers and relational databases. The need to analyze big unstructured data rose in 2005 when more and more users started

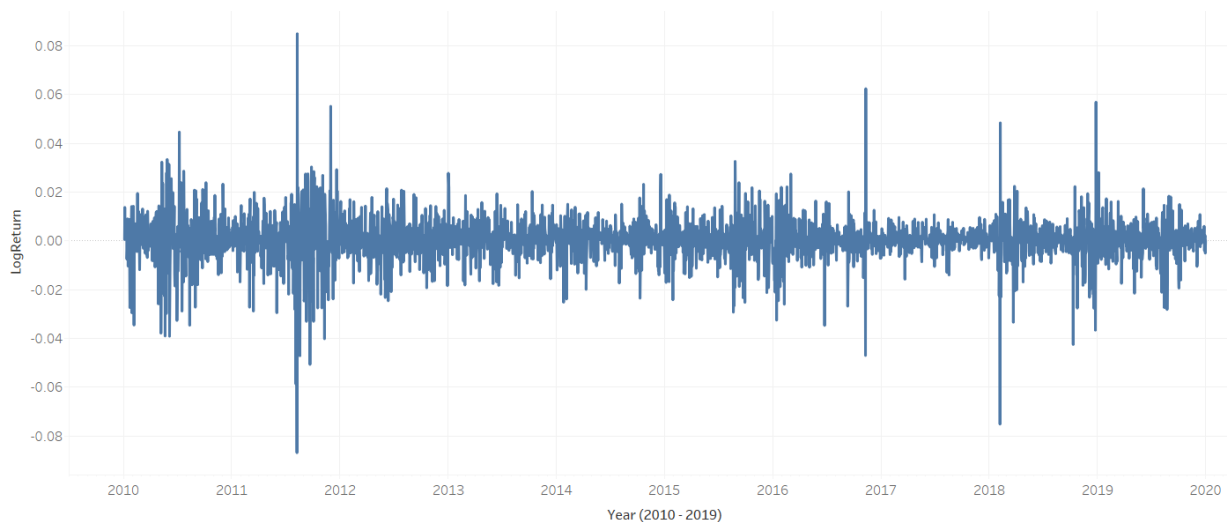


Figure 1.1: Minute Intraday Returns of S&P 500 Mini Futures from the year 2010 to the year 2019. Total number of data points in this 1 minute dataset is nearly 3.4 million. Normally at points where returns are more than 4% or 5% above or below zero (mean) accounts for high intraday volatility.

onboarding to social media platforms such as Facebook, YouTube, Twitter, etc (*Big Data, Oracle Ireland 2020*). At the same time span, the big data parallel computing framework called Hadoop was developed and attracted huge popularity amongst big tech companies. Hadoop made storing and processing of big data a lot easier. Apache Hadoop is an open-source big-data framework providing a platform for handling large data sets through distributed storage and processing (Peng, 2019).

(Gartner, 2012) *Big Data is high-volume, high-velocity, or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision-making, and process automation.*

Other than in HFT, big data also accounts for tracking illegal market activity such as money laundering, financial frauds, illegal trading, etc. The five V's of big data are Volume, Velocity, Value, Variety, and Veracity, as shown in Figure 1.2.

- Velocity – How quickly is the information changing, or data is flowing?
- Veracity answers the question if the data is clean and accurate? What is the source of the data?

- Volume is the base of big data, which is nothing but a huge chunk of data coming from various sources.
- Variety is the information flowing from different sources of data in various formats, such as structured and unstructured.
- Value is the ability to transform big data into business decisions.



Figure 1.2: Five V's of Big Data.

Today with more and more data getting generated from IOT devices or social media platforms, migration from a local file storage system or legacy systems to cloud based data systems has become the need of the hour.

Oxford dictionary defines Cloud Computing (Guardian, 2008) as *the use of networked facilities for the storage and processing of data rather than a user's local computer*. Cloud computing has been very useful in solving big data problems by providing offsite shared servers hosted on the internet for storage, computing, networking, and analysis. These resources provide a scalable and cost-efficient solution. IBM defined cloud computing as the delivery of on-demand computing resources (Huttunen et al., 2019). This paper deals

with using cloud Infrastructure as a Service (IaaS) to solve latency and scalability issues in HFT. IaaS is where Cloud Computing providers invest huge capital in developing High Performance Computing (HPC) infrastructure that is then made accessible to their clients on a “pay for what you need” basis (O’Driscoll, Daugelaite, and Sleator, 2013). Amazon Web Services (AWS) capture 70% of the total IaaS market and provide highly scalable infrastructure to its customers. Major services used in this research are S3 bucket for storage, EMR clusters for cluster computing integrated with Apache Spark for parallel processing, and real-time structured streaming using Kinesis (Varia and Mathew, 2014).

1.2 Research Objective

In order for an algorithm to take a trading decision in an HFT environment with information streaming at such a fast speed from different sources in different formats, there should be a standard secured cloud-based infrastructure. Here I am building an architecture that solves two key problems of big data, i.e., scalability and latency in HFT environment. This infrastructure is then used to compute historical as well as time dependent volatility before estimating clusters based on their execution time. The time it takes to calculate a volatility adds to the latency or delay in producing the results. The system will have less than 1 minute to compute, provide results and take trading decision before the next data arrives.

This research aims to reduce the latency by using a highly advanced open-source distributed general-purpose cluster-computing framework called Apache Spark. The implications of this research can be beneficial to researchers, econometricians, financial analysts or traders facing issues of latency and scalability in a HFT environment.

There are two key points worth noting before moving ahead in this research. Firstly, the whole focus is on calculating decision latency and not on network latency. Network latency ¹ is the time it takes for data to go from the source, such as stock exchange or data providers, to the destination, such as the end-users. Decision latency is the time it takes by a computing machine to execute a certain program. Secondly, the big data

¹[https://en.wikipedia.org/wiki/Latency_\(engineering\)](https://en.wikipedia.org/wiki/Latency_(engineering))

architecture is deployed in the research field of finance, which can just be considered an illustrative research application for measuring latency. The research application is chosen as finance because of the two key problems financial analysts, traders, or researchers face, i.e., latency and scalability, which can both be answered by the designed big data architecture.

1.3 Research Challenges

Challenges faced while implementing the big data solutions in HFT are:

1. Finding literature that covers issues of decision latency in High Frequency Trading or which incorporates big data cloud-based solutions to perform real-time risk analysis using time-dependent volatility models.
2. Searching stable 1-minute high frequency big dataset.
3. Finding powerful computing resources due to the unavailability of university computing resources because of the Covid19 pandemic.
4. Choosing real-time streaming platform best suited for low latency architecture.
5. Fitting GARCH models that best fit the High Frequency data for modelling volatility.
6. Evaluating the accuracy of the volatility models due to limited knowledge in the research application, i.e., high frequency trading.

1.4 Contributions

Summarizing the key contributions as below:

1. Researched existing implications of Big Data in HFT, such as latency and scaling.
2. Studied existing big data cloud-based architectures in the area of HFT.

3. Collection of exemplar time series futures data representing stable market.
4. Set up a cloud-based architecture to perform real-time analysis, including risk computation of a high frequency time series in a unit time interval.
5. Estimation of parallel computing clusters based on decision latencies observed on risk computation on a different scale of data inputs and different sizes of clusters.

1.5 Research Progression

This research started last year in September 2019 when under the guidance of professor Khurshid Ahmad, I started generating synthetic data using the Monte Carlo simulation technique, fulfilling the need for Big Data to analyze historical volatility in 5-minute high frequency returns of SPY ETF through standard deviation. Standard deviation is considered a most efficient measure of volatility when returns are under five standard deviations around the mean. But when the data is used for making decisions in HFT at a minute by minute interval, model certainty needs to be accurate, and the certainty of the results cannot be dependent on the sampled data. Also, in a real-time trading environment, when the standard deviation keeps changing, it is not a good idea to use standard deviation as a measure of volatility. Instead, time series econometric volatility models need to be taken into account for more accurate risk predictions. The idea started with the need to research the availability of real-time high frequency 1-minute data, which then got extracted through API from a trusted web source called IEX cloud.

With a couple of weeks lost in March due to travel emergencies, there was a halt in research progression. In April, when I started with the analysis of real-time data, my laptop couldn't handle the processing of 3 million datasets. As the machines at the university were unavailable due to Covid19, Professor John Dingliana, the course director at the School of Computer Science and Statistics, helped me in providing with AWS credits. Finally, with big data in hand and the availability of trusted cloud resources allowed me to pursue this research under the guidance of Professor Khurshid Ahmad.

Summarizing the study aims to estimate the number of clusters based on the decision

latency received while computing light and heavy volatility models and data availability in a High Frequency Trading system.

1.6 Thesis Structure

This section introduces the challenges faced by traders in making real-time decisions and how big data tools can provide a solution to these challenges. It introduces the problem to the readers. It discusses the key research objectives, contributions made, and challenges faced. Also, the research progression provides a brief idea of the journey of this study from start to end. The next of the chapters are organized as below:

Chapter 2 – This chapter divides the literature into three sections, High Frequency Trading and its challenges, need for big data solutions, and risk estimation techniques. It presents the readers with the key tension areas in this research application, along with the available resolutions. It also discusses the key methods and substantial results of existing literature in the area of big data in finance.

Chapter 3 – The design of the system architecture is presented and explained in brief in this chapter, which summarizes the key methods in the area of distributed computing frameworks such as Hadoop and Spark, cloud computing resources by AWS (Amazon Web Services), and time series volatility models. Other methods involved in the architecture, such as real-time streaming, risk estimation techniques, stylized facts for financial returns, are also discussed here. It also presents Python packages used while computing these methods.

Chapter 4 – This chapter dives deep into the implementation of the methods described in Chapter 2, followed in a way as given in the architecture. Some of the main methods implemented in this section are a collection of exemplar time series from IEXCloud, both historical and real-time, big data storage, parallel computing framework Apache Spark, security configuration, and decision latency. Lastly, a screenshot of the interface where the program is running is shown.

Chapter 5 – This chapter evaluates the architecture and provides results on the relation

between decision latency with different sizes of clusters on a different scale of data inputs. It starts with proving the three stylized facts for financial returns on intraday futures data. Then results of each experiment are discussed along with the detailed analysis of the computation time, i.e., decision latency on a different scale of data and sizes of clusters.

Chapter 6 – The chapter discusses the conclusion and contributions of this study. A discussion of the limitations of the work is presented along with the potential future works.

1.7 Conclusion

This research focuses on providing computational solutions to big data problems that traders, analysts, or researchers face in making decisions in High Frequency Trading. The vast literature on HFT suggests that traders are not concerned with real-time decision making. They may be concerned about network latency, but decision latency has never been a topic of discussion. This may be due to the unavailable distributed computing architecture. To fill this gap between computing and finance, this study focuses only on decision latency. With decision latency observed in different scenarios, the number of computing clusters is estimated. Two experiments, one with descriptive statistics and another one with heavy time-dependent volatility models, are taken as methods for calculating risk. High Frequency 1-minute returns of S&P 500 Mini Futures ² are taken for this research. Results are showing that with an increase in the size of the data, the improvement in decision latency reduces. This is observed on each different size of clusters, i.e., 2, 4 and 8. More observations are presented in Chapter 5.

²https://www.investopedia.com/terms/s/sp_500_mini.asp

Chapter 2

Motivation and Literature Review

2.1 Introduction

Previous discussions in Chapter 1 were mainly focused on introducing fundamental concepts of High Frequency Trading, decision latency, volatility, and big data. It highlighted how big data had solved significant challenges in HFT in terms of scaling and latency. It also explained the need for using cloud technology solutions for distributed computing. The motive of the chapter was to summarize the need for an infrastructure that improves the existing HFT trading pipeline. The research question, objectives, along with the challenges were discussed.

This chapter provides detailed knowledge of all the concepts described in the introduction. A literature review of the critical developments and innovation in the field is discussed and critiqued. Lastly, the motivation of the research highlighting the importance and implications is presented.

This chapter is divided into three sections. The first section will provide in-depth knowledge of the evolution, background, existing computational challenges in High Frequency Trading. The second section will cover the need for big data solutions in financial analytics, along with a brief understanding of the existing financial business cloud for high frequency trading. Lastly, the third section will cover risk estimation techniques. Different interpretations of volatility are studied along with non-linear volatility models such as ARCH and GARCH.

2.2 High Frequency Trading

2.2.1 Evolution of high frequency trading

Before the era of computing, traders used carrier pigeons to arbitrage prices of the same security by relaying information ahead of their competitors. Telegraph cables then follow this strategy for exchanging information. In the 19th century Julius Reuter, the founder of Thomson Reuters, combined the use of telegraph cables and a fleet of carrier pigeons to run a state-of-the-art news delivery system. This was the innovation that revolutionized the speed at which news got delivered to the public. In 1980, the use of computers allowed traders to access data at a rate never seen before. With an investment of \$30 million from Merrill Lynch, Bloomberg designed the first computer that can perform real-time market operations quoting stock prices and can relay information. By the late 90s, the SEC gave permission to create electronic stock exchanges. This laid the foundation for a new type of trading: High Frequency trading or HFT (Kirilenko et al., 2011).

In 1983 NASDAQ presented a complete electronic trading system, which allowed the computers to start High Frequency Trading, which later developed gradually into its advanced stage. In the early 2000s, HFT accounted for less than 10% of equity orders, but this has proliferated (Goldstein, Kumar, and Graves, 2014). By the year 2001, HFT had an execution time of seconds, which kept reducing. As per NYSE, between 2005 to 2009, HFT volume increased by 164%. By 2010, the execution time had shrunk to milliseconds and later in the year, went to microseconds.

2.2.2 The role of speed in HFT

The faster the response of the event, more is the return. This is the reason event arbitrage strategies work profitably with HFT (Aldridge, 2013). The phrase ‘time is money’ neatly captures the business model of high frequency traders” (Ladley, 2020). With the advent of new technologies, the speed of trading is increasing. This increase in speed in microsecond and millisecond is generating enormous profits for the investment firms. As a result, the exchanges are competing with each other over speed, i.e., faster trade execution time or

lower latency. (Goldstein, Kumar, and Graves, 2014).

(Hasbrouck and Saar, 2013) note that “exchanges have been investing heavily in upgrading their systems to reduce the time it takes to send information to customers.” In June 2007 London stock exchange built a millisecond trading system called TradElect, which can confirm a limit order within 10 ms latency after it is placed and in parallel processing 3000 orders per second (MacDonald, 2007). As every pro has its cons, a HFT broker working in Chicago will not be able to act on quotes as fast as the one in New York because quotes will take time to reach Chicago. (Brogaard, Hendershott, et al., 2014). The limiting factor is speed. As described by (Laughlin, Aguirre, and Grundfest, 2014) and (Angel, 2014), locating the data centers or servers close to the trading exchange can benefit brokers in New York as compared to ones in Chicago. This practice is called the adoption of the ‘co-location’ strategy.

The given literature shows that the firms only focus on network latency and there has not been any mention on computational decision latency. This motivated my idea to work further on decision latency in HFT.

2.2.3 How does latency Impact Performance

(Baron et al., 2018) stated that there are two ways in which a trader can make money by being fast: short-lived information and risk management. (FOUCAULT, HOMBERT, and ROŞU, 2016) studied the benefits of low latency for trading under the roof of short-lived information. They explored the trading opportunity by considering a speculator trader trading ahead of incoming news. They discovered that fast speculator was involved in high volume transactions, and the trades were correlated with the short-lived price changes. Also, the speculator trader was responsible for adding liquidity, which shows similar behavior of a high frequency trader. (Biais, Foucault, and Moinas, 2015) showed that the traders doing HFT takes high benefit through cross-market arbitrage opportunities. The authors summarize by concluding that the evolution of computers improved the informativeness of quotes by quickly resetting the trades upon news arrivals. (CHABOUD et al., 2014) provided empirical evidence that latency in high frequency trading results in improving informational efficiency or price discovery, thereby increas-

ing market liquidity. Benefits were clear of fast traders participating in cross-market arbitrage.

Latency helps fast traders reduce risk by making them revising placed stale quotes on the grounds of quick arrival of news information (Hoffmann, 2014). Fast trading helps to remove excessive storage of quotes in the bag, thereby reducing the inventory (Aït-Sahalia and Brunetti, 2020). Reducing the inventory to downsize the risk appetite is confirmed by (Brogaard and Roshak, 2015) through empirical evidences. Operational risk is a worry, mainly due to software errors, corrupt trades, or poorly executed algorithms. These can be explained through the crashes such as Knight Capital in August 2012 and the Flash Crash of May 2010, which was a \$4 billion sell order sent without a price limit. (Baron et al., 2018).

2.2.4 High Frequency Returns and its Stylized Facts

Detailed study of this chapter is done from the book Asset Price Dynamics, Volatility, and Prediction. (Stephen J Taylor, 2007)

Asset prices and its Frequency

Empirical research of how asset prices behave requires price data ordered with time. This research is called a time series analysis. High frequency datasets include prices and time at which they are recorded, often accurate to the nearest second.

Asset prices are dynamic; they change slowly when market conditions are calm and fluctuate more when there are more news, uncertainty, and trading. Performing statistical analysis of market closing prices is difficult than performing the analysis on price change. This is due to the fact that stock prices are highly correlated; however, there is a very weak correlation between change in prices. The price changes are represented by returns.

(Stephen J Taylor, 2007) defines returns as “changes in the logarithm of prices, with appropriate adjustments for any dividend payments” , as shown in equation 2.1. Returns are used everywhere in trading stocks, futures contracts, exchange rates and stock in-

dices. (Engle, 2000) used the term ultra high frequency data for complete observations which inherently arrive at random times. However most researchers use regularly spaced sampled data with frequency of 5 minutes. The major benefits of using high frequency data is that it help us to seek how prices respond to information. With more and more observations it is easy to model volatility and hedge against investment risk.

$$r_t = \log(p_t/p_{t-1}) = \log(p_t) - \log(p_{t-1}) \quad (2.1)$$

This benefits portfolio managers, risk managers, and derivative traders. On the other side, microstructure effects such as the spread between ask and bid become more important, which makes it necessary to model intraday behavior. It is not easy to get high frequency data, and even if it gets available, the volume of it so high that analysing it using big data computational tools is itself a huge area for research. The need for big data in HFT is studied in depth later in this chapter. As per (Dacorogna et al., 2001), the number of tick-by-tick data in single data is equal to the number of daily observations for 30 years. The study of market microstructure has largely depended on the availability of high frequency data and the use of computing technology to handle big data. (Goodhar and O'Hara, 1997). This research was further empirically taken further by (Gourieroux and Jasiak, 2001).

The frequency of the observations depends upon the data availability and the research question. The current research focuses on high frequency data of 1-minute frequency. Due to the high volume of transactions traded within each minute interval, it is suitable for modelling volatility. Although high frequency prices provide more information, the spread between the bid and ask gives birth to high microstructure noise studied in detail by (Andersen, Bollerslev, Diebold, et al., 2000) and (Andersen, Bollerslev, Diebold, et al., 2003) presented their analysis on 30-minute-high frequency data while others mainly (BANDI and RUSSELL, 2008) found that 5-minute frequency is optimal for capturing the true microstructure of the market information.

Stylized Facts for Financial Return

The general properties of returns that are true for all different classes of assets are called as stylized facts. These are the fundamental building block for different exchanges. For intraday data, generally, the frequency at which observations arrive is random. Due to this, the duration at which the data arrives is not constant (W. Sun, Svetlozar Rachev, and Fabozzi, 2007). For a high frequency time series, if the duration between intervals at which observation arrive is constant, it is considered as an equally spaced time series called a homogeneous time series. On the other hand, if the time series is unequally spaced in terms of duration, it is called inhomogeneous time series (Dacorogna et al., 2001). In this research, equally spaced time series is considered for modelling volatility. The three major stylized facts are:

1. The distribution of returns is not normal.
2. There is minimal or no correlation between returns for different periods.
3. The correlation between absolute or squared returns for nearby days is positive and statistically significant.

Distributional Properties of Returns

The first important stylized fact is that *intraday returns are not normally distributed*. Instead it can be said that the returns distribution:

1. Is approximately symmetric.
2. Has fat tails.
3. Has high peak.

(Bollerslev, Chou, and Kroner, 1992) noticed that intraday data show fatter tails in the unconditional return distributions, and (Dacorogna et al., 2001) affirmed the display of heavy tails in intra-daily return data. Therefore, the assumption of returns as normally

distributed is highly questionable. Some of the other works to understand the distribution of returns are presented here.

Heavy tailedness in the intraday data was first modeled by (Marinelli, S.T. Rachev, and Roll, 2001). (Mittnik, Paolella, and S. T. Rachev, 2002) also presented some other leptokurtic distributions such as Weibull, Student's t, and hyperbolic to study the tailedness of the distribution of returns and confirmed that these distributions lack central limit theorem properties. The findings of (Marinelli, S.T. Rachev, and Roll, 2001) were confirmed by the (M. Sun, 2007). Empirical evidence to support that stock returns are not independently and identically distributed (i.i.d) is given by (WOOD, McINISH, and ORD, 1985). Their conclusion was that the distribution of returns during the first 30 minutes of the trading day is different from the distribution of returns in the remaining day before the closing time.

In (Stephen J Taylor, 2007) presented a table showing the percentages of standardized intradaily returns, $(r_t - \bar{r})/s$. The table is summarized on the basis of average frequencies for standardized daily returns as in shown in Table 2.1:

Table 2.1: Average Frequencies for standardized returns.

Range	Observed	Normal	Observed minus normal
0 to 0.25	26.6%	19.7%	6.9%
0.25 to 0.5	22.1%	18.6%	3.5%
0.5 to 1	27.6%	30.0%	-2.4%
1 to 1.5	13.4%	18.4%	-5.0%
1.5 to 2	5.5%	8.8%	-3.3%
2 to 3	3.6%	4.3%	-0.6%
3+	1.1%	0.3%	0.8%

The first three rows of the Table 2.1 explains that the number of returns observed within one standard deviation from the mean are more than the number of returns in a normal distribution, corresponding to a high peak in an empirical distribution. It can also be noticed that there are more observations found above 3 standard deviation from the mean than in the case of normal distribution. This corresponds to two fat tails in the returns

distribution. The high values of kurtosis are caused by outliers in the tails.

$$\hat{\rho}_{\tau,r} = \frac{\sum_{t=1}^{n-\tau} (r_t - \bar{r})(r_{t+\tau} - \bar{r})}{\sum_{t=1}^n (r_t - \bar{r})^2}, \quad \tau > 0 \quad (2.2)$$

The second critical stylized fact is:

Intraday returns from traded assets are almost uncorrelated, with any important dependence usually restricted to a negative correlation between consecutive returns.

There are numerous empirical evidences for the mentioned stylized facts. Some of the estimates for the first-order negative autocorrelation for foreign exchange returns are around:

1. -0.18 for a few days of one-minute returns (Goodhart and Figliuoli, [1991](#)).
2. -0.040 for one year of five-minute DM/\$ returns (Andersen and Bollerslev, [1997](#)), with -0.070, -0.082, and -0.043 for 10, 20, and 30-minute returns.
3. -0.108 for one year of five-minute yen/\$ returns (Martens, Chang, and Stephen J. Taylor, [2001](#)), with -0.093, -0.066, and -0.018 for 10, 30 and 60-minute returns.

The first lag autocorrelation for returns from an equity index future is near to zero. The first lag autocorrelation value of around:

1. 0.009 is calculated by (Andersen and Bollerslev, [1997](#)) for four-year returns sampled at 5-minute interval which increased to 0.039 when sampled at hourly interval.
2. 0.001 is noted by (Areal and Stephen J. Taylor, [2002](#)) for eight years of five-minute FTSE 100 returns.

The third important stylized fact for intraday return is

There is substantial positive dependence among intraday absolute returns, which occurs at many low lags and also among returns separated by an integer number of days.

(Andersen and Bollerslev, 1997), found out that the the first 400 lags for 5-minute intraday returns for S&P 500 futures were positively correlated with lag 1 correlation as 0.29 which reduced to 0.07 at lag 40, rising again to 0.14 at lag 80 forming a U-shaped pattern that repeats once a day.

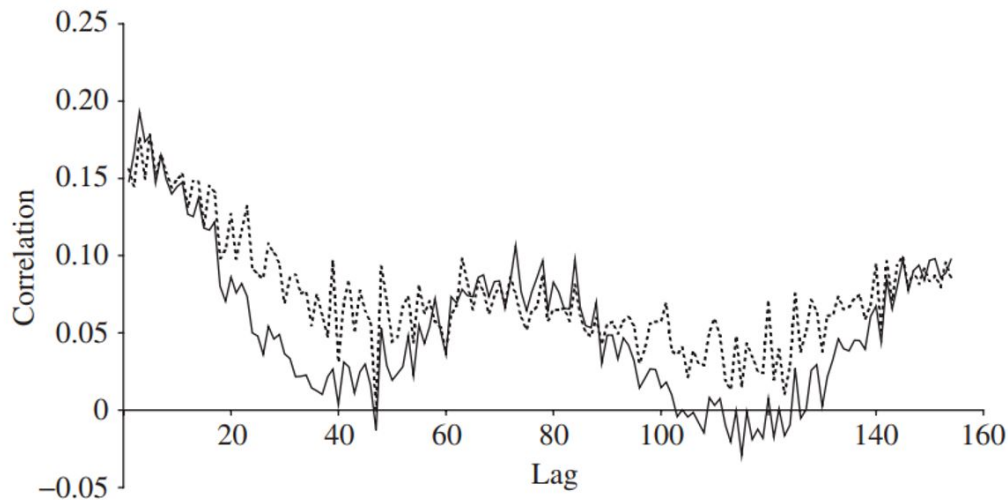


Figure 2.1: Autocorrelations for intraday absolute S&P 500 returns.

The general pattern can be seen in Figure 2.1, which shows the autocorrelations (as a solid line) for five-minute returns on the spot S&P 100 index (from July to October 1999) up to lag 154 (two days).

2.3 Need for big data Solutions

Two main big data problems that most financial institutions face while trading are latency and scalability (Tian et al., 2015). The trading firms need historical data of many years along with real-time data to predict trends in the movement of prices, estimate risks so that they can make some profit margin. It seems difficult to provide a perfect answer to the question, how to store and process a massive amount of data generating at an ultra-high frequency rate. Now with the advent of the Hadoop distributed file system (HDFS) deployed on hundreds or thousands of computers, with effective fault tolerance and data balance algorithm, it is easy to store such volume of data securely. Also, big data parallel processing framework such as Hadoop MapReduce has made systems solves

complex problems within a very less interval of time. The rapidly growing data requires the storage architecture to provide good scalability to support scaling out when the data size increases up to the storage boundary. A cloud-based big data architecture can provide a solution to both the problems.

(Khurshid, 2019) noted that Varian, one of the pioneers of the economics of information technology (Varian, Farrell, and Shapiro, 2004) and then of the economics of network technologies (Shapiro and Varian, 2013), have argued about the importance of big data tools as part of modern economic and financial analytics, mainly for two reasons: availability of huge data and model uncertainty (Shapiro and Varian, 2013) and (Varian, Farrell, and Shapiro, 2004). Normally researchers or financial analysts use a small volume of data such as monthly or daily observations as it is easy to obtain. The use of small data can lead to model uncertainty and can account for hollow predictions. Big data tools will eliminate the fear of sampling uncertainty and will help financial analysts to use years of historical data in making accurate future predictions by allowing end-users to run many linear and non-linear models concurrently on networks of high-performance machines.

(Tian et al., 2015) explained that MapReduce mostly works well for long-running batch processing jobs. However, in a real-time decision-making environment where low latency is the key prerequisite, using a distributed computing framework such as Apache MapReduce will not be a good idea as it does not support low latency. To overcome this problem, the AMPLab of Berkeley university developed Apache Spark, which is a low latency distributed data management system (Zaharia et al., 2012). It provides fast data sharing across parallel jobs by caching data that need to be reused. The programming abstraction of Spark is called resilient distributed datasets (RDD), which is a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

(Zaharia et al., 2012) also evaluated the processing power of Spark by running machine learning experiments by using two models Logistic Regression and K-Means clustering. The same experiments were performed on Hadoop. The comparison resulted in some interesting insights:

- Spark was 20 times faster than Hadoop in iterative machine learning and graph applications. The speedup comes from avoiding I/O and deserialization costs by storing data in memory as Java objects.
- Generating analytics reports in Spark was 40 times faster than in the Hadoop framework.
- Spark can provide a decision latency of up to 5-7 seconds in querying a 1TB dataset.

(Bluhm, 2018) provided a detailed analysis of setting up Spark both on a standalone machine and on cloud-based EMR clusters. More specifically, he explained how to use Spark to (i) explore big data sets that exceed retail grade computers memory size and (ii) run typical econometric tasks, including time-series regression models, which are prohibitively expensive to evaluate on standalone machines. (Bluhm, 2018) concluded that using an Elastic Map Reduce (EMR) setup, he was able to pre-process a 150 GB dataset in just under five minutes, whereas the standalone approach on our local machine crashes. Most importantly, his research concluded that for the time series analysis case, the distribution scheme reduces total runtime performance by about 95% relative to a single machine. There are many other works of literature on comparison of Hadoop MapReduce and Apache Spark with implementation on different application, both of them (Verma, Mansuri, and Jain, 2016) and (Chebbi et al., 2018) proves that Spark is way faster in terms of decision latency than MapReduce.

(Agopyan, Şener, and Beklen, 2010) provides a business cloud architecture for high frequency trading. This architecture (See Figure 2.2) is studied and used as a base reference in this research. CM is directly connected to electronic execution platforms and data providers. All routing, data and protocol transformations, mediations, and messaging between modules and CM are done via ESB.

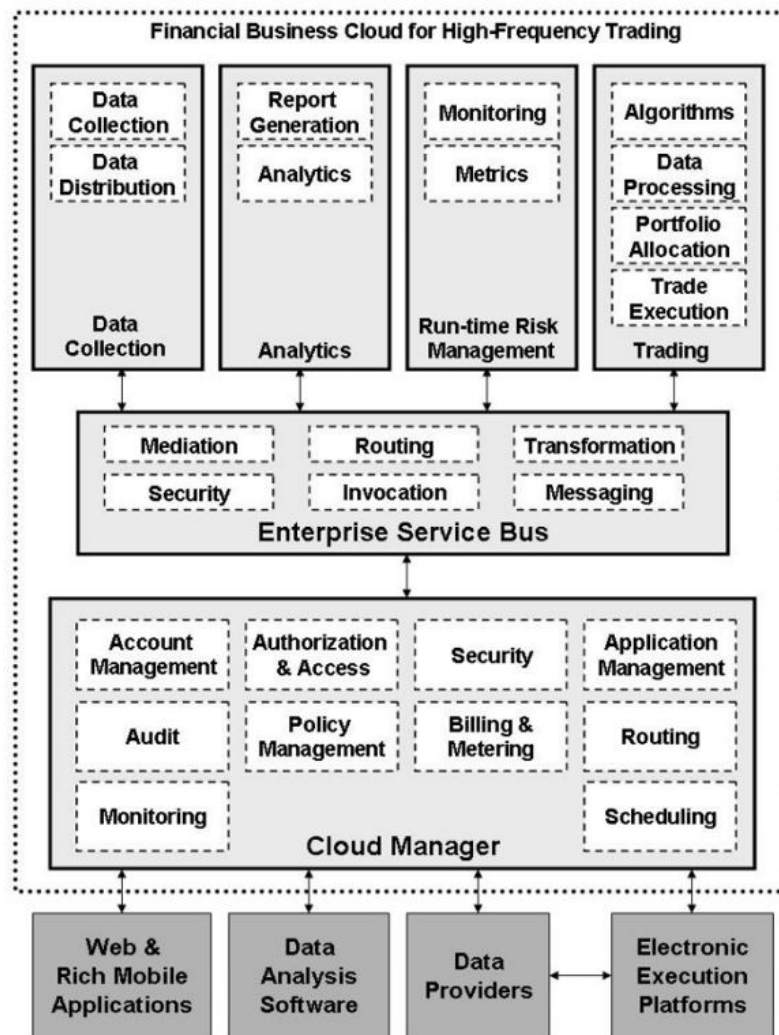


Figure 2.2: Reference component architecture of Financial Business Cloud. Cloud Manager is the only source connected to the stock exchange and data providers. ESB (Enterprise Service Bus) is like a cluster manager in the Hadoop framework responsible for managing resources, distributing tasks, etc. It provides intelligent routing to perform tasks such as real-time analytics, run time risk management, or trade execution.

2.4 Risk Estimation

Volatility is considered as the quantified measure of market risk. Risk, in financial terms, is defined as the chance that actual investment returns do not meet the expected investment return. It can be defined as the loss; a person can incur on his/her investment. Risk is normally quantified by studying assets, historical behavior, and outcomes. Standard deviation is generally used as a common metric associated with risk. Volatility is considered a measure of risk estimated by calculating the standard deviation of asset returns. The unique idea of using high frequency data goes back to 1980 when (Merton, 1980) noted that the variance of returns could be a more precise measurement in the computation of volatility than the expected returns. Volatility is exactly not the same as risk, although the two are quite related. Risk is a kind of negative volatility, or one can say that is the negative return on investment. On the other hand, volatility can be both positive and negative, as it measures the spread of the outcomes. (Ladokhin, 2009).

Volatility measures the asset price varying over some period of time. In other terms, volatility is the standard deviation of the change in the logarithm of prices over a certain period of time (Stephen J Taylor, 2007).

2.4.1 Different Interpretations to Volatility

As described in (Stephen J Taylor, 2007), there are five ways in which volatility can be phrased or interpreted. Out of those two significant methods used in this research are defined here:

I *The volatility of tomorrow's price is 1%, given our observations of recent prices.*

Realized Volatility (Historical Volatility) - It is the standard deviation of a set of previous historical returns. Realized volatility is the assessment of variation in returns for an investment product by analysing its historical returns within a defined time period. If σ is the daily standard deviation or daily volatility, then the annual volatility can be measured by $\sigma\sqrt{N}$.

N - Number of trading days (If annualized then N = 252 days in a trading year).

Annual Realized Volatility = $\sigma\sqrt{N}$

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (r_i - m)^2}{n-1}} \quad (2.3)$$

n = Number of observations

m = Mean

σ = Standard Deviation

r_i = Returns

Volatility can be more precisely calculated as the frequency of returns increases i.e. in case of intraday returns where returns are uncorrelated. We suppose the periods are trading days with daily returns r_t that is the sum of N intraday returns $r_{t,j,N}$

$$r_t = \sum_{j=1}^n r_{t,j,N} \quad (2.4)$$

For $N = 1, 2, 3, \dots$ we define the realized variance for day t as (Andersen et al., 2001)

$$\hat{\sigma}_{t,N}^2 = \sum_{j=1}^n r_{t,j,N}^2 \quad (2.5)$$

II *The volatility of tomorrow's price is 1%, given our observations of recent prices.*

Conditional Volatility – It is the standard deviation of a future return that is conditional on known information. Previous or known information can be anything such as history of previous returns.

Unlike realized volatility, the expectation for the next period is calculated using a time-series model. Convenient and accurate equations for volatility expectations are provided by ARCH (Auto regressive conditional heteroskedastic) models. In statistics, when the variance of a term is affected by the variance of one or more other variables, it is "conditional." Variance is not just uniform, but is affected by variances preceding it.

As per Oxford dictionary (*Oxford Dictionary 1928*):

Heteroskedasticity – “Statistics of unequal scatter or variation; having different variances.”

Homoskedasticity – “Statistics of equal scatter or variation; having equal variances” Equation:

Third statement can be made if the parameters (μ, α, β , and ω) from time series ARCH model are estimated using daily returns suppose until 23 November 2005 (day t-1) and had then found $h_t = (0.01)^2$ for the return from the 23rd to the 24th (day t).

2.4.2 Time Series Volatility Modelling

Considering historical data, which is often the subject for unconditional volatility. Historical volatility is a process of measuring the dispersion in the price of the underlying assets; however, since that measurement is just historical and the volatility is varying over time, it might not be the right way of measure future volatility. To capture the dynamic properties of returns sampled at high frequency intervals, it is necessary to use linear or non-linear models to estimate the changing mean and standard deviation. The time it takes to calculate a volatility measure adds to the latency or delay in producing the measure.

The ARCH or **Autoregressive Conditional Heteroskedasticity** method introduced by (Engle, 1982) provides a way to explicitly model the change in variance or change in past squared observations in a time series, such as increasing or decreasing volatility. These models use variance instead of standard deviation as a measure of volatility. However, there is a direct relation between the two. An extension of ARCH model named GARCH or **Generalized Autoregressive Conditional Heteroskedasticity** incorporates a moving average component together with the autoregressive component. The introduction of a moving average component allows to model both the conditional change in variance over time as well as changes in the time-dependent variance. The detailed explanation of the models is explained in the section 3.6.3 of Chapter 3.

Volatility Forecast Evaluation

The paper by (Andersen and Bollerslev, 1998) was an answer to an evaluation of GARCH models. Numerous papers had noted that GARCH models were incapable of describing much of the changeability in squared returns when evaluated on out-of-sample data, despite the fact that GARCH models had a good in-sample fit, see, e.g. (JORION, 1995) and (Figlewski, 1997). The incorrect implication that resulted from this observation was that GARCH models were of little practical importance. By using the realized variance, which is a more accurate measure of volatility than squared returns, (Andersen and Bollerslev, 1998) showed that standard volatility models perform rather well. So, the apparent ‘poor’ performance can be attributed to the fact that the squared return is a very noisy proxy of the conditional variance.

2.5 Motivation

Current quantitative analysis involves the use of single model small datasets which pose sampling uncertainties, which can eventually lead to biased results (Hoffmann, 2014). It is crucial to embrace more massive datasets, thereby reducing sampling uncertainty. One proposal is that we should use a large number of explanatory models to deal with modelling uncertainty. The two uncertainties are exemplar fundamental research problems related to the scalability and responsiveness of analytics capabilities of systems dealing with complex markets – especially about data mining of large High Frequency structured data sets – time series of prices and traded volumes in complex markets.

It is difficult to see literature on the concerns of decision latency in real-time high frequency trading environment. High frequency traders are not focused on taking real-time decision which incorporates dependency on large historical data to estimate future risk through the computation of heavy volatility models such as GARCH. This research is to fill the gap in the field of finance and computing through the use of big data tools.

Chapter 3

Design and Methods

3.1 Introduction

In the coming chapters, the design of the architecture is discussed briefly. Each stage in the design adds some latency to the model. Adaption of tools such as AWS Kinesis for real-time streaming, AWS S3 bucket for Big Data storage, and Apache Spark for distributed computing are discussed with their benefits in terms of latency and scalability. Significant differences between Hadoop MapReduce and Apache Spark, local storage, and cloud-based storage solutions are also discussed. Important risk estimation volatility models and stylized facts for financial returns are also discussed in the upcoming chapters. It is worth noting that Spark can also be configured on a standalone machine; however, as this research is focused on reducing decision latency, hardware latency cannot be tolerated.

3.2 System Architecture

Two forms of data are collected, one of which is historical data, and the other one is streaming data. Historical data is first collected by calling a web API provided by data source provider IEXCloud. Historical data is stored in a secured AWS S3 bucket. Real-time streaming data from the IEX cloud is collected put into the shards by Kineses

Producers. On the other side, data is collected by Kinesis Consumer from the shards. Before the real-time data is distributed over EMR clusters sitting on the Apache Spark framework, it gets merged with the historical data stored in the S3 bucket. After data is aggregated, risk estimation techniques are used to calculate future volatility. Based on the estimated volatility, a decision is taken to buy, sell, or hold security, and most importantly, execution time is noted. This whole process is followed only to observe decision latency of the machines in the end. Whole processing and visualization are performed on Apache Zeppelin notebook. Output data is stored again in the S3 bucket and used for next minute processing. In the following chapters, each tool and technology used are discussed with the methods implemented in estimating the risk of high frequency financial returns.

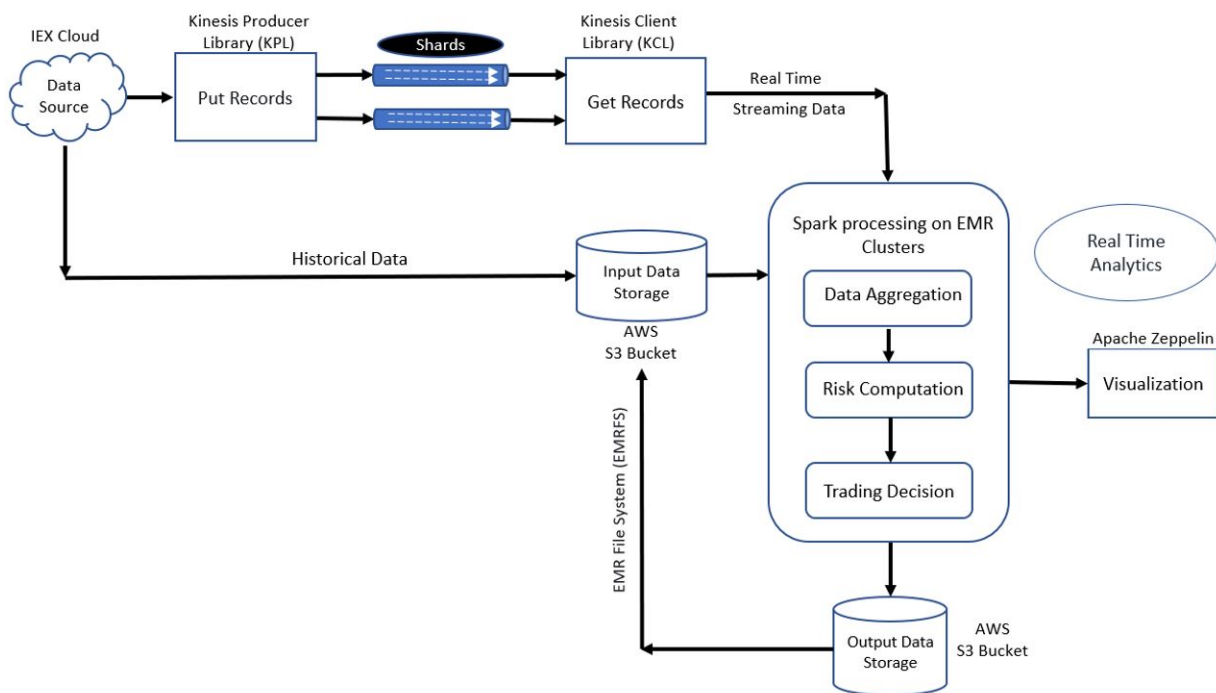


Figure 3.1: System Architecture Overview. Three major steps involved are data collection and aggregation, risk estimation using unconditional and conditional volatility models, analysis of decision latency.

3.3 Parallel Computing

In most simple language, the use of more than one computer resource together to solve a computational problem is called parallel computing. In different ways, parallel computing can be explained in simple steps:

- Breaking the problem in simple independent problems that can be solved concurrently.
- Each independent problem is further broken in small series of instructions.
- Instructions from each part execute simultaneously on different processors.
- An overall control/coordination mechanism is employed.

Parallel computing breaks a problem into small independent problems and takes the help of the cluster manager to distribute these problems in the form of tasks to different clusters or computing machines. The solution to each problem is combined at the end of the overall algorithm. The objective of parallel computing is to increase computation power for faster application processing and problem-solving. Until a few years ago, there was a possibility to process a huge volume of data using Vertical Scaling¹ techniques. But now, as the volume of data managed by our conventional storage systems has surpassed its processing ability, companies have started adopting cloud-based Horizontal Scaling techniques to resolve the need for scalability.

Storing and processing Big Data is a daunting task, and there are well-developed frameworks for such tasks. Apache Hadoop is one such framework.

3.3.1 Apache Hadoop

Apache Hadoop is an open-source big-data framework providing a platform for handling large data sets through distributed storage and processing. It is an open-source implementation of the MapReduce programming model developed by Google to process

¹<https://www.geeksforgeeks.org/horizontal-and-vertical-scaling-in-databases/>

and analyze massive datasets. The Hadoop ecosystem has two core components HDFS (Hadoop Distributed File System) and MapReduce. HDFS handles the data storage between all the machines on which clusters are running. MapReduce, on the other hand, handles the processing of the task by breaking it into Map and Reduce actions.

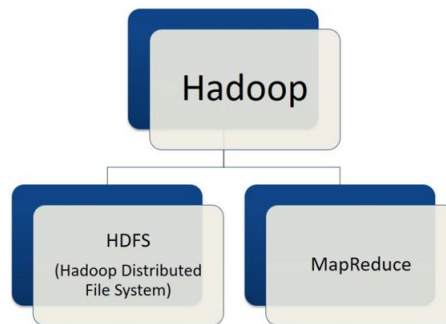


Figure 3.2: Hadoop Core Components

Hadoop MapReduce

MapReduce is a programming model in which each task is divided into map and reduce functions. It provides a solution by first distributing data over clusters and processing tasks on each cluster parallelly in an efficient time interval. MapReduce model splits the data instead of splitting the processing tasks. Multiples nodes are responsible for processing a small dataset each rather than processing all the data on one single node.

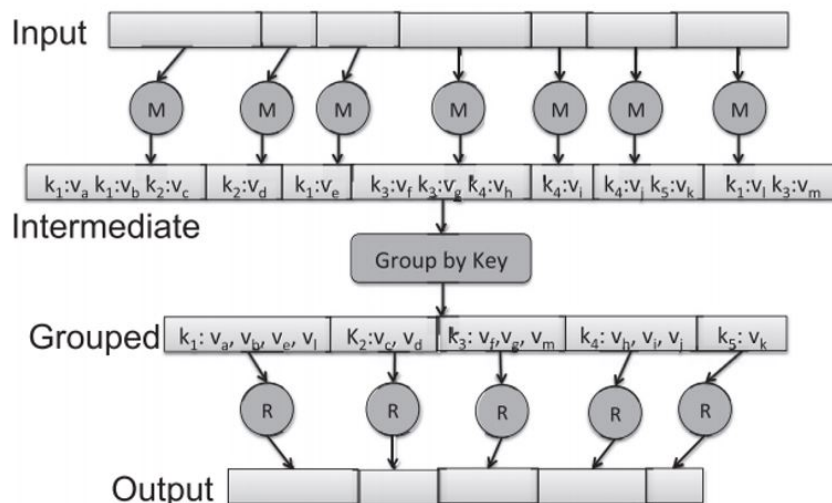


Figure 3.3: The MapReduce programming model. K elements represent the keys in pairs.

As shown in Figure 3.3, MapReduce can be described with three basic rules:

- The map function reads the data and transforms it into a key, value pair called a tuple. Any sequence of operations can be applied as a transformation to the tuples before they are grouped by key.
- Tuples with the same key are grouped, and the data is sent over the network to the reduce function.
- Finally, the reducer function performs operation or sequence of operations on multiple values for each key and results in one value for each pair. This action reduces the total amount of data sent across the network.

HDFS (Hadoop Distributed File System)

HDFS is the main module of the Hadoop framework. It is a fault-tolerant storage system. It is also a scalable distributed file system used to store the volume of data that cannot be stored in a single machine. HDFS stores all files in blocks. The default block size is 64Mb. All files on the HDFS have many copies, which help in parallel processing. HDFS clusters have two types of nodes, name nodes, and data nodes.

- Name node: It is a master node that handles the directory of tasks distributed over name nodes for processing. It tracks on which cluster data file is stored; it also informs cluster manager if a data file gets corrupted in any of the data nodes. It manages the metadata of the whole file system. It does not store actual data.
- Data nodes: They are slave nodes that store actual data as blocks. It sends all the information of file it contains, the health status of the file system, and responds to the actions requested by name node.

3.3.2 Apache Spark

Just like MapReduce, Apache Spark is a distributed cluster computing framework used to store and process large volumes of data at lightning speed due to its in memory-intensive scheme. Unlike Hadoop MapReduce, which uses disk-based operators, Spark

uses memory-based operators to handle the processing of the tasks, which makes it fast and prone to multiple input/output (I/O) operations.

Spark Core is the base of the platform. It handles many important tasks such as managing the memory of the system by interacting with system storage, helping in the recovery of the systems during failure, distributing and monitor tasks on clusters, etc. Spark core uses Python, Java, or Scala-based application programming interfaces (API) to hide complex operations of parallel computing behind simple high-level operators. Spark is based on a master/worker architecture where the Spark driver communicates with the YARN cluster resource manager as a single coordinator, which is responsible for managing the Spark workers in which executors run.

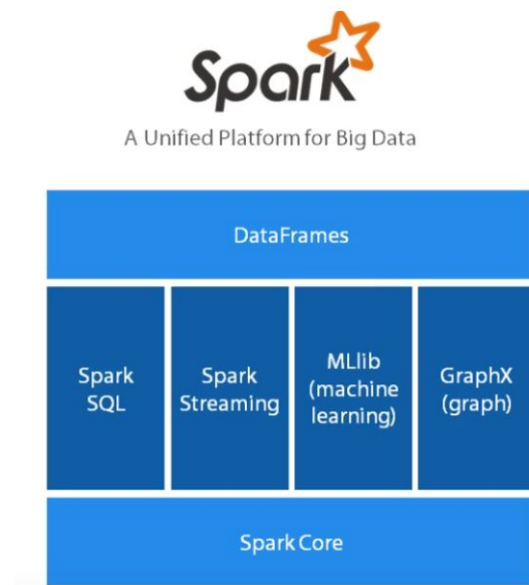


Figure 3.4: Spark Framework - Spark Core as the foundation for the platform, Spark SQL for interactive queries, Spark streaming for real-time analytics, Spark MLlib for machine learning and GarphX for graph processing.

Apache Spark Architecture

Spark framework is built on three main components:

Driver – A spark context object drives the process by converting user code into parallel tasks distributed across worker nodes.

Executors – Spark uses executors to process tasks on each worker node. It is also tasked with storing data on the application. A manager is required to connect drivers and ex-

ecutors where the cluster manager comes into existence.

Cluster Manager – It works as a mediator between a driver and multiple executors. Spark context can connect to the cluster manager to allocate resources or tasks across the worker nodes. There are several types of cluster managers that deal with on-demand task allocation. These are Hadoop YARN, Spark Standalone, Mesos.

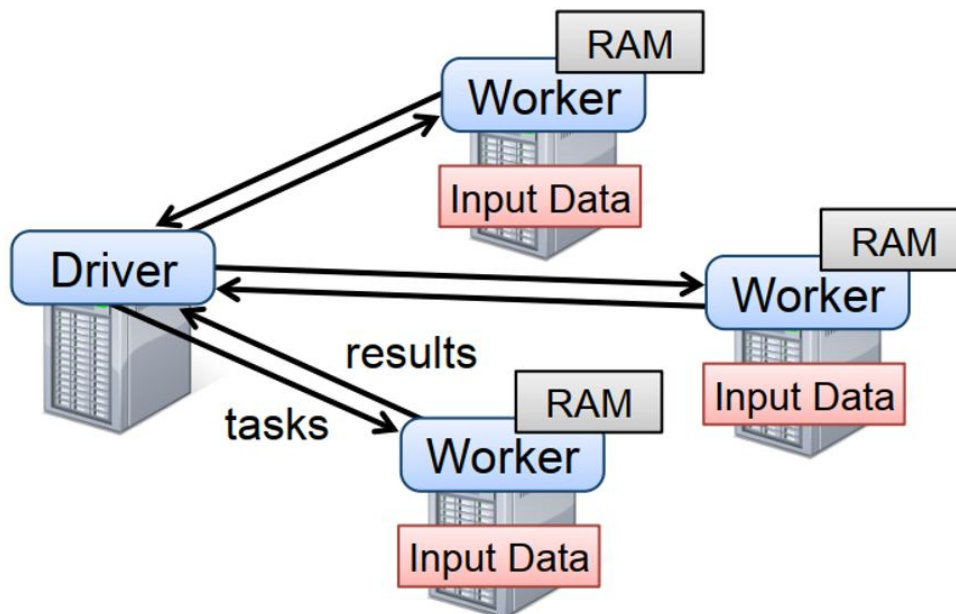


Figure 3.5: Spark runtime. The user’s driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

Spark RDD (Resilient Distributed Datasets)

RDD the fundamental data structure of Spark and is considered as the heart of its framework. It is an immutable distributed assembly of objects. The term resilient means that an RDD can be transformed into a new RDD without altering the original. The distributed means RDD divides the dataset into multiple logical partitions and distributes them over nodes for processing. One way to create an RDD is to parallelize data objects stored in external file systems such as Hadoop Distributed File System (HDFS) or S3 or any other file system. Another way to parallelize the already existing collection of objects in the driver program.

DAG (Directed Acyclic Graph)

Apache Spark builds the user's data processing commands into a Directed Acyclic Graph, or DAG. The DAG is Apache Spark's scheduling layer; it determines what tasks are executed on what nodes and in what sequence. The vertexes in a DAG are like parallel tasks distributed across a tree, and edges are used to exchange information between a driver and a worker. As shown in Figure 3.6, vertexes can have multiple connections between inputs and outputs, which imply that the same task can be run in different data and the same data in other partitions.

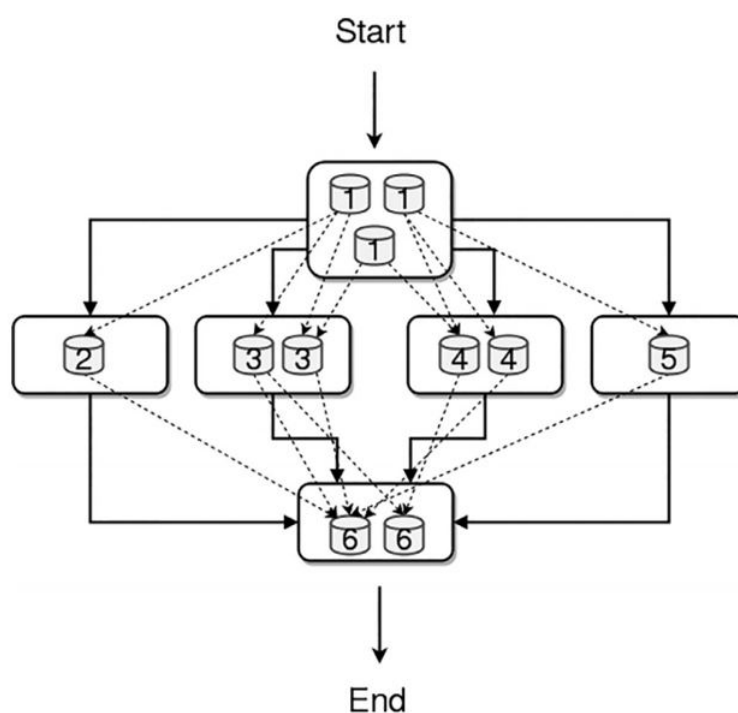


Figure 3.6: Direct Acyclic Graph Architecture. Each square is a task, inside which nodes are present for processing information. Arrows connecting nodes represent the data flow between nodes and the vertexes in the graph, dashed lines represent the dependencies between data blocks (cylinders) .

3.3.3 Limitations of Hadoop MapReduce

There are two significant advantages due to which Spark is the preferred framework over MapReduce paradigm for storing and processing Big Data. Two of the significant benefits of using Spark over MapReduce are speed and inbuilt Spark components such as

Spark SQL, Spark Streaming, MLlib (machine learning), and GraphX (graph processing). Other major differences are given in Table 3.1.

Table 3.1: Difference between Hadoop MapReduce and Spark.

Difference	Hadoop	Spark
Category	Basic Data processing engine	Data analytics engine
Usage	Batch processing with a massive volume of data	Process real-time data, from real-time events like Twitter, Facebook
Latency	High latency computing	Low latency computing
Data	Process data in batch mode	Can process interactively
Ease of Use	Hadoop's MapReduce model is complex, need to handle low-level APIs	Easier to use, abstraction enables a user to process data using high-level operators
Scheduler	The external job scheduler is required	In-memory computation, no external scheduler required
Security	Highly secure	Less secure as compare to Hadoop
Cost	Less costly since MapReduce model provide a cheaper strategy	More expensive than Hadoop since it has an in-memory solution

3.3.4 Hadoop On-Premise vs. Hadoop on Cloud

There are two options for running Hadoop clusters, one is on local machines, and the other is on the cloud platform. This chapter aims to discuss the limitations of storing and processing Big Data on clusters running on local hardware machines, and why is it important to adopt cloud-based solutions. On-premise systems face both challenges of storage and processing. Its physical nature adds latency to the processing. As the demand for data increases, more and more physical servers need to be installed to handle

big data storage and processing challenges on local machines, making the whole process inefficient, time-consuming, and costly. On the other side, the cloud platform offers full scalability; with on-demand services, one can increase storage capacity, the number of clusters for processing, or the number of cores within a cluster without spending too much money. Any number of virtual servers can be spun up in the cloud within a few minutes as per requests. The great advantage of using the cloud is one can scale the size of clusters while running parallel jobs simultaneously. A few of the other benefits of using the cloud are presented below:

1. Security

Hadoop on-premise infrastructure can be more secured than on the cloud. All the sensitive data can sit behind the firewalls on local machines. However, for the cloud, security is all dependent on the cloud service provider. The major leader in providing cloud infrastructure as a service is AWS (Amazon Web Services). Amazon Web Services does offer security features such as virtual private clouds, encryption, security groups, and more, so Hadoop on the cloud can be secure if implemented correctly.

2. Performance

Apache Hadoop runs slowly on virtual environments due to intensive I/O operations is a common assumption. However, cloud-based setups have better performance running Hadoop with real-world applications.

3. Availability

Data can be accessed and used for processing at any location in the world on the cloud. A cloud service provider such as AWS has data centers all over the world. This is one of the main benefits of using cloud infrastructure for Big Data storage and processing.

4. Scalability

With data increasing day by day, it is not possible to store such a massive volume on the on-premise infrastructure. Scalability is the flexibility of a company to handle data increasing capabilities in between of an ongoing process. Big Data in the cloud allows us to scale the CPU cores up or down with incredible ease and without negative financial implications.

3.4 AWS Cloud Solutions

AWS provides high performing easy to use services for storing and processing massive volumes of data. This research makes use of AWS Kinesis for real-time streaming, AWS S3 bucket for storing Big Data, and EMR (Elastic MapReduce) clusters for parallel processing of Big Data through the Apache Spark framework. This chapter provides a detailed understanding of all three services used with their architectures.

3.4.1 Real-Time Streaming

Real-time data streaming is a continuous flow of data generated from various sources waiting to be processed and utilized for decision making. The streams of data are generated from all different kinds of sources in a structured or unstructured format. The sources can range from various web sources such as server log files, IoT (Internet of Things) devices, networking devices, website activity, banking transactions, and location data, etc. The data from all these sources can be aggregated in real-time and be mined and analyzed for business improvement decision-making process. Similarly, high frequency data in the stock market is generated every time there is a change in price. It can be in milliseconds or seconds. Real-time high frequency data can be streamed using various streaming platforms such as Apache Flink, Apache Kafka, AWS Kinesis, etc. In the architecture used in this research, AWS Kinesis is used.

AWS Kinesis

AWS Kinesis has four different capabilities. These four are Kinesis Data Streams, Kinesis Data Firehose, Kinesis Video Streams, and Kinesis Data Analytics. This research focuses on Kinesis Data Streams. Kinesis data streams enable you to process and analyze data as it arrives and responds instantly instead of having to wait until all your data is collected before the processing can begin. Amazon Kinesis data stream uses shards as its base throughput unit. The capacity of 1 shard is 1 mb/sec as input and 2 mb/sec as output rate. Also, in terms of records, in one second, a single shard can support 1000 PUT records.

Kinesis Streams Overview

- Streams are divided into shards/partitions.

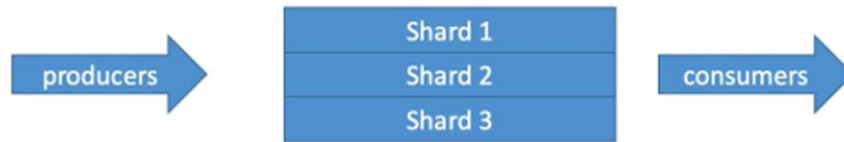


Figure 3.7: Data distribution in shards.

- Data can be recovered if lost within 24 hours and can be extended to 7 days if configured externally.
- Kinesis can reprocess or replay data.
- Multiple applications can consume the same streams.
- Provides real-time processing with a scale of throughput.
- Once data is inserted into streams, it can't be deleted (immutability).

Kinesis Streams Shards

- One stream is made up of many different shards.
- Billing is per shard provisioned on-demand basis.
- Batching available or per-message calls.
- The scaling of shards can be done in parallel.
- Records are ordered per shard.

Kinesis Streams Records

- Data Blob is the data being sent, serialized as bytes. The maximum data which can be sent is 1 MB/sec.

- A record key is sent alongside a record, which helps to group records in shards. Same key = same shard.
- The sequence number is the unique identifier for each record put in shards. It is added by Kinesis after ingestion.

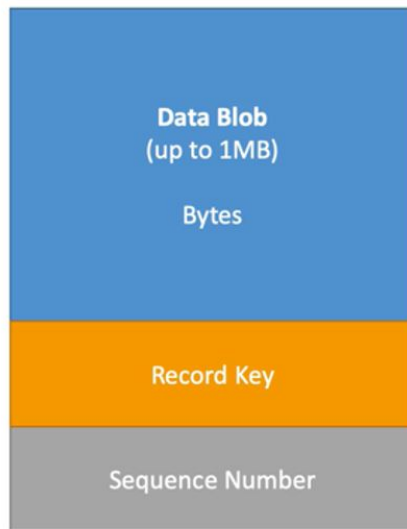


Figure 3.8 Kinesis Stream Records Components

Figure 3.8: Kinesis Stream Records Components.

Overview of Kinesis Producer Library (KPL)

The high-level architecture on Kinesis Data Streams:

- The producers put records (data ingestion) into shards. AWS Kinesis Producer Library (KPL) is a simple Java program written to send data from source to kinesis data streams with high throughput.
- Kinesis data stream a set of shards. Each shard has an arrangement of data records. Data records are a collection of a partition key, sequence number, and a data blob (up to 1 MB), which is an immutable sequence of bytes.
- The consumers get records from Kinesis Data Streams and process them. You can build your applications using either Kinesis Data Analytics, Kinesis API, or Kinesis Client Library (KCL). The KCL defaults follow the best practice of polling every 1 second. This default results in average propagation delays that are typically below 1 second.

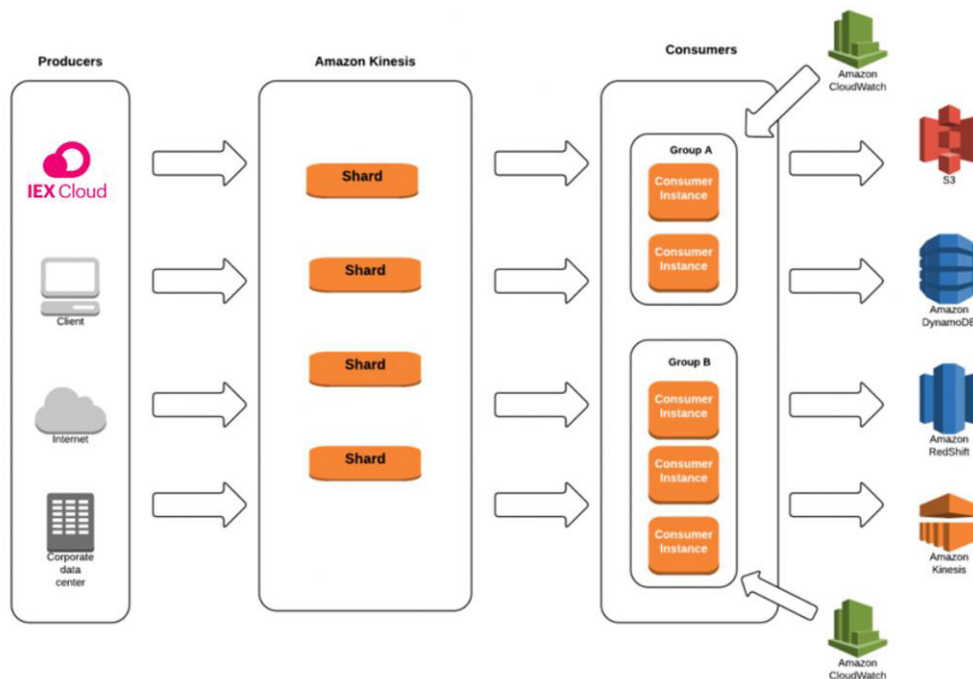


Figure 3.9: Kinesis Architecture

3.4.2 AWS EMR (Elastic MapReduce) Big Data Platform

Amazon EMR provides an easy to manage cluster platform on which a large volume of data can be easily processed using different parallel computing frameworks such as Apache Spark or Hadoop MapReduce. It helps Spark or Hadoop to quickly distribute large data stored in AWS data stores such as S3 bucket or database such as AWS DynamoDB over clusters for parallel processing.

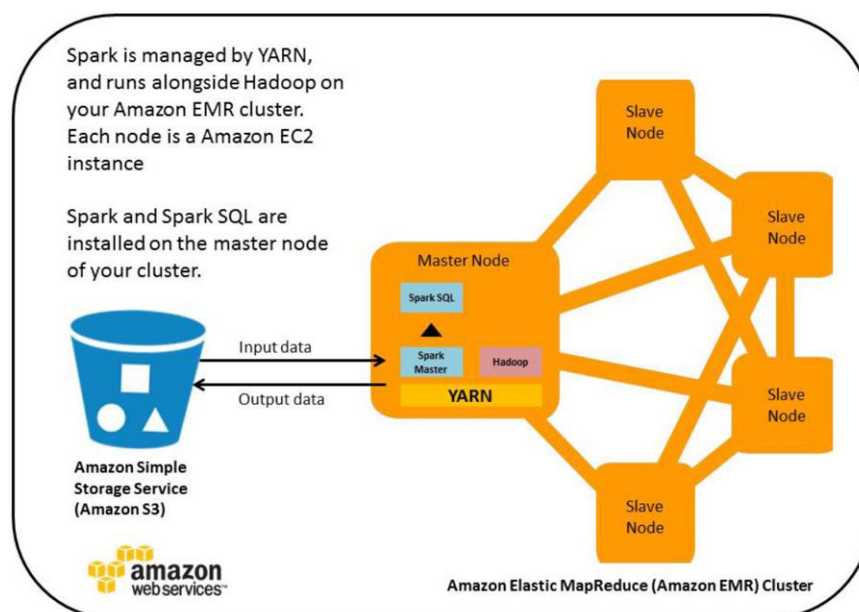


Figure 3.10: AWS EMR (Elastic MapReduce) Architecture

AWS EMR architecture has many layers, each of which provides certain functionalities. Below are some tools used for each functionality used in this research:

Storage - AWS S3 Bucket

Amazon Simple Storage Service or (S3) bucket is considered as the storage for the Internet. It is used by users all over the world to store data in virtual servers that can be accessed anytime from anywhere in the world on the web. S3 allows people to store objects (files) in buckets (directories). Buckets must have a globally unique name. These buckets are defined at a regional level. It gives developers access to scalable, fast, low-cost, and reliable data storage infrastructure that AWS uses to run its own global network of web sites. Object or files have a key to access the path where data is stored. For Big Data storage on cluster computing platform, there are many storage options available such as HDFS (Hadoop Distributed File System), EMRFS (Elastic MapReduce File System), or Local File System. This research deals with the EMRFS storage solution.

EMR File System (EMRFS)

AWS EMR allows Spark or Hadoop to access data directly from the S3 bucket considering it as a distributed file system such as HDFS. Therefore one can use S3 as a file system to store and process data over clusters. Mostly S3 is used to store input/output data, and intermediate results are stored in HDFS.

Cluster Resource Management

By default, AWS EMR uses YARN (Yet Another Resource Negotiator) is a resource manager to manage cluster resources for multiple data-processing frameworks. The function of cluster managers is to decide which task has to be performed by which cluster. Yarn uses the node manager to do task allocation. The other duties of the resource managers are to monitor cluster memory and performance and handle task failure. The resource manager has two components, scheduler and application manager. The scheduler allocates resources to the various running applications and performs scheduling based on the

resource requirements of the applications. The resource manager accepts job submissions, and each job is allocated to the application manager.

AWS Clusters

AWS EMR (Elastic MapReduce) clusters are nothing but a collection of AWS EC2 Ubuntu virtual machines running in parallel. At the back, these EC2 (Elastic Compute) instances support multithreading, which allows the threads to run concurrently on a single CPU core. One can scale the number of cores up to a limit within a single instance also. Each thread represents one virtual CPU (vCPU) on an instance. An example of instance is a m5.xlarge, which comes with 2 CPU cores and 2 threads per core by default. This makes a total of 4 vCPUs in total in a single instance.

As per business needs, one can either use default EC2 instances with given memory, and a number of vCPUs or can change the number of CPU cores, threads per core as and when required.

- **Number of CPU cores:** One can modify the count of CPU cores for an instance. Reducing the number of cores is also an option if one needs more RAM for memory-intensive workloads but fewer CPU cores.
- **Threads per core:** Multithreading can also be disabled by enabling a single thread option for each CPU core.

Also, one can specify these CPU selections during the start of the instance as there is no extra charge for specifying CPU options in advance. At the time of processing, if one needs to improve the capacity of CPU cores, they can select the option from the application and will be charged on-demand basis.

Data Processing Framework

The engine of the whole Spark framework is the data processing framework layer used for processing and analysing the input data. The most common resource management

package used in the Hadoop framework is YARN (Yet Another Resource Negotiator). Spark also has the ability to use YARN. The two primary data processing frameworks are Hadoop MapReduce and Apache Spark. YARN allows data stored in HDFS to be processed in various forms, such as batch processing, stream processing, or graph processing. Both of them are already discussed, along with their limitations. However, this section mainly describes the service level architecture of AWS EMR with Spark.

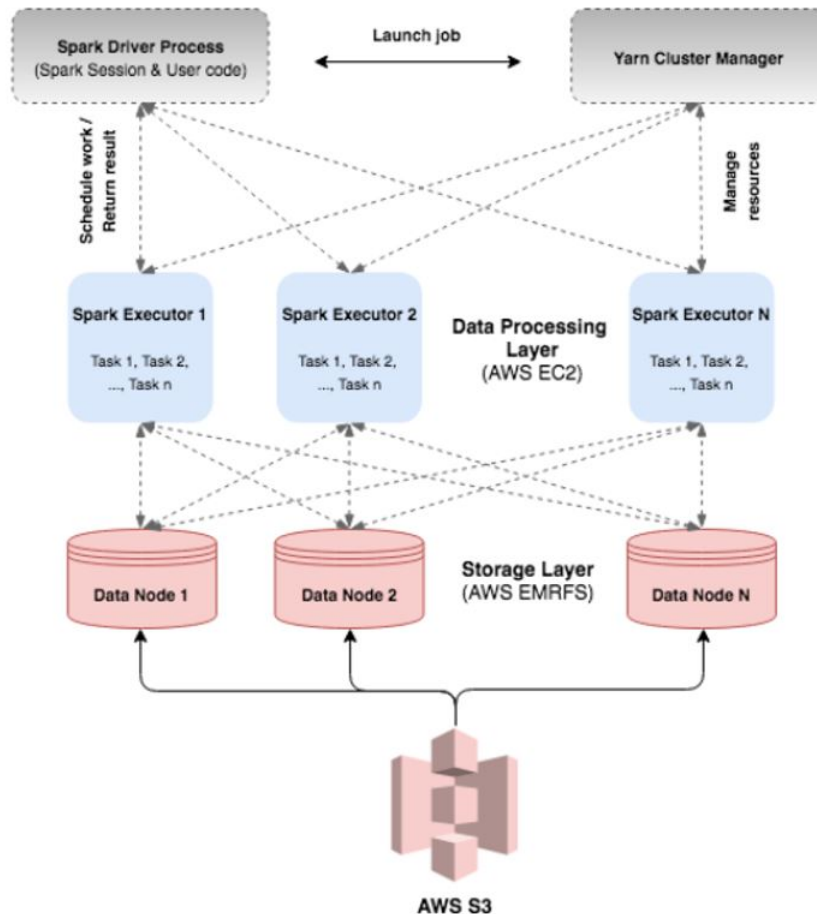


Figure 3.11: Apache SPARK Framework used for processing Big Data distributed over AWS EMR clusters. On launching a spark application, the Spark Driver breaks the program into multiple tasks and distribute them over spark executors. The Spark Driver has to request cluster manager to start the jobs over executors. The Spark Executors perform the tasks received from the Spark Driver. The Distributed Storage Layer is based on the Hadoop API and holds the distributed dataset, which is partitioned across hard drives of the Spark worker nodes

A simple diagram of the Amazon EMR service architecture is illustrated in figure 3.11. There are four layers, providing different capabilities and functionalities to the cluster. The storage layer uses the EMR File System (EMRFS), which contains Amazon S3 as a distributed, scalable file system, where input and output data is stored. In the Spark application of this research, time-series data and output from model fitting and forecasting will be stored in Amazon S3. The resource management layer uses YARN (Yet Another Resource Negotiator) and Apache Spark as a data processing framework.

Finally, the EMR cluster contains a layer for applications and programs that interact with Spark. For the experiments presented in this research, the Python library of Spark, i.e., PySpark, is used.

3.5 High Frequency Methods

High Frequency Returns and its Stylized Facts

Different methods for estimating volatility are discussed in this chapter. But before moving to time series volatility modes, the chapter also throws light on understanding the stylized facts for financial returns. Stylized facts are necessary to understand the statistical properties of intraday returns. These properties are studied to understand return distribution, autocorrelation, unconditional means, and variances. A detailed study of the methods in the upcoming sections is done from the book *Asset Price Dynamics, Volatility, and Prediction* (Stephen J Taylor, 2007).

Returns vs. Prices

Statistical analysis of market prices is more difficult than analysis of changes in prices. The reason for this is market prices are highly correlated. However, change in the logarithm of prices produces little or no autocorrelation. Logarithm returns are calculated by the below-given formula:

$$r_t = \log(p_t/p_{t-1}) = \log(p_t) - \log(p_{t-1})$$

Autocorrelation of returns

One of the earliest studies done on intraday returns by (WOOD, McINISH, and ORD, 1985) shows that the intraday returns are not stationary and follow a low order autoregressive process. Autocorrelation is considered as a measure of the dependence of the returns at period t with previous returns at period $t + \tau$ separated by interval τ . Autocorrelation is the same as calculating the correlation between two-time series, except autocorrelation uses the same time series in two forms, one in the original form and one in the lagged form.

The correlation between returns τ periods apart is estimated from observations by the sample autocorrelation at lag τ , with \bar{r} the sample mean of all n observations

$$\hat{\rho}_{\tau,r} = \frac{\sum_{t=1}^{n-\tau} (r_t - \bar{r})(r_{t+\tau} - \bar{r})}{\sum_{t=1}^n (r_t - \bar{r})^2}, \quad \tau > 0$$

The symbol $\hat{\rho}$ indicates that the sample statistic estimates a correlation parameter ρ of a stochastic process when the data come from a stationary process. The two subscripts τ and r respectively state the lag and the series that provide the estimates. The range of autocorrelation is from -1 to +1, where a correlation of positive 1 represents perfect positive autocorrelation, and a correlation of negative 1 represents perfect negative autocorrelation. Normally technical analysts use autocorrelation to find out how past prices affect future stock prices.

Stylized Facts for Financial Returns

The general properties of returns that are true for all different classes of assets are called as stylized facts. These are the fundamental building block for different exchanges. The three major stylized facts are:

1. The distribution of returns is not normal. Instead, distribution is asymmetrical with fat tails and a high peak.

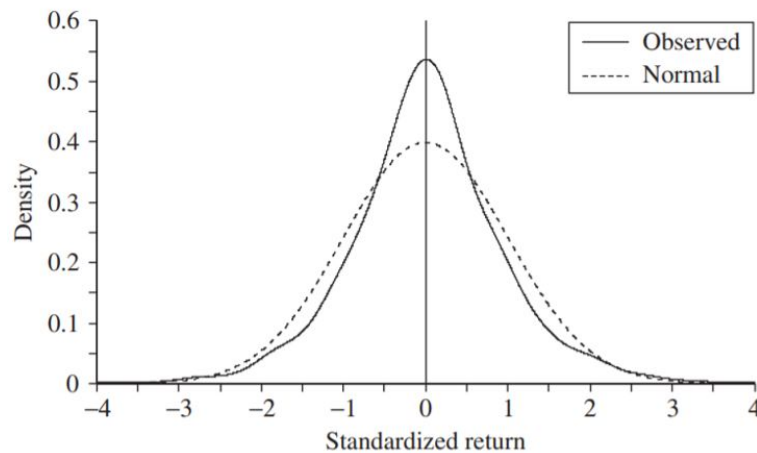


Figure 3.12: The shape of the S&P500 return distribution explains the fact that the observed standardized returns are approximately symmetric, have a fat tail and high peak distribution, which is not the same as the normal distribution.

2. There is minimal or no correlation between returns for different days. The sample autocorrelation of returns is generally close to zero, regardless of the time lag.
3. The correlation between absolute or squared returns for nearby days is positive and statistically significant. The function of returns can have substantial autocorrelation even if the returns are not autocorrelated.

Summary Statistics

The distribution of returns can be statistically summarized by its mean, standard deviation, skewness, and Kurtosis.

1. Mean

The mean is a statistical indicator that can be used to gauge the performance of a company's stock price over a period of days, months, or years, a company through its earnings over a number of years.

$$\bar{r} = \frac{1}{n} \sum_{t=1}^n r_t$$

2. Standard Deviation

Standard deviation is a statistical measure that describes the dispersion of price returns around its mean (Investopedia, 2019). Standard deviation is considered as a measure of historical volatility. Greater the standard deviation of the securities, the greater is the variance between each price and the means.

$$s^2 = \frac{1}{n-1} \sum_{t=1}^n (r_t - \bar{r})^2$$

3. Skewness

Skewness is the change in symmetry of a normal distribution bell curve. If the distribution of the data is even slightly tilted towards the right or left of a perfect symmetrical bell-shaped curve, then the data will be considered skewed. A normal distribution has a skewness of zero. If the distribution is right-skewed or positively skewed, it means the curve is bent toward the left, and a strong tail is observed on the right side of the curve. Similarly if the distribution is left-skewed or negatively skewed, it means the curve is bent toward the right, and a strong tail is observed on the left side of the curve.

$$b = \frac{1}{n-1} \sum_{t=1}^n \frac{(r_t - \bar{r})^3}{s^3}$$

4. Kurtosis

Kurtosis is another statistical measure used to describe the distribution of the data. Skewness focuses on one tail of the distribution at a time, whereas Kurtosis measures the combined weight of a distribution's tails relative to the center of the distribution. A standard normal distribution has a kurtosis value 3. Kurtosis value higher than 3 tends to have heavier tails than a normal distribution, and a value lower than three tend to have lighter tails than the normal distribution.

$$k = \frac{1}{n-1} \sum_{t=1}^n \frac{(r_t - \bar{r})^4}{s^4}$$

5. Z-Score

Z Score or a standard score tells how far a data point is away from the mean. Standardized returns are calculated to understand how many standard deviations away from the mean are returns distributed. This helps in understanding the volatility of the market.

$$z_i = \frac{x_i - \bar{x}}{S}$$

Where

Z_i – Z Score

X_i – Value of the price return

\bar{x} – Mean of the price returns

S – Standard Deviation

3.6 Volatility

Volatility is a measure of price variability over some period of time (Stephen J Taylor, 2007). Standard deviation is the most common measure of volatility if the returns are within five standard deviations around the mean. If the standard deviation is constant, volatility is unconditional, whereas if it is time-dependent or changing, it is called conditional volatility. The volatility in high frequency data was first discussed extensively by (Andersen and Bollerslev, 1998), who used realize variance to prove that standard volatility models do provide accurate forecasts. Daily volatility can be more accurately calculated by calculating realized variance, which is nothing but aggregation on intraday squared returns.

3.6.1 Realized Volatility (Historical Volatility)

The historical or realized volatility is the observed volatility during specific intervals of time; the historical is usually computed from daily data while the realized volatility is computed from more high frequency data (such as 1,5,10 or 15-minute data). Realized volatility was introduced as a measure of estimating daily volatility by adding high frequency intraday returns. Volatility can be more precisely calculated if the returns are sampled at high frequency and are uncorrelated. Realized volatility is measured by the sum of squared intraday returns.

$$\text{Realized Variance} = \sum_{i=1}^N r_t^2$$

Historical volatility is a method of measuring the variation in the price of the underlying assets, but since that measurement is just historical and the volatility is varying over time, it might not be a good way of measure future volatility.

3.6.2 Conditional Volatility

The conditional variance is the most widely estimated measure of volatility. The variance of the daily log-returns is assessed by an ARCH (Autoregressive Conditional Heteroscedasticity) framework conditional on the information set that is available on the previous day. The model-based approach to volatility forecasting is constructed from a model for returns, such as a GARCH type model that specifies the entire distribution of returns.

3.6.3 Time Series Volatility Models

ARCH (Autoregressive Conditional Heteroskedasticity)

Traditional econometric models consider constant mean and variance; however, ARCH introduced by (Engle, 1982) allows the conditional variance to change over time as a function of past errors leaving the unconditional variance constant. He defined a stochastic process whose variables have conditional mean zero and conditional variance given by a linear function of previous squared variables.

As the variance of the residuals or the error terms are not constant and are affected by the preceding variances, it is referred to as autoregression. The objective of the ARCH model is to measure volatility that can be used in financial decision making. The conventional notation for the ARCH model is ARCH(q) model where

q : The number of lag squared residual errors to include in the ARCH model.

The distribution of the return for period t , conditional on all previous returns are normal with constant mean μ and time-varying conditional variance h_t defined by

$$r_t | r_{t-1}, r_{t-2}, \dots \sim N(\mu, h_t)$$

and

$$h_t = \omega + \alpha(r_{t-1} - \mu)^2$$

The volatility parameters are $\omega > 0$ and $\alpha \geq 0$. The volatility of the return in period t

then depends solely on the previous return.

The residual at time t is

$$e_t = r_t - \mu$$

The model becomes more complicated if long term lags are needed to be included in the estimation. To allow for better volatility estimation dependent on long term lags, (Bollerslev, 1986) suggested an extension to the ARCH model, i.e., GARCH(p,q) model that incorporates a more flexible lag structure. GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model introduces a new parameter “p” that describes the number of lag variance terms:

p : The number of lag variances to include in the GARCH model.

q : The number of residual lag errors to include in the GARCH model.

GARCH (1,1)

The GARCH (1,1) is the extension of the ARCH (1) model, which includes a lagged variance term in the conditional variance equation. The GARCH (1,1) is one of the most popular ARCH specifications for modelling daily volatility. There are three reasons to support this hypothesis. First, the model uses four parameters (μ , α , β , and ω) which are easy to estimate. Secondly, it supports stylized facts for daily financial returns. Third, the forecast accuracy of the model is similar to the accuracy of the more complex volatility models.

The distribution of the return for period t , conditional on all previous returns, is defined by

$$r_t | r_{t-1}, r_{t-2}, \dots \sim N(\mu, h_t)$$

With

$$h_t = \omega + \alpha(r_{t-1} - \mu)^2 + \beta h_{t-1}$$

There are four parameters, namely (μ , α , β , and ω). The model is styled GARCH(1,1) because one previous squared residual and one prior value of the conditional variance is used to define the conditional variance for period t .

Computation behind ARCH Model

Building the ARCH model takes multiple iterations before it converges and finds optimal coefficients, namely α , β , and ω . This is due to the fitting of two equations at the same time, one of mean and another of variance. It works by fitting the model with certain coefficients, then check how well it has performed. Afterward, it adjusts the coefficients before moving to the next iteration. The program stops when the Log-Likelihood decreases and reached to a minimum point.

3.7 Important Classes of Spark SQL and DataFrames

Table 3.2: PySpark classes used in the research.

Language	Class	Description
PySpark	<code>pyspark.sql.Session</code>	Main entry point for DataFrame and SQL functionality.
	<code>pyspark.sql.Column</code>	A column expression in a DataFrame
	<code>pyspark.sql.Row</code>	A row of data in a DataFrame
	<code>pyspark.sql.DataFrameNaFunctions</code>	Methods for handling missing data (null values)
	<code>pyspark.sql.functions</code>	List of built-in functions available for DataFrame
	<code>pyspark.sql.types</code>	List of data types available
	<code>pyspark.sql.Window</code>	For working with window functions
	<code>pyspark.sql.GroupedData</code>	Aggregation methods, returned by <code>DataFrame.groupBy()</code>

Conclusion

Here the methods studied and implemented in the research are discussed with their limitations. The methods revolve around Big Data storage and processing framework along with cloud computing services offered by Amazon Web Services. The detailed analysis of the methods used in performing time series analysis of high frequency returns is performed. The two experiments implementing all the methods discussed here are presented in the next chapter.

Chapter 4

Implementation

This chapter focuses on implementing all the methods described in Chapter 3. Two experiments are studied and implemented here. The first experiment deals with estimating risk on historical data by considering volatility as constant. The second experiment takes into consideration real-time streaming data aggregated with historical data while assessing risk. In this case, volatility at time t is dependent on previous time periods; therefore, it needs to be modeled and forecasted. For both experiments, data is scaled in a binary fashion along with different cluster sizes. Then for the given data and cluster size as input, decision latency is calculated. Finally, after decision latency is observed and noted for each case, a model is built to estimate cluster requirements for analysing high-frequency time series. The whole architecture is built on the AWS Cloud platform. Access to the AWS resources has been possible only due to the help received by the School of Computer Science and Statistics, Trinity College Dublin.

4.1 Experiment 1

This experiment deals with estimating risk on historical 1-minute tick data from the year 2010 - 2019 in High-Frequency Trading market. This experiment aims to calculate the execution time for computing summarized statistics and estimating historical volatility. Here the mean and standard deviation of the 1-minute tick data is constant. Surely the

hypothesis here is that the time taken to compute descriptive statistics will be less than the time taken to compute heavy volatility time-dependent models. The flowchart in Figure 4.1 represents the end to end process starting from data extraction to decision making.

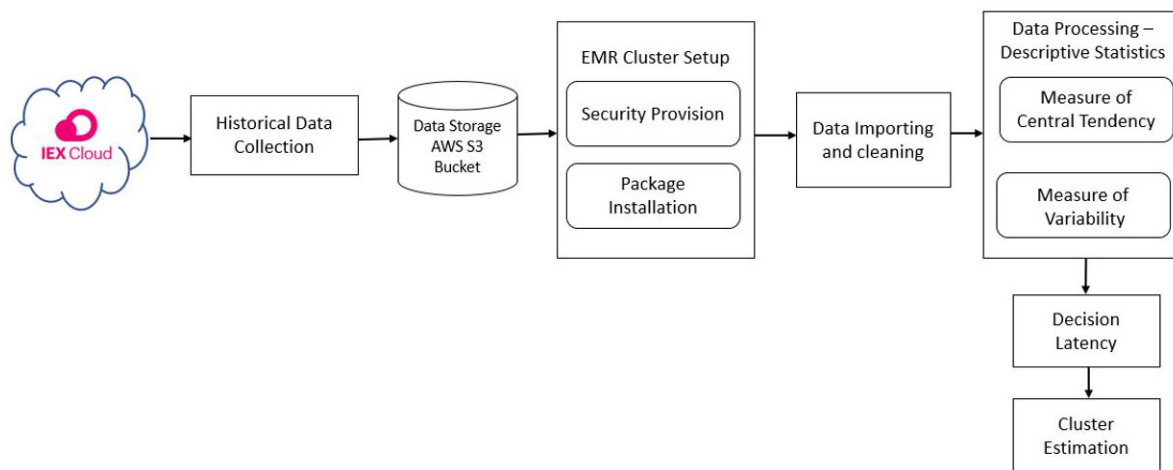


Figure 4.1: Flowchart Representation of experiment 1 – To observe decision latency in computing Descriptive Statistics and measure historical volatility by calculating standard deviation.

The process starts by collecting historical 1-minute tick data of E-Mini S&P 500 Futures data from the year 2010 – 2019. The source of the data is IEXCloud, which is a renowned data service provider for the stock market. Data received has approximately 3.4 million datapoints and is portioned into different ranges in a binary format. Each data file is stored in a secured AWS S3 bucket. Before creating clusters, pricing analysis is done, keeping in mind the AWS credits in hand. An EC2 key pair is created to authenticate the clusters before sign in. Also, two ports are opened, one to SSH into the master cluster and another one to open Apache Zeppelin Notebook. After the security configuration is completed, AWS EMR clusters are created. Before running the program for risk estimation, python packages are installed into the clusters. In the Zeppelin notebook, code is written to perform the operations on high frequency data. After performing implementation for both experiments on a different range of data input and varying cluster sizes, total execution time is observed and noted. In the end, when the execution time for each case is noted, analysis is done to estimate the cluster size based on data requirement

and decision latency. A detailed explanation of each method implementation is given in upcoming sections.

4.2 Experiment 2

There are two differences between the two experiments. In experiment one, data is historical, and the standard deviation is constant. Also, in experiment, descriptive statistics are performed on different inputs of data. In this experiment, the standard deviation is not constant or volatility at time t is dependent on time period $t-1$. The assumption is that the execution time for each scenario in this experiment is more than the execution time of each respective scenario in experiment 1. The models used for estimating and forecasting volatility are ARCH and GARCH, discussed in Chapter 3. The representation of the movement of data can be seen in the flow diagram in Figure 4.2.

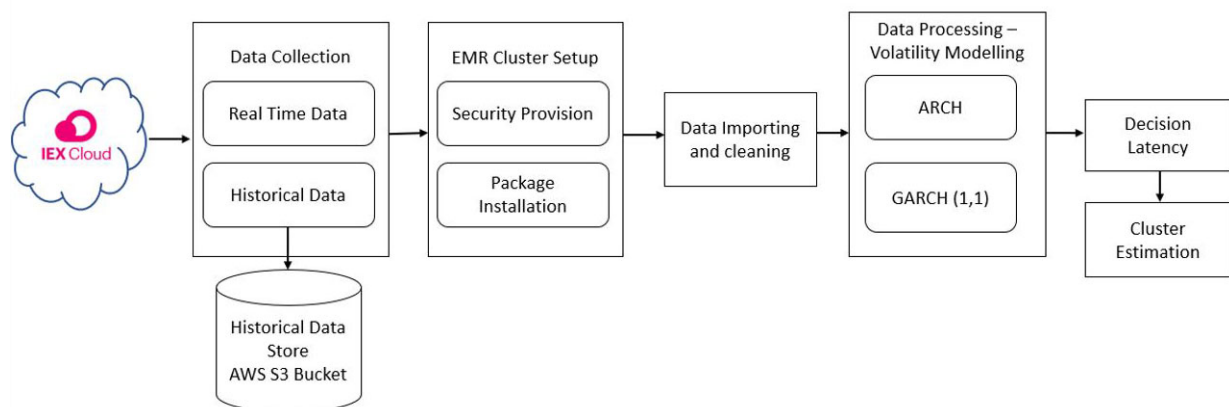


Figure 4.2: Flowchart Representation of experiment 2. To observe decision latency in computing time series volatility models namely ARCH (1) and GARCH (1,1)

The process starts with real-time data extraction of 1-minute tick data of E Mini S&P 500 Futures data from IEXCloud API. AWS Kinesis is used for real-time extraction. With the integration of AWS Kinesis and spark streaming, data is distributed over clusters. After the real-time information is consumed, it gets aggregated with historical data before computing risk models. Finally, programs for volatility models, i.e., ARCH and GARCH, are developed, and execution time is noted. Detailed implementation of the methods is provided in the upcoming sections.

Summary of the steps performing in experiment 2 is given below:

- a. Create an Amazon Kinesis stream.
- b. Spin up an EMR cluster with Hadoop, Spark, and Zeppelin applications from advanced options.
- c. Use a Simple Java producer to push 1-minute tick data from IEXCloud into the Amazon Kinesis stream.
- d. Connect to the Zeppelin notebook.
- e. Import the Zeppelin notebook, which contains a program for computing risk.
- f. Analyze and visualize the streaming data.

4.3 Data Discovery and Acquisition

To study the volatility pattern in High-Frequency Trading market, it is necessary to consider an exemplar time series. An exemplar dataset covers all the properties of a problem that had been observed before or will be observed in the future. As discussed in the initial chapter of research progression, I started generating synthetic data using Inverse CDF transformation and Monte Carlo simulation to fulfill the need for Big Data. After studying various literature on Big Data implications in the financial market, I concluded that the accuracy of data is highly essential for accurate risk estimation.

Initially, the time series considered was of SPY ETF¹ (Exchange Traded Fund) 5-minute. An ETF tracks typically the Standard & Poor's 500 Index, comprising of 500 large and mid-cap U.S. stocks. Due to its limited availability and also low volatile nature (refer (Wermers and Xue, 2015), I progressed towards searching for a more volatile time series. Unlike regular stock, futures contracts derive their value from an underlying asset, which can be a traditional stock, index, or a bond. The reason futures contracts are volatile is because unlike spot prices, they cannot be bought or sold immediately at a fixed price when Ask price meets Bid price. Futures are just like a bet in the future without knowing

¹<https://www.investopedia.com/articles/investing/122215/spy-spdr-sp-500-trust-etf.asp>

the underlying risk. This makes Futures highly volatile. There are several literature explaining the high volatile nature of Futures (Refer (Andersen, Bondarenko, et al., 2018), (Kurov and Lasser, 2004) and (Jiang et al., 2018)).

The major differences between the futures contracts and spot prices are given in Table 4.1

Table 4.1: Spot vs. Futures Prices

Factors	Spot Market	Futures Market
Exchange	Financial Assets	Financial Instruments (Contracts)
Examples	Stocks, Cryptocurrencies, Foreign Exchange Currencies	Futures, Forwards, Options, Swaps
Time Horizon	Executed Immediately	Executed at a specific future date
Complexity	Straightforward	Highly Complex
Volatility	Low, Medium	High

Finally, the data decided to be analyzed for risk estimation is S&P 500 mini, called the S&P 500 E-mini, which is a futures contract worth 1/5 the value of a standard S&P 500 futures contract. It is made up of 500 individual stocks representing the market capitalizations of large companies; the S&P 500 Index is a leading indicator of large-cap U.S. equities.

4.3.1 Historical Data Acquisition

S&P 500 mini futures data of 1-minute frequency is collected from the year 2010 – 2020. Some simple steps are followed to acquire the data:

- a. Register on IEXCloud by paying monthly subscription charges for an individual account.
- b. Receive your API key after completing full registration.
- c. Request JSON data using code listing 4.1. Convert the JSON data into CSV format.

Note - Last one-year data is available for free on IEXCloud and can directly be requested using the same API key. The detailed documentation on IEXCloud API is provided on its official website².

²<https://iexcloud.io/>

Code Listing 4.1: Code for Futures contracts data extraction through IEXCloud Web API from the year 2019-2020. Similarly, more data is extracted from the year 2010 – 2019

```
### Year 2019 - 2020
string1 = "https://sandbox.iexapis.com/stable/stock/ES/chart/date/"
string2 = "?token=Tpk_fdadb6f399f8496b895b4888c84701b5"
def getDataIEX(string1,string2):
    empty_item_list = []
    df2 = pd.DataFrame(columns = ["date", "minute", "close","volume"])
    for i in range(20190101,20200101):
        response = requests.get(string1 + str(i) + string2)
        test = response.json()
        if test == []:
            empty_item_list.append(i)
        else:
            df = pd.DataFrame(test)
            df2 = pd.concat([df2,df[['date','minute','close']]], axis = 0)
    return df2
```

Table 4.2: S&P 500 Mini Futures (ES) 1-minute intraday prices from the year 2010-2019

datetime	Open	High	Low	close	Volume
2010-01-03 18:02:00	938.75	939.75	938.75	939.25	437
2010-01-03 18:03:00	939.25	939.25	938.75	939.00	596
2010-01-03 18:04:00	938.75	939.25	938.75	939.00	131
2010-01-03 18:05:00	939.00	939.00	938.75	938.75	166
2010-01-03 18:06:00	938.75	939.25	938.75	939.00	305
...
2019-12-30 23:56:00	3205.50	3205.75	3205.50	3205.75	33
2019-12-30 23:57:00	3205.75	3205.75	3205.50	3205.50	17
2019-12-30 23:58:00	3205.50	3205.75	3205.25	3205.25	71
2019-12-30 23:59:00	3205.50	3205.50	3205.25	3205.50	50
2019-12-31 00:00:00	3205.25	3205.50	3205.25	3205.25	30

3478489 rows × 5 columns

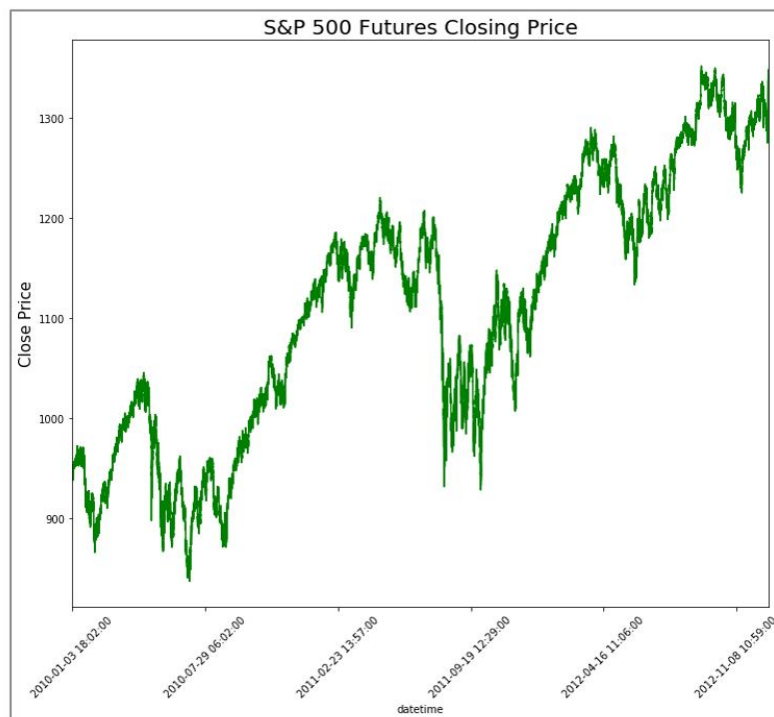


Figure 4.3: S&P 500 Mini Futures (ES) 1-minute intraday prices from the year 2010-2019

4.3.2 Data Partitioning and File Storage Format

Imported data is divided into eight parts in powers of two, which are often used to measure computer memory. Each experiment is implemented on the below-given number of datapoints and on the given clusters. Also, the processing for a given number of data points is done on 2, 4, and 8 clusters, and finally, the decision latency is noted in each case.

Table 4.3: Data Scaling

Format	Value
2^{16}	65536
2^{17}	131072
2^{18}	262144
2^{19}	524288
2^{20}	1048576
2^{21}	2097152
Till Last	3478488

Table 4.4: Cluster Scaling

Datapoints	Clusters		
65536	2	4	8
131072	2	4	8
262144	2	4	8
524288	2	4	8
1048576	2	4	8
2097152	2	4	8
3478488	2	4	8

Data Storage

Data is divided as given in the previous section and stored in AWS S3 (Simple Storage Service) bucket, which is a public cloud storage resource available in Amazon Web Services (AWS). The main advantage of using S3 object storage service is that it offers its customers leading solutions in terms of scalability, data availability, security, and performance.

a. Data Access and Availability

Data in S3 can be provided to any user in the world from a single location with read or write permissions.



Figure 4.4: AWS Data Permissions

b. Data Encryption

AWS S3 offers data encryption capabilities. Server-side encryption for data stored in the S3 bucket can either be done using Amazon S3-managed keys (SSE-S3) or customer master keys (CMKs) stored in AWS Key Management Service (AWS KMS).

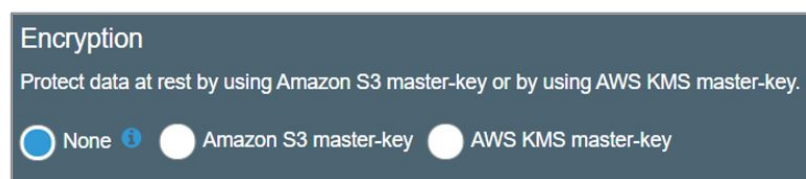


Figure 4.5: AWS S3 Object Storage Data Encryption

4.3.3 Real-time Data Extraction from Amazon Kinesis Streams Using Zeppelin and Spark Streaming

This section shows how we can use Amazon Kinesis streams for real-time data extraction. The actual implementation of AWS Kinesis data streams on High Frequency data has

not been possible in this research due to the limited AWS student credits. However, a theoretical structure is ready to be implemented in the future. The detailed end to end flow of to analyze Realtime Data from Amazon Kinesis Streams Using Zeppelin and Spark Streaming can be found on AWS official documentation³

As discussed in section 3.4.1, the Kinesis Producer Library (KPL) is an application that puts user data records into a Kinesis data stream (also called data ingestion). Here KPL extracts data from IEXCloud at every 1 minute, put the records into shards, then KCL (Kinesis Client Library) collects these data streams and sends it to Spark through Spark EMR clusters through Spark streaming where real-time streams get aggregated with the historical data in the S3 bucket. This flow can be visualized from the flow diagram shown in Figure 4.6. Also, detailed steps for writing and receiving data to and from AWS Kinesis data streams can be found on official AWS documentation⁴.

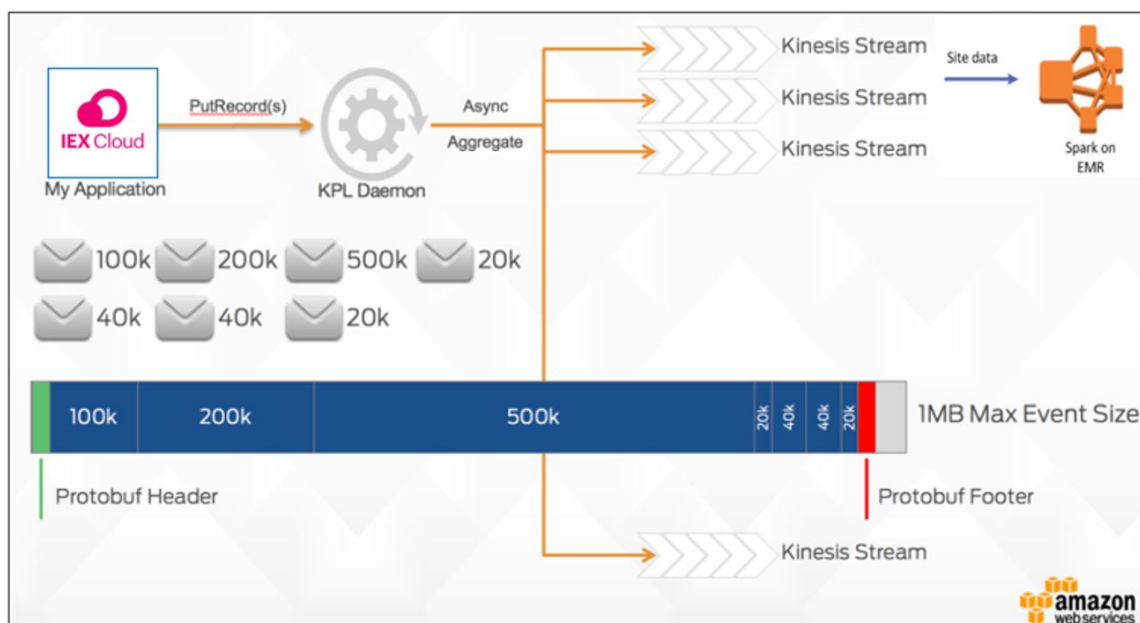


Figure 4.6: Real-Time Data Extraction and Analysis using AWS Kinesis Framework

KPL uses Java library to build a small application that extracts data from web sources such as IEXCloud, processes it, and distribute it to the shards. KPL and KCL applica-

³<https://aws.amazon.com/blogs/big-data/analyze-realtime-data-from-amazon-kinesis%2Dstreams%2Dusing%2Dzeppelin-and-spark-streaming>

⁴<https://docs.aws.amazon.com/streams/latest/dev/working-with-streams.html>

tions can be written in Python; however, Java needs to be installed in the system as its library will be running in the background to support development in Python. Detailed documentation on developing kinesis. The installation of Java libraries is given in the official documentation of AWS⁵.

The following python code is needed to write a minimal working producer.

Code Listing 4.2: KPL application written in Python to send records to shards aggregated up to 100KB, with every 60 seconds and joined with ‘\n’

```
from kinesis_producer import KinesisProducer
config = dict(
    aws_region='us-east-1',
    buffer_size_limit=100000,
    buffer_time_limit=0.2,
    kinesis_concurrency=1,
    kinesis_max_retries=10,
    record_delimiter='\n',
    stream_name='KINESIS_STREAM_NAME',
)
k = KinesisProducer(config=config)
for record in records:
    k.send(record)
k.close()
k.join()
```

⁵<https://aws.amazon.com/premiumsupport/knowledge-center/kinesis-data-stream-kpl/>

Data Aggregation

After data is received by Kinesis Client Library (KCL) application at every minute, it gets aggregated with the historical data. This can be more clearly visualized from Figure 4.7

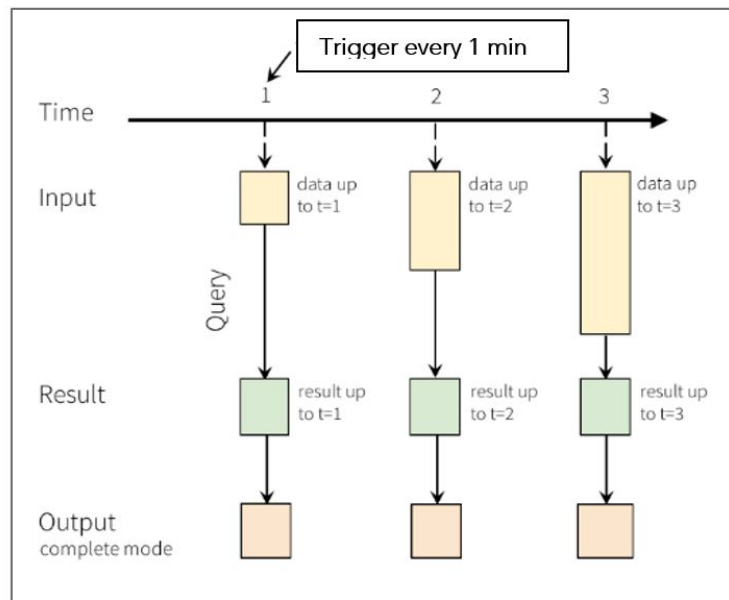


Figure 4.7: Data Aggregation (Historical and Real Time) at every 1-minute.

4.4 Environment Setup

The whole process of setting up the cloud computing machines involves four main steps, the fourth being explained within each experiment. First, it is important to note that with the AWS educated students account, it is only possible to perform computation on a maximum of 8 clusters; however, there is no limitation on the storage of the big data. Also, the use of real-time streaming platform, i.e., AWS Kinesis, has not been possible due to the cost limitations on the student account. Before creating clusters for computing, security settings need to be configured to access the master clusters. As discussed in Chapter 3, each master controls multiple slave clusters. All these clusters are nothing but EC2 Ubuntu machines combined together to form an EMR cluster. Access to the

master machine is done by Secure Shell (SSH) protocol. After access is granted, necessary packages are installed.

4.4.1 Security Configuration

There are two steps to configure the security setting for EMR clusters:

- a. Create an EC2 Key Pair, which consists of a public key and a private key. It allows users to download the private key, which is required to login to the master ubuntu machine using SSH. Detailed steps of creating an EC2 key pair are given in the AWS official documentation⁶.
- b. Configure security settings for the clusters to be created. This step is performed while the cluster is in the creation stage and is waiting to move to the active stage. Steps are given on the official AWS EMR page⁷.

In addition to the steps given on the page, one more rule is added to open the port for Apache Zeppelin notebook:

- Click Add rule and enter the following details:
- Type – Custom IP
- Port Range – 8890
- Destination – Anywhere

⁶Creating an EC2 Key Pair <https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair>

⁷Allow SSH to EMR Clusters: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-ssh.html>

The screenshot shows the 'Inbound rules' configuration page in AWS IAM. It features a table with columns for Type, Protocol, Port range, and Source. Two rules are visible: one for SSH (port 22) and one for Custom TCP (port 8890). The source for the SSH rule is 'My IP' with a specific IP range. The source for the Custom TCP rule is 'Anywh...' with two IP ranges: '0.0.0.0/0' and '::/0'. An 'Add rule' button is located at the bottom left of the configuration area.

Figure 4.8: AWS Security Settings. SSH lets user login to EC2 master Ubuntu machine using Putty. Custom TCP port 8890 is the default port to open Zeppelin Notebook running on distributed platform.

4.4.2 Setting up a Spark cluster on AWS EMR

The whole setup of Spark on AWS EMR is referred from (Bluhm, 2018) and official AWS documentation⁸.

This section explains the necessary steps taken to create EMR clusters involving specific type EC2 instances. If one does not have an AWS account yet, he/she needs to sign up for one. Once the account is set up, one has to create an S3 bucket and an Elastic Compute Cloud (EC2). In this section, I have only emphasized the steps specific to the use case, which are not described in the official documentation.

As the Python program is developed under Spark version 2.2.0, the first step is to select a corresponding EMR release when launching the cluster. Under advanced options, select emr-5.10.0 in the software configuration panel and make sure to check the box with Spark 2.2.0, Zeppelin 0.7.3 Notebook, Hadoop 2.7.3. Hive is only optional and depends on the use case.

⁸Detailed Guide to AWS EMR cluster: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-prerequisites.html>

Software Configuration

Release emr-5.10.0 ⓘ

<input checked="" type="checkbox"/> Hadoop 2.7.3	<input checked="" type="checkbox"/> Zeppelin 0.7.3	<input type="checkbox"/> Livy 0.4.0
<input type="checkbox"/> Tez 0.8.4	<input type="checkbox"/> Flink 1.3.2	<input type="checkbox"/> Ganglia 3.7.2
<input type="checkbox"/> HBase 1.3.1	<input type="checkbox"/> Pig 0.17.0	<input checked="" type="checkbox"/> Hive 2.3.1
<input type="checkbox"/> Presto 0.187	<input type="checkbox"/> ZooKeeper 3.4.10	<input type="checkbox"/> MXNet 0.12.0
<input type="checkbox"/> Sqoop 1.4.6	<input type="checkbox"/> Mahout 0.13.0	<input type="checkbox"/> Hue 4.0.1
<input type="checkbox"/> Phoenix 4.11.0	<input type="checkbox"/> Oozie 4.3.0	<input checked="" type="checkbox"/> Spark 2.2.0
<input type="checkbox"/> HCatalog 2.3.1		

Figure 4.9: Selection of the EMR type clusters. Necessary packages such as Spark, Hadoop, Zeppelin or optional Hive are installed on EC2 machines. Note EMR cluster is just collection of EC2 machines running concurrently.

Next, you have to configure the hardware of your cluster, including the instance type and the number of instances. While the hardware configuration strongly depends on the resource requirements (and budget considerations) of the specific use, the forecasting example in this paper is based on a general-purpose instance type, providing a balanced ratio of the number of CPUs relative to the amount of RAM.

Instance Details

Details of the Instance used are given in Table 4.5. Whole list of available pricing is on official AWS documentation ⁹

Table 4.5: Details of Instance used in the research along with its pricing.

Name	Type	Default vCPUs	Memory (GiB)	Network Performance	Maximum bandwidth (Mbps)	Maximum throughput (MB/s, 128 KiB I/O)	Maximum IOPS (16 KiB I/O)	Price
m4.xlarge	M4	16	64	High	2000	250	16,000	\$0.24 per Hour

⁹A list of available instance types and prices can be found at <https://aws.amazon.com/de/ec2/pricing/on-demand/>

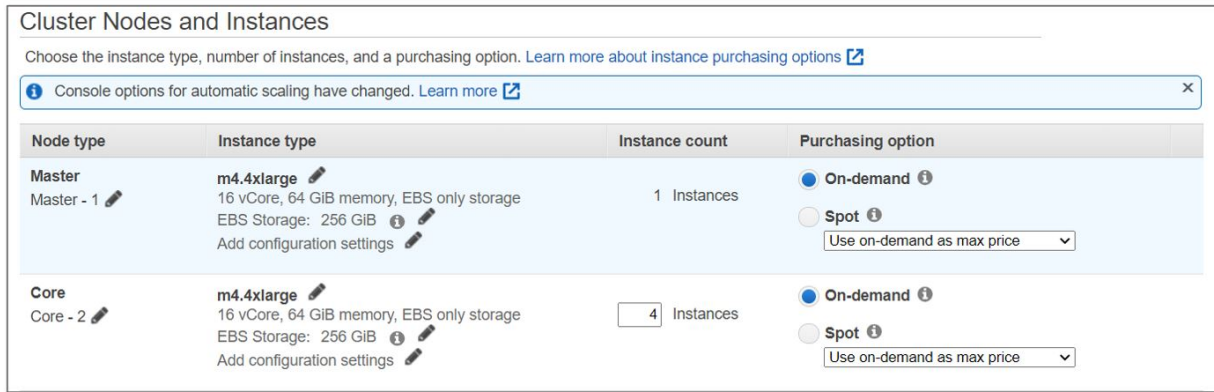


Figure 4.10: Screenshot of AWS interface showing selection of clusters mentioned in Table 4.5

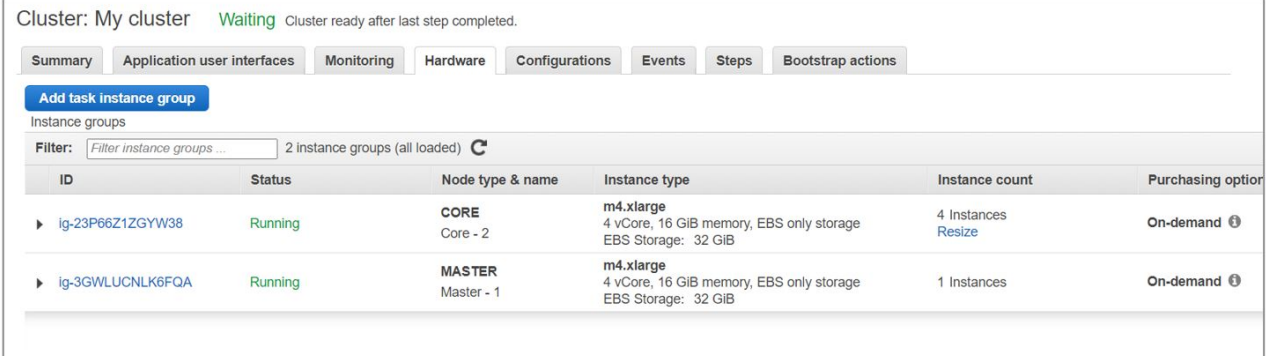
The two types of nodes used in this research are Master and Core. The master node handles the distribution of the task to core nodes, while the core node handles all the processing.

Once you have selected your preferred hardware configuration, go to the next section, which asks you to specify some general cluster settings. A bootstrap action panel is present at the bottom of the page where custom actions can be specified to install additional Software or to customize the configuration of cluster instances. In essence, bootstrap actions are scripts that run on all nodes after the cluster is launched. We will use the bootstrap action to install a few python libraries required for the parallel forecasting exercise. For this purpose, a shell script called `install python libraries.sh` with the following content must be uploaded to a folder in the S3 bucket:

Code Listing 4.3: Install python libraries.sh

```
#!/bin/bash -xe
sudo yum install python36 python36-pip
python3 -m pip install arch
python3 -m pip install matplotlib
python3 -m pip install seaborn
python3 -m pip install sklearn
python3 -m pip install scipy
```

Next, add the bootstrap action by selecting custom action and, under the configure and add button, browse the S3 path to the shell script (no optional arguments needed). After adding the custom bootstrap action, move on to the last step security settings, which are explained in the previous section of security configuration. Finally, the cluster can be created.



ID	Status	Node type & name	Instance type	Instance count	Purchasing option
ig-23P66Z1ZGYW38	Running	CORE Core - 2	m4.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 32 GiB	4 Instances Resize	On-demand ⓘ
ig-3GWLUCNLK6FQA	Running	MASTER Master - 1	m4.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 32 GiB	1 Instances	On-demand ⓘ

Figure 4.11: EMR clusters (2 Cores or 4 Instances) running with installed Spark framework.

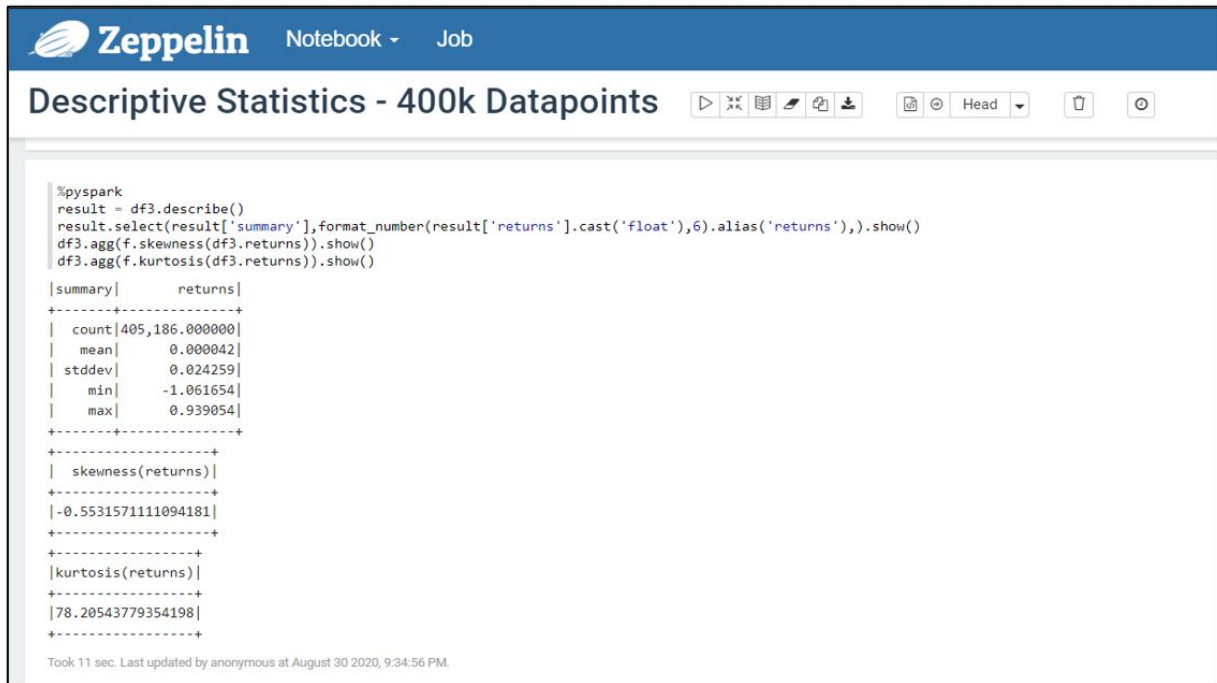
To deploy the Spark application with our parallel forecasting algorithm to the master node, we have to establish an SSH connection between our local machine and the master node and create an open port for Apache Zeppelin notebook, detailed steps are given in section Section Configuration (See 4.4.1).

4.5 Risk Estimation

4.5.1 Summarized Statistics

Few lines of code are written in PySpark, which is a Python API for Spark for estimating the statistical moments of returns. This program runs for different scales of data input on each of the given cluster sizes (2,4 and 8). Statistical moments of returns for one of the use cases on 400,000 datapoints are calculated as displayed in Figure 4.12. The image is of the Zeppelin notebook, which is in the master machine handling spark jobs on the slave machines. Standard deviation, which is considered as a standard measure of volatility, is calculated along with the other measure of variability such as skewness

and kurtosis. However, before calculating the descriptive statistics, some steps such as cleaning data, calculating returns from prices, and observing stylized facts for returns are performed.



```

%pyspark
result = df3.describe()
result.select(result['summary'],format_number(result['returns'].cast('float'),6).alias('returns')).show()
df3.agg(f.skewness(df3.returns)).show()
df3.agg(f.kurtosis(df3.returns)).show()

```

summary	returns
count	405,186.000000
mean	0.000042
stddev	0.024259
min	-1.061654
max	0.939054

```

skewness(returns)
-0.5531571111094181
kurtosis(returns)
78.20543779354198

```

Took 11 sec. Last updated by anonymous at August 30 2020, 9:34:56 PM.

Figure 4.12: Output Window for one of the use case in experiment 1 - Descriptive Statistics (Measuring Historical Volatility)

4.5.2 Volatility Model

Usually, the standard deviation is assumed to be a constant of the series, but in some situations, there are fluctuations in its values. Volatility is sometimes computed using historical time series, and we have to estimate the means and standard deviations using linear and non-linear models. But in the case of streaming data where the standard deviation is not constant and keeps changes. Volatility needs to be modeled in order to understand the fluctuation pattern before making quick decisions. Here time-series volatility models such as ARCH and GARCH (1,1) are implemented for forecasting volatility. The total time to train, test, and predict volatility using ARCH and GARCH volatility models is high as compared to just calculating historical volatility using standard deviation. Volatility modelling through GARCH(1,1) for one of the use cases on 400,000 datapoints is done using the Python library for, as shown in Figure 4.13.

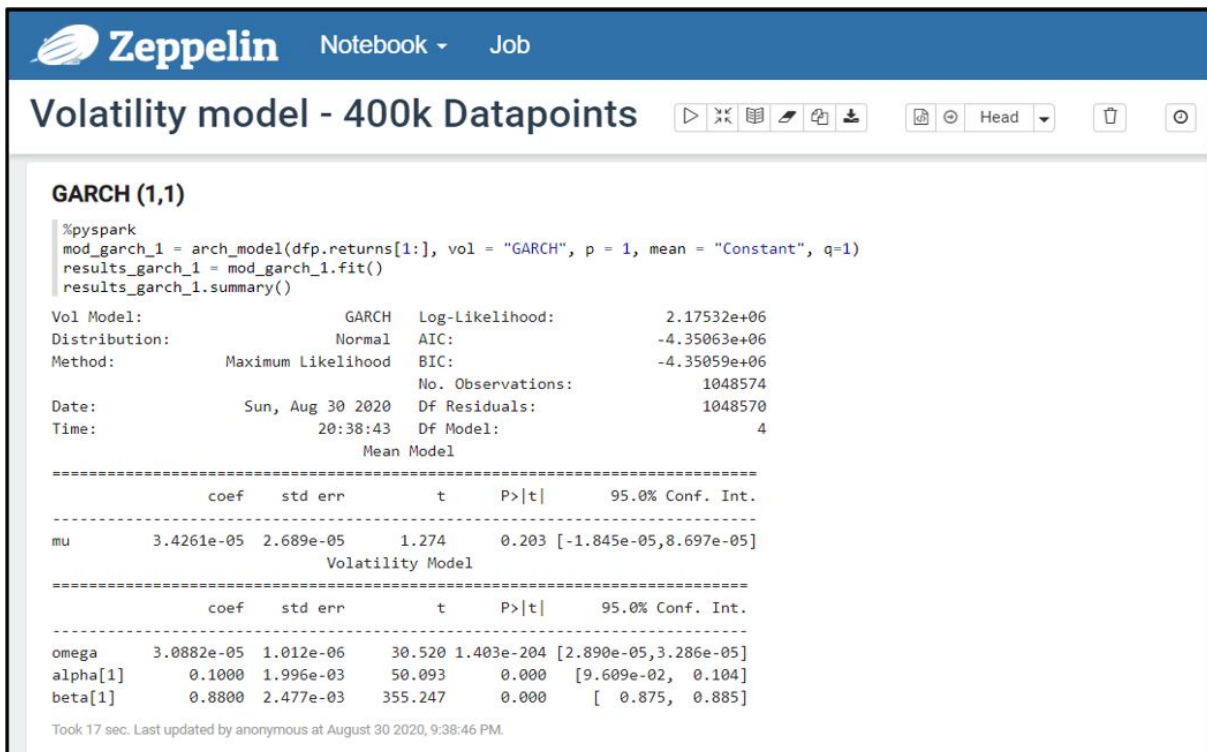


Figure 4.13: Output Window for one of the use case in experiment 2.

Modelling volatility using 400k datapoints

4.6 Decision Latency

Decision latency or the system execution time is noted in each case. Each scenario is executed five times, and then the average of the execution time is considered. This is to cover the different range of execution time observed while computing volatility. The method to calculate decision latency is to just take the system time at the start of the run and the system time at the end of the run. The difference between the two execution times is decision latency.

4.7 Conclusion

This chapter dives deep into the implementation of the methods described in Chapter 3 followed in a way as given in the architecture. Some of the main methods implemented in this section are a collection of exemplar time series from IEXCloud, both historical and real-time, big data storage, parallel computing framework Apache Spark, security configuration, and decision latency. Lastly, a screenshot of the interface where the program is running is displayed.

Chapter 5

Results

5.1 Introduction

Two main challenges which big data analysts face are latency and scalable computing. Low latency, which means systems respond quickly to actions, can help traders take rapid decisions without depending on computation power. Capturing market trends at every interval, storing the massive volume of structured data, and analysing the data at a regular frequency to reveal high-profit margins is not possible without achieving low latency. It is easy to observe low latency with less volume of data; however, if data is scaled, it is difficult to make trading decisions within a defined interval due to limitations of computing power.

The standardized intraday future returns¹ for S&P 500 Mini² are plotted, as shown in Figure 5.1. At many points, returns are deviating by more than 5 standard deviation from the mean which accounts for high volatility. The objective of this chapter is to analyze decision latency observed in calculating intraday volatility either by measuring standard deviation or by modelling it through time series models. To model intraday volatility, GARCH (1,1) model is used in experiment 2. In experiment 1, historical volatility is computed with other statistical moments of returns.

¹<https://www.investopedia.com/terms/z/zscore.asp>

²https://www.investopedia.com/terms/s/sp_500_mini.asp

This chapter starts with proving the three stylized facts for financial returns on intraday futures data. Then results of each experiment are discussed along with the detailed analysis of the computation time, i.e. decision latency on a different scale of data and sizes of clusters.

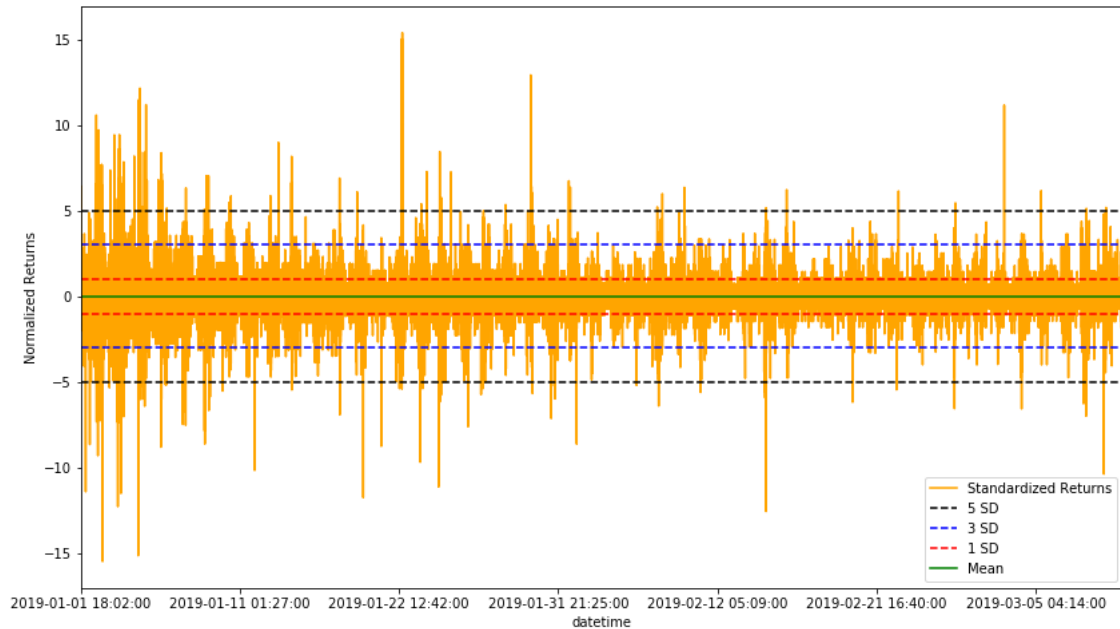


Figure 5.1: Intraday Future Standardized Returns of S&P 500 Mini from 2019-01-01 to 2019-03-11. At many points, returns are deviating by more than 5 standard deviation from the mean which accounts for high volatility.

5.2 Stylized Facts for Financial Returns

There are three significant stylized facts of financial returns are common to all different types of assets as mentioned in Chapter 3.5. Although for both experiments, I have taken different scales of datasets, estimated volatility, and noted the decision latency. For calculating stylized facts, I have considered Futures S&P 500 Mini 1-minute tick data with a volume of 65536 data points. The detailed analysis of stylized facts is presented in the next three sections.

5.2.1 First stylized fact

The first stylized fact is that the distribution of returns is not normal. It is approximately symmetric, has fat tails. Summary of the statistical moments of returns is provided in Table 5.1, and standardized distribution can be seen from Figure 5.2.

The distribution of returns is approximately symmetric. The distribution has a very long tail and normalized returns reach to maximum 15 standard deviations from the mean (see Figure 5.2). The high peak describes the maximum returns are concentrated within 0.5 standard deviations (refer Table 5.2). Skewness statistics are used to represent the symmetry of distributions, and kurtosis statistics are used to interpret as a measure of similarity of the distribution to a normal distribution. The S&P 500 Mini Futures returns from 2019-01-01 to 2019-03-11 has positive skewness equal to 0.0121 and high kurtosis value of 15.25. Positive skewness indicates that the right tail is stronger and the mass of the distribution is concentrated on the left-hand side of the figure. The Kurtosis of any normal distribution is three and the kurtosis value of 15.25 in this case, indicates that it has strong fatter tails than a normal distribution as shown in Table 5.2.

Table 5.1: Descriptive Statistics on log-returns of S&P 500 Mini Future series for nearly three months dated 2019-01-01 to 2019-03-11. Note that returns are not normally distributed, have fat tails and high peak. The mean is near zero. Skewness is slightly greater than zero, unlike in normal distribution where it is zero. High Kurtosis of 15.25 explains the heavy tails, distinct peakedness near the mean. It is much higher than the Kurtosis of a normal distribution which is 3.

Mean	0.000001
StDev	0.000247
Variance	0.000000
Skewness	0.0121
Kurtosis	15.2546
N	65535
Minimum	-0.003832
1st Quartile	-0.000095
Median	0.000000
3rd Quartile	0.000095
Maximum	0.003812

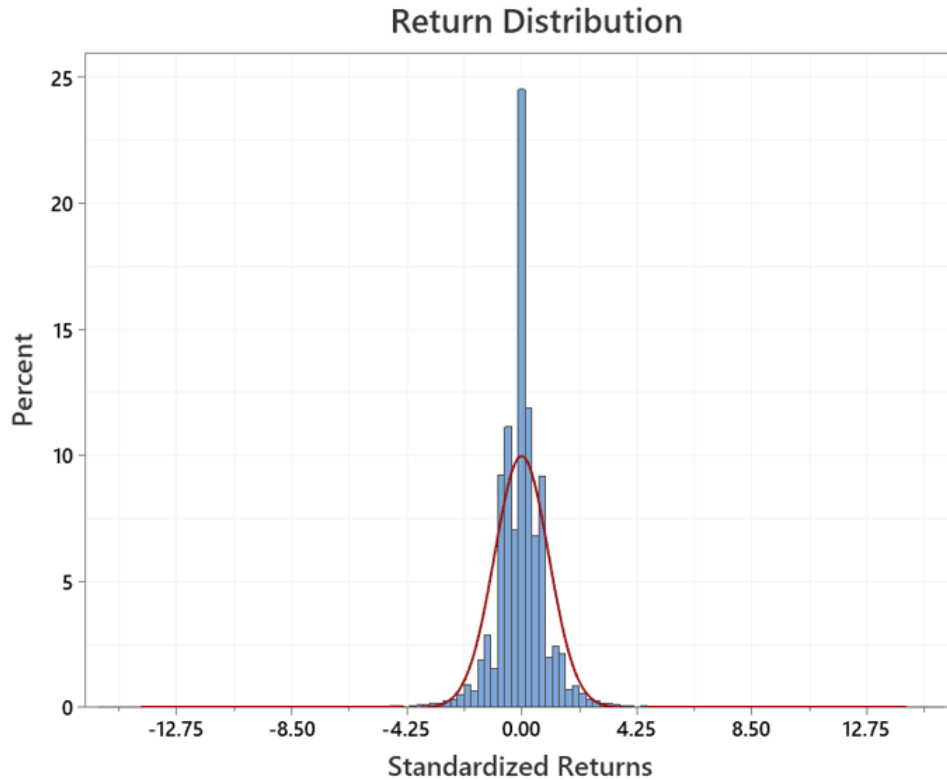


Figure 5.2: Return Distribution. The distribution has a very long tail and normalised returns reach to maximum 15 standard deviations from the mean. The high peak describes the maximum returns are concentrated within 0.5 standard deviations (see Table 5.2)

Table 5.2: Frequency for standardized returns of S&P 500 Mini Futures. The frequency of observations lies below $\bar{r}-3s$ or above $\bar{r}+3s$ is approximately six times as it of the normal distribution, corresponding to the high value of Kurtosis

Range	Observed	Normal	Observed minus normal
0 to 0.25	24.50%	19.70%	4.80%
0.25 to 0.5	36.80%	18.60%	18.20%
0.5 to 1	18.30%	30.00%	-11.70%
1.0 to 1.5	10.50%	18.40%	-7.90%
1.5 to 2	5.01%	8.80%	-3.79%
2 to 3	2.90%	4.30%	-1.40%
3+	1.70%	0.30%	1.40%

The percentages of standardized returns, $(r_t - \bar{r})/s$, within various intervals are summarized and compared with returns in a normal distribution as shown in Table 5.2.

The first two rows of Table 5.2 show 61.37% of total returns lie within 0.5 standard deviations from the mean which when compared to a value of 38.30% in case of normal distribution is exceptionally high. So, there are more observations in the range from $\bar{r} - 0.5s$ to $\bar{r} + 0.5s$ than are expected from a normal distribution, corresponding to a high peak in empirical distributions. The final row shows there are also more extreme observations, either below $\bar{r} - 3s$ or above $\bar{r} + 3s$, corresponding to two fat tails which we have already discussed before. The high values of Kurtosis are caused by the outliers in the tails. As the frequencies total 100%, there must be fewer observations elsewhere that occur beyond $\bar{r} \pm 3s$. Note that the high peak and fat tails effects are interdependent because extreme returns contribute large squared returns to the variance of the distribution, which implies there must be more observations near the centre of the distribution than are found for a normal distribution having the same mean and variance.

5.2.2 Second stylized Fact

There is almost no autocorrelation between the returns of different periods.

5.2.3 Third stylized Fact

There is positive dependence between absolute returns on nearby days, and likewise for squared returns.

A **correlation** of -1 indicates a perfect negative **correlation**, meaning that as **one** variable goes up, the other goes down. A **correlation** of $+1$ indicates a perfect positive **correlation**, meaning that both variables move in the same direction together. The prices are highly autocorrelated, even till 40 lags. Returns which move around mean are not correlated. However, the function of returns can have substantial autocorrelations even though returns do not. The correlation coefficients are positive for transformed

returns. It means returns are dependent on the returns on previous periods, but the form of dependency is not linear. The evidence for such non-linear dependence is obtained here by considering the autocorrelations of various powers of absolute returns, $|r_t|$ and r_t^2 . Squared returns are therefore used to measure realised intraday volatility.

Table 5.3: Autocorrelation in S&P 500 Mini Futures returns from 2019-01-01 to 2019-03-11. Proving second and third stylized fact that prices are highly autocorrelated, log returns are not autocorrelated or close to zero, squared returns are positively autocorrelated, and lastly absolute returns have high positive autocorrelation than squared returns. There is a very high correlation between prices at time t and $t+1$. The correlation between log return r_t and r_{t+1} is negative and continues for a few lags after which it becomes positive proving that return reverts to mean. There is a positive dependence between absolute returns on nearby days, and likewise for squared returns.

Type	1 Lag	2 Lag	3 Lag	4 Lag	5 Lag
Close Price	0.998	0.997	0.993	0.990	0.988
Log Returns	-0.047	-0.002	-0.011	0.013	-0.007
Squared Log Returns	0.18	0.14	0.15	0.12	0.12
Absolute Log Returns	0.30	0.277	0.272	0.26	0.25

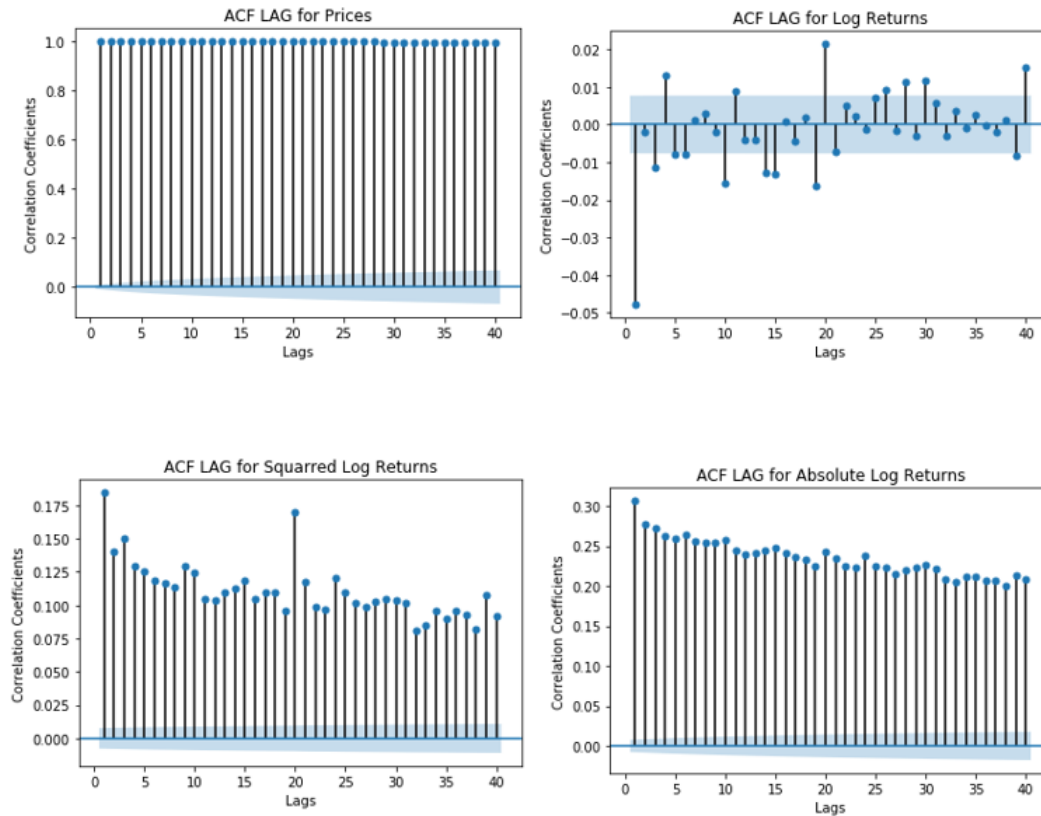


Figure 5.3: Autocorrelation summary of Table 5.3 but with up to 40 lags. A plot of the autocorrelation of a time series by lag is called the Autocorrelation Function, or the acronym ACF. This plot is sometimes called a correlogram or an autocorrelation plot. Running the example creates a 2D plot showing the lag value along the x-axis and the correlation on the y-axis between -1 and 1. The greater the distance in time, the more unlikely that autocorrelation exists

5.2.4 Latency in Stylized Facts

The stylized facts are just computed to infer the behaviour of returns and to verify if these are common to all different type of asset as stated in (Stephen J Taylor, 2007). To keep in line with the objective of estimating latency, the time it took to compute these stylized facts on 65536 datapoints on 2, 4 and 8 clusters are 24.38 seconds, 19.47 seconds and 15.34 seconds. As a common observation, the latency has reduced on increasing the cluster size. Analyzing latency for all different scale of data sizes is not the focus of this

section.

5.3 Experiment 1 Results

The time it takes to compute historical volatility by calculating standard deviation and also the descriptive statistics is less than computing time-dependent volatility models such as GARCH. Analysis of decision latency resulted from each execution and estimation of clusters is done on each scale of data input, i.e. from approximately 65,000 datapoints to 3.4 million data points and on different sizes of clusters. However, evaluating the output of summary statistics and volatility models in each case is not the focus of this research. Therefore, the results of descriptive statistics and ARCH models are only discussed for one of the data inputs i.e. with data of 1 million datapoints.

5.3.1 Experiment 1

As discussed in Chapter 4, in this experiment, the focus is to compute summary statistics of historical financial returns with different sizes of data inputs. As the volume of data increases, the computation time for calculating the four statistical moments of returns increases, and it undoubtedly impacts the decision time resulting in low-profit margins in High Frequency Trading. To avoid this bottleneck, three cluster sizes are taken into consideration while executing each scenario to confirm that total decision time is less than the computation time. In the upcoming two sections, I discuss the brief output of summary statistics calculated on 1 Million data points and the detailed analysis of decision latency.

Summary Statistics for High Frequency Returns Considering 1 Million data points as input.

The statistical characteristics of the distribution of a set of returns can be summarised by numbers such as their mean, standard deviation, skewness, and kurtosis. These summary statistics are presented in Table 5.4 and Figure 5.4 for the S&P 500 future series of 1-

minute returns introduced in Section 4.5.1. The table also includes the minimum and maximum returns with other statistical measures to summarise returns. The mean of the 1-minute high-frequency return dated from the year 2010-01-03 to 2013-01-02 is near zero with a standard deviation of 0.039%. The series has high Kurtosis of 99.009 and negative skewness of -0.29. Negative skewness indicates that the left tail is stronger and longer than the right tail. Also, the left tail is stronger, and the mass of the distribution is more concentrated on the right side as compared to a normal distribution.

Table 5.4: Summary Statistics for Future Returns computed on 1 million data points. \bar{r} , s , b , and k are the mean, standard deviation, skewness, and Kurtosis for the time series.

Series	Date Range	$10^4 \bar{r}$	$10^2 s$	b	k
S&P 500 Mini Futures	2010-01-03 -2013-01-02	0.0034	0.039	-0.29	99.009

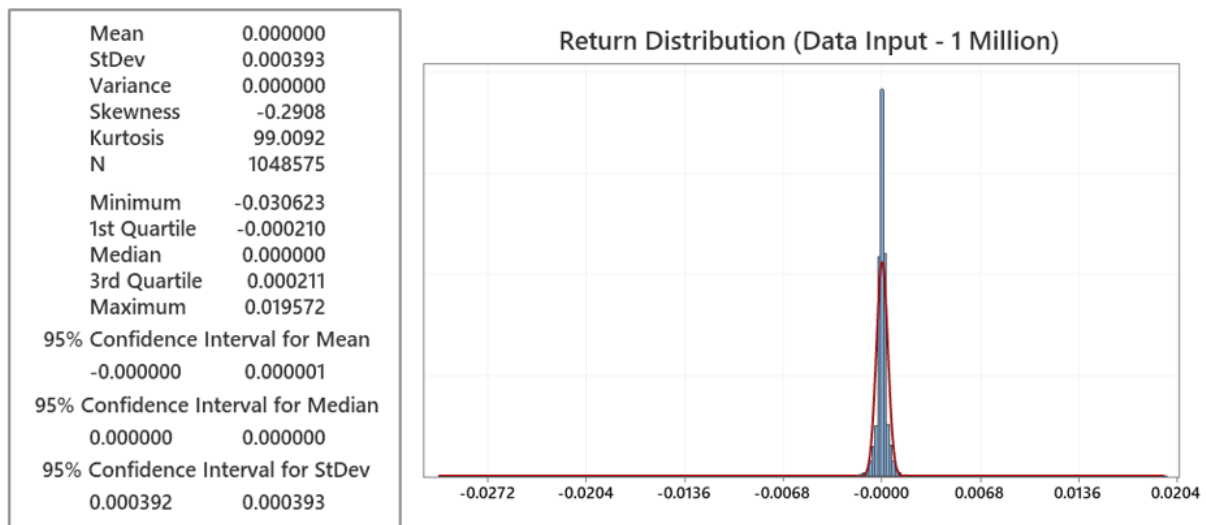


Figure 5.4: Descriptive Statistics of 1 Minute Returns for S&P 500 Mini Future Series. The mean of the 1-minute high-frequency return dated from the year 2010-01-03 to 2013-01-02 is near zero with a standard deviation of 0.039%. The series has high Kurtosis of 99.009 and negative skewness of -0.29. Negative skewness indicates that the left tail is stronger and longer than the right tail. Also, the left tail is stronger, and the mass of the distribution is more concentrated on the right side as compared to a normal distribution.

5.3.2 Cluster Estimation based on Decision Latency

The first column of Table 5.5 represents the number of data points taken for computing risk. It ranges from approximately 65,000 to 3.4 Million. The second column shows the number of clusters used in an execution in a scale of 2,4 and 8. The third column provides decision latency which is the execution time for computing summarised statistics and the most crucial historical volatility. This execution time is the average time taken in computing the same scenario for five times. The reason for this approach is the changing response time at each execution. The fourth column explains the improvement in execution time when cluster sizes are changed first from 2 to 4, then 4 to 8, and lastly from 2 to 8. Information on change in cluster size can be seen in the fifth column.

Summarizing the first case in Table 5.5 where the number of datapoints taken for computing descriptive statistics are 65536. The execution time for computing summary statistics on 2 clusters is 4.91 seconds, on 4 clusters is 3.06 seconds, and on 8 clusters is 2.68 seconds. The improvement in execution time moving from 2 clusters to 4 clusters is 60.37%, 4 clusters to 8 clusters is 13.99%, and 2 clusters to 8 clusters is 82.81%. Similar results are self-explanatory in Table 5.5.

Common Observations

From Table 5.5 it is evident that the execution time in computing summary statistics decreases with an increase in the size of clusters. The greatest improvement in performance is observed on moving from 2 to 8 clusters where decision latency is reduced by more than half. It is clear that decision latency increases with increase in size of the dataset. In a 1-minute trading environment, if minimum of 30 seconds is needed as a buffer to take trading decision, and the dataset has more than 2 million data points, it is almost impossible to compute summary statistics on 8 or less than 8 clusters. In that case, more than 8 clusters needs to be adapted by the trading firm.

Table 5.5: Decision Latency for experiment 1 on different sizes of data and clusters.

Descriptive Statistics				
Data Scaling (Cases)	Number of Data points	Number of Clusters	Execution Time (Seconds)	Speed Improvement (%) From Clusters 2-4, 4-8, 2-8 Respectively
2^{16}	65536	2	4.91	60.37%
		4	3.06	13.99%
		8	2.68	82.81%
2^{17}	131072	2	6.10	39.01%
		4	4.39	12.60%
		8	3.90	56.52%
2^{18}	262144	2	7.09	17.55%
		4	6.04	2.71%
		8	5.88	20.74%
2^{19}	524288	2	11.34	4.13%
		4	10.89	50.00%
		8	7.26	56.20%
2^{20}	1048576	2	32.19	58.38%
		4	20.33	59.56%
		8	12.74	152.70%
2^{21}	2097152	2	52.71	32.20%
		4	39.87	71.71%
		8	23.22	127.00%
Till End	3478488	2	89.19	6.70%
		4	83.59	85.60%
		8	45.04	98.03%

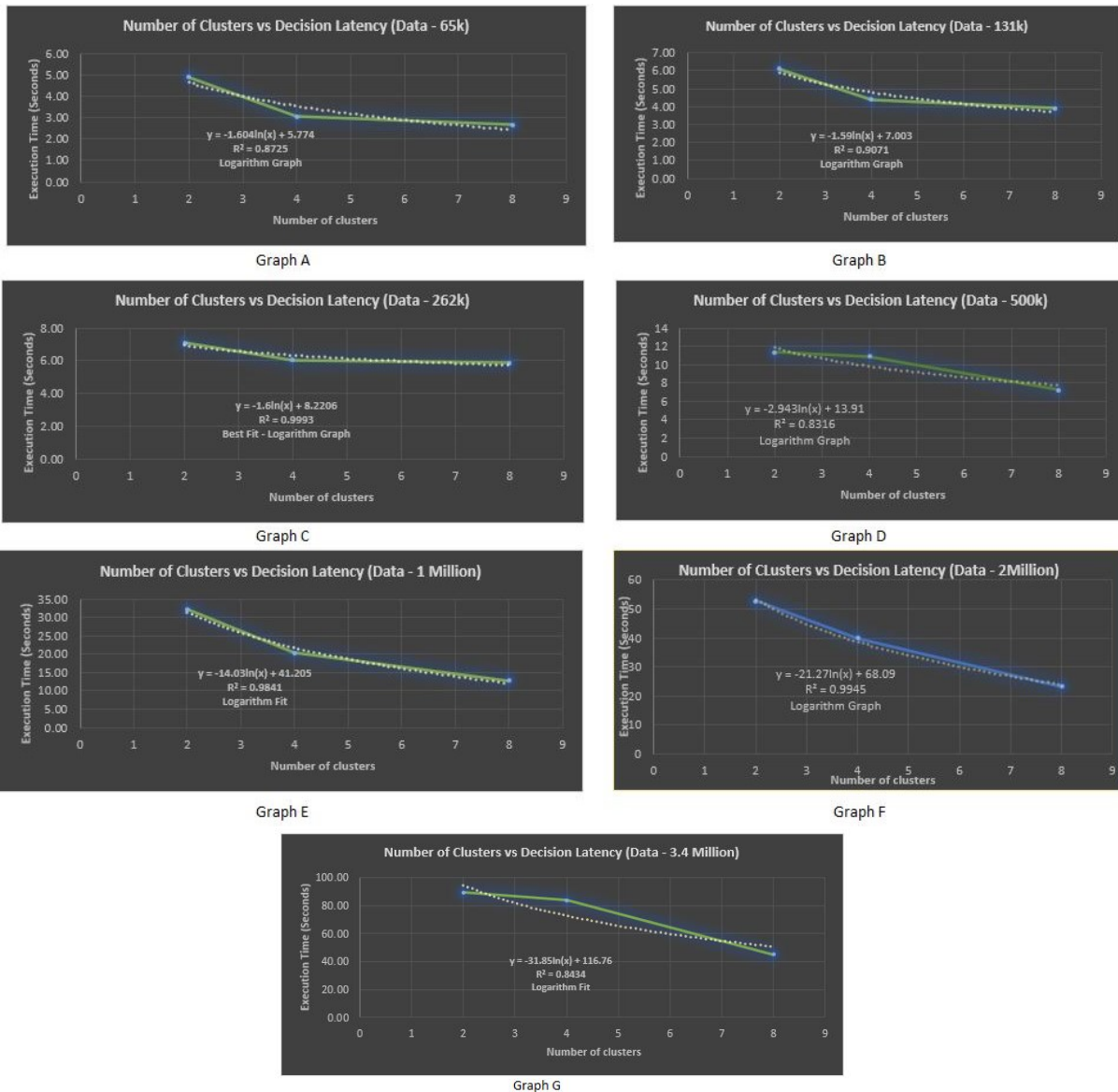


Figure 5.5: Graphs (A-G) describing the relationship between execution time and number of clusters. It is clearly observed that increasing the size of cluster reduces decision latency, after which the logarithm curve leads towards a saturation point post which there cannot be more improvement in performance even with increase in size of cluster. There is a little scope in improvement of decision latency when jumping from 8 clusters to 16 clusters.

As observed from Figure 5.5 there is less scope in improvement of decision latency when less amount of data is used for computing descriptive statistics proved in each of the first four graphs (A-D) of Figure 5.5, here the decision latency is not reduced by more than 3 seconds. However, in graphs (E,F,G) where more than 1 million data points are used

for computation, the minimum reduction in decision latency is around 20 seconds. These graphs shows that the improvement in decision latency is observed on scaling size of the clusters. The curve leads towards a saturation point post which there cannot be more improvement in performance.

5.3.3 Change in Decision Latency based on Data Points

The first column from Table 5.6 shows the different sizes of data points (as the power of 2) taken as input to compute summary statistics. The second, third, and fourth columns provide the execution time on each of the 2,4 and 8 clusters. It is clear from Table 5.5 that with the increase in the size of the data, the computation time or latency increases. This behaviour is the same with all different sizes of clusters. It is also observed that as we reduce the size of the dataset, the improvement in execution time also decreases. For instance, computation time on 3.4 million datapoints is 89.19 seconds on 2 clusters, 83.59 seconds on 4 clusters, and 45.04 seconds on 8 clusters. On moving to 2 million data points, decision latency is reduced by approximately half. On the other hand, when we move from 0.26 million data points to 0.13 million data points, the computation time only decreases by a few seconds.

On an exponential graph of decision latency, even though the size of data is increasing along with size of the clusters, a point is observed where the rate of decline starts to level off when that exponential growth has stopped (see Figure 5.6)).

Table 5.6: Decision latency on a different scale of data inputs. Every time the data size is doubled execution time increases by some specific percentage

Decision Latency (Seconds)				
Data Points		2 Clusters	4 Clusters	8 Clusters
2^{16}	65536	4.91	3.06	2.68
2^{17}	131072	6.10	4.39	3.90
2^{18}	262144	7.09	6.04	5.88
2^{19}	524288	11.34	10.89	7.26
2^{20}	1048576	32.19	20.33	12.74
2^{21}	2097152	52.17	39.87	23.23
Till Last	3478488	89.19	83.59	45.04

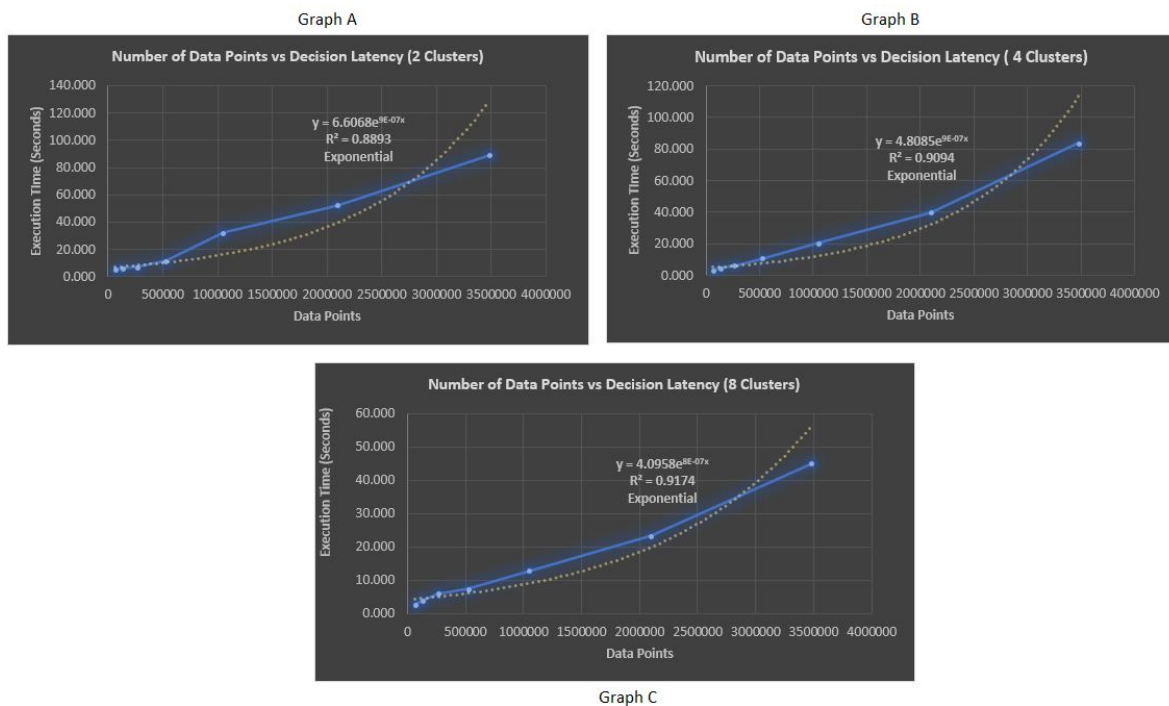


Figure 5.6: Graphs (A, B, and C) showing relation between decision latency and data points on each given cluster. Latency is increased with an increase in size of the data.

5.4 Experiment 2 Results

The objective of this research is to compute heavy volatility models such as ARCH and GARCH and to observe the decision latency on different size of data inputs. Critical evaluation of the models is out of the scope of this research as the objective is to calculate latencies based on which clusters are estimated.

5.4.1 Experiment 2

Volatility modelling on High Frequency Returns Considering 1 Million data points as Input

Econometric models assume that the variance of the error terms or residuals will be uniform. This is known as Homoskedasticity. However, in some circumstances, this variance is not uniform, as in the case of Heteroskedasticity where the variance of these error terms is not just non-uniform but is affected by variances preceding it. It is crucial to note that ARCH family models are used to predict future variance rather than future returns. This is particularly true in time-series analysis of financial markets. For example, in securities markets, periods of low volatility are often followed by periods of high volatility. So, the variance of the error term describing these markets would vary depending on the variance of previous periods.

In this experiment, simple conditional volatility models ARCH (1) and GARCH (1,1) models are fitted on S&P 500 Mini futures 1-minute returns from the year 2010 to 2013. The dataset has approximately 1 million data points. Some critical steps to the models followed in this research are:

1. Define a model. Import relevant ARCH python packages.
2. Import and split the data into train and test datasets.
3. Train the model on the training dataset.
4. Evaluate the model on test data.

5. Display and analyze output table of summarised statistics.
6. Perform variance forecasts using inbuilt libraries.
7. Note decision latency for further analysis.

ARCH models are commonly employed in modelling financial time series that exhibit time-varying volatility and volatility clustering, i.e. periods of swings interspersed with periods of relative calm.

ARCH (1)

The simplest model of the ARCH family is ARCH (1). The distribution of the returns for period t , conditional on all previous returns, is normal with constant mean μ and time-varying conditional variance h_t defined by

$$r_t | r_{t-1}, r_{t-2}, \dots \sim N(\mu, h_t)$$

and

$$h_t = \omega + \alpha(r_{t-1} - \mu)^2 \tag{5.1}$$

The volatility parameters are $\omega > 0$ and $\alpha = 0$. The volatility of the return in period t then depends solely on the previous return. The residual at time t is:

$$e_t = r_t - \mu$$

The summary of the model output is shown in Table 5.7.

```

%pyspark
model_arch_1 = arch_model(dfp['returns'][1:],vol = "ARCH",p=1)
results_arch_1 = model_arch_1.fit()
results_arch_1.summary()

```

```

Iteration:      1,  Func. Count:      5,  Neg. LLF: 7186768.560987371
Iteration:      2,  Func. Count:     15,  Neg. LLF: 1266648.0285784535
Iteration:      3,  Func. Count:     22,  Neg. LLF: 415843.14435076265
Iteration:      4,  Func. Count:     27,  Neg. LLF: 427242.12449352903
Iteration:      5,  Func. Count:     33,  Neg. LLF: 398709.89063607773
Iteration:      6,  Func. Count:     37,  Neg. LLF: 398709.83819004823
Iteration:      7,  Func. Count:     41,  Neg. LLF: 398709.83808913274
Iteration:      8,  Func. Count:     44,  Neg. LLF: 398709.83808913303
Optimization terminated successfully (Exit mode 0)
      Current function value: 398709.83808913274
      Iterations: 8
      Function evaluations: 44
      Gradient evaluations: 8
<class 'statsmodels.iolib.summary.Summary'>
"""

```

Constant Mean - ARCH Model Results			
Dep. Variable:	returns	R-squared:	-0.000
Mean Model:	Constant Mean	Adj. R-squared:	-0.000
Vol Model:	ARCH	Log-Likelihood:	-398710.
Distribution:	Normal	AIC:	797426.
Method:	Maximum Likelihood	BIC:	797461.
		No. Observations:	1048574
Date:	Fri, Sep 04 2020	Df Residuals:	1048571
Time:	22:30:42	Df Model:	3

```

      Mean Model
=====
      coef  std err      t  P>|t|  95.0% Conf. Int.
-----
mu      3.7737e-04  4.722e-04   0.799   0.424  [-5.481e-04,1.303e-03]
      Volatility Model
=====
      coef  std err      t  P>|t|  95.0% Conf. Int.
-----
omega    0.0872  1.019e-03  85.607   0.000  [8.520e-02,8.920e-02]
alpha[1] 0.4981  1.043e-02  47.735   0.000  [ 0.478, 0.519]
=====
Covariance estimator: robust

```

Figure 5.7: ARCH Model Summary Statistics

The value of the mean model as constant signifies mean is constant rather than moving. In other words, the mean value does not depend on its past values. Simple ARCH (1) model is used here for modelling variance. The distribution of the residuals from the

fitted model follows a normal distribution. The Maximum Likelihood method is used to find model coefficients. If multiple models need to be compared, then log-likelihood and information criteria values can be used. The value of Adj. R-squared is -0.000 means; the absolute value is so very minute; it rounds it to 0. R-squared is a measurement of the explanatory variation away from the mean. In the mean model, residuals are simply the version of the original dataset, where every value is decreased by a constant, then there will be no actual variance. Therefore, the r squared value is zero.

Log-Likelihood value is a measure of goodness of fit for any model. The higher the value, the better is the model. For a better ARCH model, the log-likelihood value should increase, and information criteria (AIC, BIC) values should decrease.

The *coef* column contains the numeric values of the model parameters such as α and ω in equation 5.1. The values in column *std err* explains how far away, on average, the model predictions are from the true values. Column *t* gives values for the test-statistic for model significance. p values for both constants ω and one lag value α are zero, which means both coefficients are significantly different from zero. The last two columns represent the critical value for the 95% confidence interval, if these columns do not contain any zero value, means again that coefficients are significant from zero. This means that volatility at time t is dependent on previous lagged value. The value of Df Model as three just signifies the number of numeric coefficients the model is trying to estimate. These are μ , ω , α .

From the summary Table 5.7, mean is resulted in constant, which means mean is not dependent on past lagged periods. However, in a case where it is time-dependent or changing, mean can be modelled using AR (Autoregressive Equation) or ARMA (Autoregressive Moving Average) equation. Different probability distributions for residuals can be checked by giving input parameter *dist* as students t , or log-normal etc.

$p < 0.05 \rightarrow$ lags are significant different and can be used for prediction.

$p > 0.05 \rightarrow$ Insignificant to predict using this lag.

Computation behind ARCH(1)

As can be seen from Table 5.7 the ARCH (1) model takes eight iterations before it

converges. This is due to fitting two equations at the same time, one of mean and another of variance. It works by fitting the model with certain coefficients, then check how well it has performed. Afterwards, it adjusts the coefficients before moving to the next iteration. The program stops when the log-likelihood decreases; however, in this case, the log-likelihood values are negative; therefore, it has to increase till model converges finding the best model coefficients. Due to the inbuilt feature of the ARCH module in Python, the program itself stops till it finds optimal parameters of the model. For a more complicated model, this program can take numerous attempts.

GARCH (1,1) – Generalised Autoregressive Conditional Heteroskedasticity

The ARCH model allowed the conditional variance to change over time as a function of past errors leaving the unconditional variance constant. An extension of this approach named GARCH or Generalized Autoregressive Conditional Heteroskedasticity incorporates a moving average component together with the autoregressive component. The introduction of a moving average component allows analysts to both model the conditional change in variance over time as well as changes in the time-dependent variance. The extension of the ARCH process to the GARCH process bears much resemblance to the extension of the standard time series AR process to the general ARMA process. The GARCH (1, 1) model with conditional normal distributions is the most popular ARCH specification in empirical research, mainly when modelling daily returns. Here variance is considered as the measure of volatility. The GARCH (1,1) model is built on E-Mini Future returns from the year 2010 to 2013.

The distribution of the return for period t , conditional on all previous returns, is defined by:

$$r_t | r_{t-1}, r_{t-2}, \dots \sim N(\mu, h_t)$$

and

$$h_t = \omega + \alpha(r_{t-1} - \mu)^2 + \beta h_{t-1} \quad (5.2)$$

$h_t = \text{Var}(y_t | y_{t-1})$ = The variance at time period t is conditional on values of the variable at time period $t - 1$.

ω = Constant term of the model.

α = Numeric coefficient for the squared residual for the past period.

$\epsilon_{t-1}^2 = (r_{t-1} - \mu)^2$ = Squared residual for the past period.

β = Numeric coefficient for the conditional variance from last period.

h_{t-1} = Conditional variance from the last period.

GARCH (p, q) with p as one and q as one is GARCH (1,1). The p refers to ARCH order (past ϵ_t^2 -Autoregressive), and q refers to GARCH order (past σ_t^2 - Moving Average (MA)).

GARCH (1,1)

```

%pyspark
mod_garch_1 = arch_model(dfp.returns[1:], vol = "GARCH", p = 1, q=1)
results_garch_1 = mod_garch_1.fit()
results_garch_1.summary()

Iteration:      1,  Func. Count:      6,  Neg. LLF: 208346693.23016724
Iteration:      2,  Func. Count:     18,  Neg. LLF: 239110.22940731928
Optimization terminated successfully (Exit mode 0)
      Current function value: 239110.22675517012
      Iterations: 6
      Function evaluations: 18
      Gradient evaluations: 2
<class 'statsmodels.iolib.summary.Summary'>
"""
                Constant Mean - GARCH Model Results
=====
Dep. Variable:          returns  R-squared:                -0.000
Mean Model:            Constant Mean  Adj. R-squared:          -0.000
Vol Model:             GARCH         Log-Likelihood:         -239110.
Distribution:          Normal        AIC:                    478228.
Distribution:          Normal        AIC:                    478228.
Method:               Maximum Likelihood  BIC:                    478276.
Date:                 Fri, Sep 04 2020  No. Observations:      1048574
                                Df Residuals:                1048570

Time:                 22:30:59  Df Model:                4
                                Mean Model

=====
                coef  std err      t      P>|t|      95.0% Conf. Int.
-----
mu              4.6256e-04  2.687e-04      1.722  8.513e-02  [-6.402e-05,9.891e-04]
                Volatility Model
=====
                coef  std err      t      P>|t|      95.0% Conf. Int.
-----
omega           3.0882e-03  2.057e-04     15.010  6.342e-51  [2.685e-03,3.491e-03]
alpha[1]        0.1000  3.082e-03     32.446  6.084e-231  [9.396e-02, 0.106]
beta[1]         0.8800  4.167e-03     211.186  0.000      [ 0.872, 0.888]
=====
Covariance estimator: robust
"""

```

Figure 5.8: GARCH (1,1) Model Output

Evaluation of GARCH (1,1)

The figure 5.8 depicts that the model only six iterations to converge. Mean is considered constant in this model. Variance residuals follow a normal distribution. Maximum-Likelihood methods estimate model coefficients. Log-Likelihood value of -239110 is greater than the Log-Likelihood value of ARCH 1 Model, which was -398710. Also, information criteria value (AIC) for GARCH (1,1) is higher than the ARCH (1). The value of Df Model as 4 represents the estimation of 4 coefficients, i.e. μ, ω, α and β . β is the key to volatility in GARCH model. P value of less than 0.05 signifies the significance of this model. These all parameters signifies that the GARCH (1,1) is a better model than the ARCH (1).

Adding a single past variance in GARCH model gives more predictive power than squared residuals of 1 lag ago in ARCH model. Hence including past values as a form of baseline provides much greater accuracy. All the coefficients ω, α, β are significant with p values less than 0.05. Hence GARCH (1,1) model is a front runner for measuring volatility. It has already been proven that no higher-lag GARCH models outperform the GARCH (1,1) when it comes to variance of market returns. As all the effects of the conditional variance two or more periods ago will be contained in the conditional variance of 1 period ago. So, there is no need to include more than 1 GARCH component.

Computation behind GARCH (1,1)

As can be seen from figure 5.8 the GARCH (1,1) model took six iterations before it converges. This is due to fitting two equations at the same time, one mean and another variance. It works by fitting the model with certain coefficients, then check how well it has performed. Afterwards, it adjusts the coefficients before moving to the next iteration. The program stops when the Log-Likelihood decreases; however, in this case, the log-likelihood values are negative; therefore, it has to increase till model converges finding the best model coefficients. Due to the inbuilt feature of GARCH module in Python, the program itself stops till it finds optimal parameters of the model. For more complicated models, this program can take numerous attempts. This is the reason ARCH type models take more time in computing volatility than simple calculations such as historical volatility. There

is a significant difference in decision latency observed in calculating summary statistics in experiment 1 and building ARCH type models in experiment 2.

Forecasts through GARCH (1,1)

In order to test for the validity of the variance forecasts when using GARCH models, it should be made sure that the model adequately captures the dynamics of the data. Now though the focus of this research is not to dive into the evaluations of the GARCH models forecasts, inbuilt residual variance forecast module in Python has helped me to take into account the decision latency of the execution in forecasting next period volatility. This GARCH (1,1) model trained previously produces one period ahead forecasts of variance.

One of the most exciting papers, co-authored by (Andersen and Bollerslev, 1998) himself, attempts to combat the argument that, while GARCH models often seem to fit well in-sample, they have poor forecasting performance at an intraday level. He argues that the ex-post measure of variance commonly being used, the daily squared return, is a poor estimate of the actual variance that one should measure performance against. Instead, one should use a summation of squared intraday returns aggregated to the daily level, termed realised variance, as a measure of that day's variance. These are then compared against the performance measures of realised variance. Aggregation methods are found to be more accurate than 1-step ahead intraday forecasts.

```

Forecasting 1 Period Ahead Volatility

%pyspark
pred_garch = results_arch_1.forecast(horizon=1)
dff = pred_garch.residual_variance[start_date:end_date]
dff['standardized_residuals'] = (dff['h.1'] - dff['h.1'].mean())/dff['h.1'].std()

%pyspark
dff

          h.1  standardized_residuals
datetime
2011-12-01 09:30:00  0.140526          -0.086257
2011-12-01 09:31:00  0.147622          -0.038406
2011-12-01 09:32:00  0.138341          -0.100996
2011-12-01 09:33:00  0.130291          -0.155279
2011-12-01 09:34:00  0.118063          -0.237738
...
2011-12-30 16:11:00  0.217196           0.430766
2011-12-30 16:12:00  0.318585           1.114491
2011-12-30 16:13:00  0.328119           1.178786
2011-12-30 16:14:00  0.291534           0.932075
2011-12-30 16:15:00  0.279288           0.849489
[27061 rows x 2 columns]

Took 0 sec. Last updated by anonymous at September 04 2020, 11:42:41 PM.

```

Figure 5.9: GARCH (1,1) Residual Variance Forecast of 1 minute ahead

For 1-minute sampling, the forecasting procedure is:

- 1) An initial GARCH (1,1) model is fit using the first 3,23,532 1-minute returns from date 2011-01-01 to 2011-11-31. This ends the fit on 2011-11-31 06:30:00, the end of that day.
- 2) The next 30 days worth of 1-minute variance forecasts are generated recursively.
- 3) Those 1-minute intraday forecasts are summed at a daily level to generate a prediction of variance for that day.
- 4) Steps 2-3 are repeated to generate daily forecasts of variance for December in the year 2011. More accurate forecast using GARCH model is a daily forecast aggregated 1-minute returns. These forecasts are compared against those generated from models fit to daily data in Figure 5.10.

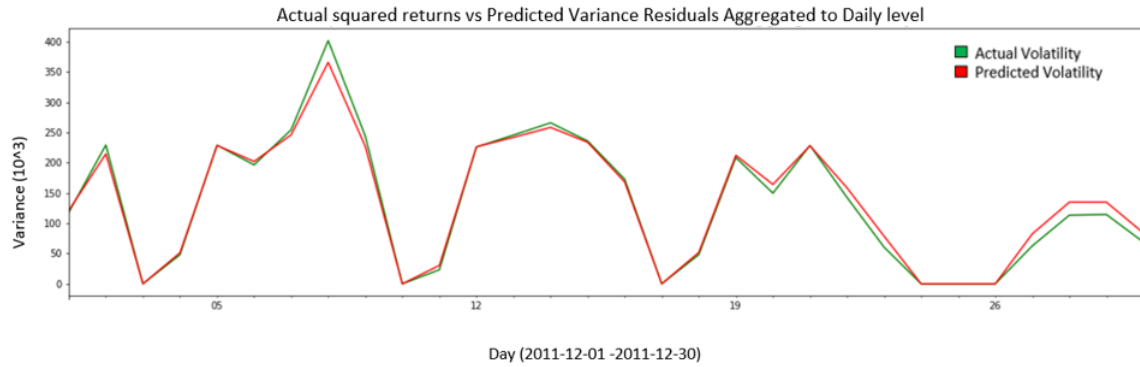


Figure 5.10: Actual vs Predicted Daily Volatility. Summation of squared intraday returns aggregated to the daily level, termed realised variance, as a measure of that day’s variance is more accurate than 1-minute ahead intraday forecasts

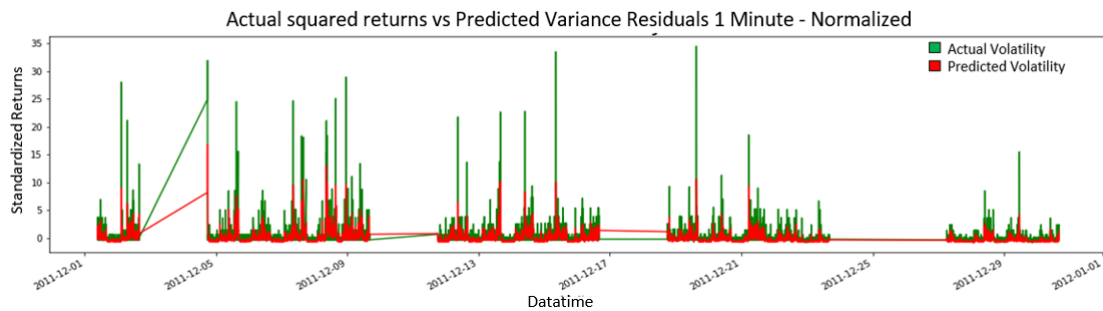


Figure 5.11: Actual vs Predicted 1-minute volatility forecast. 1 Minute variance forecast results in noisy measurement. The reason for this is that the GARCH takes squared returns into consideration while modelling volatility. $R_t = \sigma_t \cdot z_t$ where z_t denotes an independent mean zero and t denotes latent volatility. Squared returns cause noisy measurements due to the idiosyncratic error term z_t^2 .

5.4.2 Cluster Estimation based on Decision Latency

Due to the iterations, the GARCH (1,1) model takes before it converges to find the optimal model parameters, the high computation time is observed. The decision latency here is more than the one observed in computing summary statistics in experiment 1. Table 5.7 lists the execution time noticed while computing ARCH (1) and GARCH (1,1) models. The computation is performed on a different range of data points as was in the experiment 1.

Table 5.7: Decision Latency table for experiment 2 on different scales of data sizes and clusters.

ARCH and GARCH			
Number of Data points (Binary) Range - (65k - 3.4 Million)	Number of Clusters	Execution Time	Speed Improvement (%) From Clusters (2-4, 4-8, 2-8)
65536	2	6.43	4.05%
	4	6.18	31.77%
	8	4.69	37.10%
131072	2	9.25	4.30%
	4	8.87	15.38%
	8	7.69	20.34%
262144	2	15.16	5.67%
	4	14.35	15.11%
	8	12.47	21.63%
524288	2	35.846	28.60%
	4	27.875	17.87%
	8	23.648	51.58%
1048576	2	42.99	5.41%
	4	40.79	21.87%
	8	31.86	34.92%
2097152	2	110.3	4.25%
	4	105.8	19.68%
	8	84.98	29.80%
3478488	2	Out of Memory Error	NA
	4	Out of Memory Error	NA
	8	142.65	142.65

Summarizing the first case in Table 5.7 where the number of datapoints taken for computing ARCH (1) and GARCH (1,1) are 65536. The execution time for computing on 2 clusters is 6.43 seconds, on 4 clusters is 6.18 seconds, and on 8 clusters is 4.69 seconds. The improvement in execution time moving from 2 clusters to 4 clusters is 4.05%, 4 clusters to 8 clusters is 31.77%, and 2 clusters to 8 clusters is 37.10%. Similar results are self-explanatory in Table 5.7.

Common Observations

From Table 5.7 it is evident that the execution time in computing ARCH (1) and GARCH (1,1) decreases with an increase in the size of clusters. The highest improvement in performance is observed on moving from 2 to 8 clusters. It is clear that decision latency increases with increase in size of the dataset. Also the execution time decreases with an increase in the size of clusters.

In a 1-minute trading environment, if minimum of 30 seconds is needed as a buffer to take trading decision, and the dataset has more than 1 million data points, it is impossible to model volatility on 8 or less than 8 clusters. In that case, more than 8 clusters need to be adapted by the trading firm. In the case when execution is shifted from 4 clusters to 8 clusters, It is observed that the speed at which the performance in execution is improved is decreasing with increase in the size of the input data.

Note that the execution time for modelling volatility on 3.4 million datapoints resulted in out of memory error on 2 and 4 clusters. The execution time here on 8 clusters is 142.65 seconds.

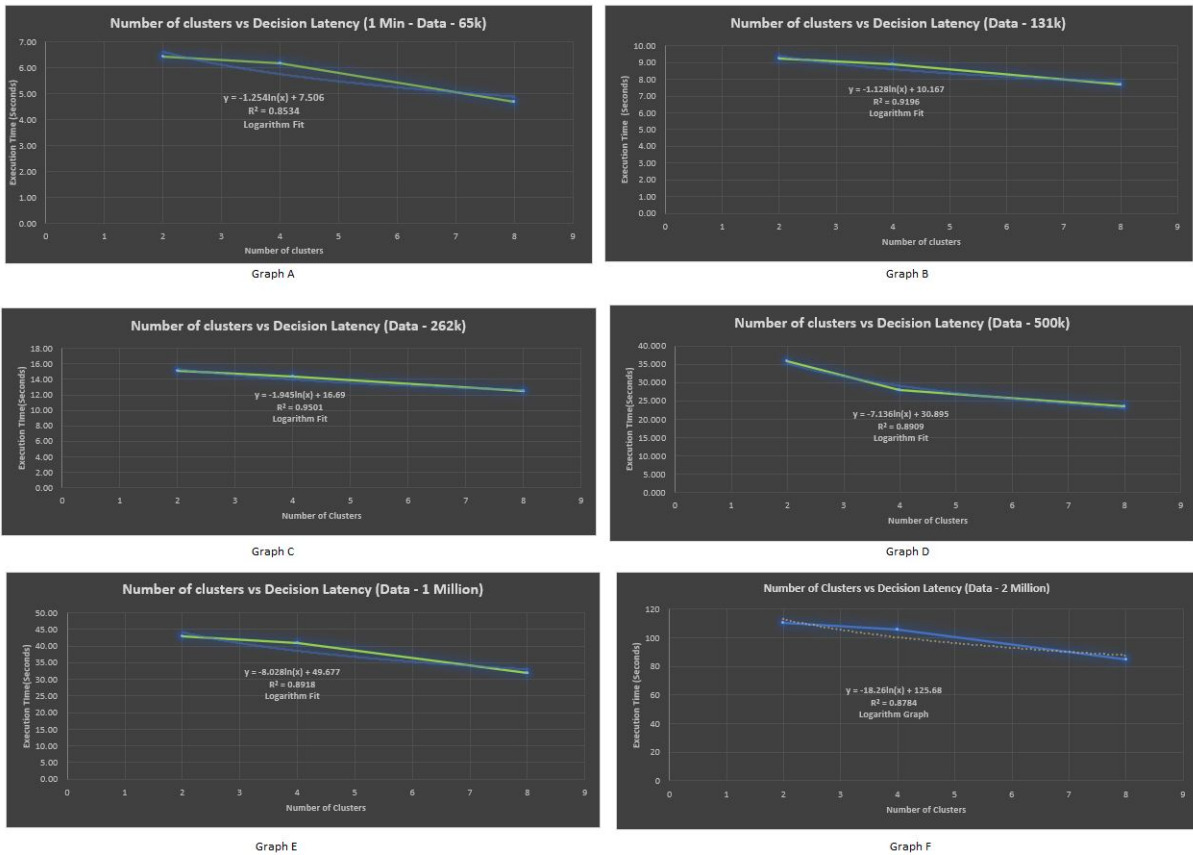


Figure 5.12: Graphs(A-F) showing the relation between decision latency and data points on each given size of a cluster. The logarithm fit curve shows that a saturation point will be reached as the latency reduces further as it cannot pass beyond a certain threshold level.

Also due to the movement of latency which follows the path of logarithm curve (see Figure 5.12), it is not necessary that with increase in cluster size from 8 to 16 the decision latency will be reduced extensively. Logarithmic scale charts can help show the bigger picture. On increasing the size of cluster will the curve will lead towards a saturation point post which there cannot be more improvement in performance.

5.4.3 Change in Decision Latency based on Data Points

The first column from Table 5.8 shows the different sizes of data points (as the power of 2) taken as input to heavy time-series volatility models. The second, third, and fourth columns provide the execution time on each of the 2,4 and 8 clusters. It is clear from Table

5.8 and Figure 5.13 that with the increase in the size of the data, the computation time increases. This behaviour is the same with all different sizes of clusters. The table helps to decide how much historical data should be considered given the decision latency. For example, if a trader wants to have 40 seconds buffer to decide a 1-minute high frequency trading environment and has a dataset of even 500k data points, he/she has to use more than 8 clusters for computation.

It is also observed that as we reduce the size of the dataset, the improvement in decision latency decreases. For instance, computation time on 2 million datapoints is 110.30 seconds on 2 clusters, 105.80 seconds on 4 clusters, and 77.98 seconds on 8 clusters. On moving to 1 million data points, decision latency is reduced by an average of 153.58%. On the other hand, when we move from 1 million data points to nearly 500,000 data points, the computation time only decreases by an average of 33.67%.

Table 5.8: Decision latency on a different scale of data inputs. Every time the data size is doubled execution time increases by a definite percentage.

Execution Time (Seconds)			
Data Points	2 Clusters	4 Clusters	8 Clusters
65536	6.429	6.183	4.900
131072	9.252	8.870	6.688
262144	15.164	14.351	13.467
524288	35.846	27.875	23.648
1048576	42.993	40.786	31.865
2097152	110.305	105.808	77.988
3478488	Out of Memory Error	Out of Memory Error	142.6478362

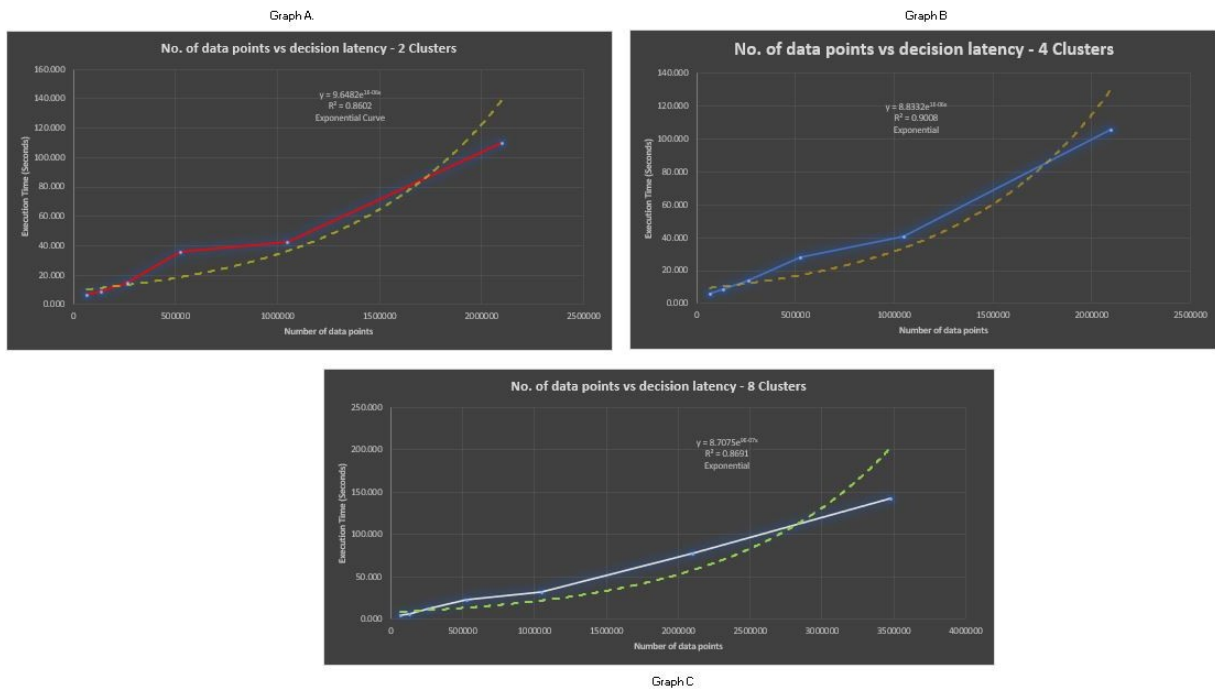


Figure 5.13: Graph(A-C) showing the relation between decision latency and data points on each given cluster. On a logarithmic graph of decision latency, even though the size of data is increasing along with size of the clusters, a point is observed where the rate of growth starts to level off when that exponential growth has stopped (see Figure 5.12).

5.5 Combined Evaluation of Experiment 1 and 2

First it is clear from the output of two experiments in Table 5.9 that in a 1-minute trading environment, decision latency will be lower when calculating standard deviation as a measure of volatility then modelling it through time series volatility models.

As can be observed from Table 5.9, if we compute both descriptive statistics and volatility models, the total decision latency can be observed as the sum of each decision latency. This total time can help traders measure both historical and conditional volatility making the volatility prediction more accurate.

Table 5.9: Combined evaluation of execution time for both experiment 1 and experiment 2 based on Decision Latency. Example includes that with data size as 1 million or more it is not possible to compute both summary statistics and ARCH models within 1-minute interval. This might be possible on scaling cluster size from 8 to 16.

Number of Data Points (Power of 2)	Number of Clusters	Descriptive Statistics Execution Time (Seconds)	ARCH (1) and GARCH (1,1) Execution Time (Seconds)	Combined Execution Time (Seconds)
65536	2	4.91	6.43	11.34
	4	3.06	6.18	9.24
	8	2.68	4.69	7.37
131072	2	6.1	9.25	15.35
	4	4.39	8.87	13.26
	8	3.9	7.69	11.59
262144	2	7.09	15.16	22.25
	4	6.04	14.35	20.39
	8	5.88	12.47	18.35
524288	2	9	26.95	35.95
	4	8.86	19	27.86
	8	6.34	17.06	23.40
1048576	2	32.19	42.99	75.18
	4	20.33	40.79	61.12
	8	12.74	31.86	44.60
2097152	2	52.71	110.3	163.01
	4	39.87	105.8	145.67
	8	23.22	84.98	108.20

5.6 Status of Results

The objective of the research to estimate clusters based on decision latency in high frequency trading is met. This is a purely computational based study. The results look interesting as well as promising and can progress towards a publication under the supervision of Professor Khurshid Ahmed.

5.7 Conclusion

Adapting big data architecture in High Frequency Trading can help traders take into account a large volume of historical data along with real-time data in taking quick decisions, perhaps within 1 minute trading environment. This is only possible if there is a base architecture to decide how many clusters are needed on how much volume of data given the decision latency as a prerequisite.

The big data architecture provided in this research is a solution to real-time decision making problem using parallel computing framework. With the given results, traders can decide in real-time how many parallel computing machines or clusters they need or how much data input they can take into account given two possible scenarios of estimating volatility i.e. descriptive statistics and ARCH volatility models. As a prerequisite, they should have decision time in hand before deciding on clusters.

Chapter 6

Conclusion and Future Work

The remainder of this chapter will discuss the conclusion and contributions of this study (Chapter 6.1). A discussion of the limitations of the work is presented along with the potential future works (Section 6.2).

6.1 Conclusion

In financial literature people work with usually daily data where volatility is aggregated out. Small datasets are used so traders are not worried about latency. But they are using a very important source of information i.e. high frequency data, which is very relevant when we look at issues like volatility. In computer science literature people perform computations on time series that are not large but the computations largely deal with issues related to descriptive statistics. In one sense, we have looked at a complex computation (volatility) with large data sets to show the problems of latency and one solution suggested was using AWS Elastic MapReduce (EMR) based scalability.

This paper introduced a step-by-step practical guide for setting up a minimum working example of a distributed system for time series analysis and forecasting. The system is built-in Apache Spark and the integrated with AWS EMR clusters to perform parallel computations on large time-series datasets. A simple forecasting exercise illustrates that the parallelization scheme reduces total runtime performance substantially relative to a

single-machine setting. The presented approach requires minimal installation and configuration effort, and it can be implemented with little background in computer science and parallel programming.

In this research, the importance of latency in making a high-frequency trading decision in real-time is studied. Here I have implemented the existing theoretical business cloud architecture for high-frequency trading in addition to the usage of advanced Big Data tools such as AWS Elastic MapReduce (EMR) with Apache Spark. The computation involves estimating risk both using historical and real-time changing volatility.. The whole data is taken from the year 2010 to 2019, divided into a binary format from 65000 to 3.4 Million data points. This futures data can be taken as a reference for all other time series analysis because it has all the properties of a real problem a trader can face. The literature on volatility nature of futures data is discussed in section 4.3. This makes this series an exemplar time series.

Next, risk computation is performed on 2,4 and 8 clusters. Each scenario in experiment 1 and experiment 2 is executed five times to take into account changing execution time, and in the end, an average of the decision latency is noted. Finally, the relationship between decision latency and size of clusters based on a different range of datapoints is evaluated along with the relation between decision latency and the size of the dataset.

Significant improvement in decision time is observed in moving the computation from 2 to 8 when compared from 2 to 4 or 4 to 8 clusters. High-frequency traders don't often think about decision latency in calculating volatility due to the small size of the dataset. Even if the dataset is large, there is a computational gap which can be filled by secured big data cloud-based architecture. With the architecture developed in this research, traders can take real-time decisions based on their computing needs without depending a lot on their intuition. The implications of this research will help readers from banking and financial organization who aim to compute big data in financial markets, employees in the investment banking sector who realized the need to move their legacy trading system from on-premise to cloud, and most importantly, the HFT firms who wish to use parallel computing framework for trading but are bounded by the unavailability of implemented big data architecture. With further expansion, evaluation, and improvement of this big data architecture, traders will be able to decide the distributed computing machines or

clusters based on the data they have and the minimum execution time they require keeping in mind the cost to value ratio.

Limitations

Due to the limited features of the AWS student account, execution on 16 and 32 clusters could not be performed to observe the saturation point when the decision latency stops reducing further. Also, real-time streaming using AWS Kinesis could not be performed due to the access limitations on a student account. Till now, Spark does not provide packages for time series analysis volatility models such as ARCH and GARCH. Python library for Spark, i.e., PySpark or Scala, is way faster on distributed systems than original Python. Due to the unavailability of the GARCH packages in PySpark, in this research, some of the packages of Python are used for modelling volatility, which can add a few seconds latencies to the execution.

6.2 Future Work

Data sampled at a different minute or second frequencies can be considered for future research such as 5 minutes, 1 second etc. Evaluation of the volatility models can be studied and improved by readers in finance industries or econometricians doing their research in this domain. Execution time by implementing various other ARCH type models can be evaluated and compared under the same architecture. Real-time streaming through AWS Kinesis can be performed, which has not been possible in this research due to limited AWS credits. As a few seconds latency in extracting real-time streams of tick data can add to the overall decision latency, it is necessary to perform its implementation and evaluation. The development of security and management approaches are also subject to future work as different security, and access policies can be attached to the EMR clusters. Another issue that remains to be researched in this field is understanding how to make good use of high-performance GPU on cloud-based servers to speed up the whole computation process. The method of increasing computational power on a

standalone machine by scaling the capacity of GPU and CPU core is called Vertical Scaling. Similar architecture can be deployed on a single machine with high processing power in order to note decision latencies. Then the decision latencies on cloud and local machines can be compared and evaluated. The cost to value ratio is not evaluated in this research and can be key research further in the area of big data analytics in finance.

References

- Agopyan, Arden, Emrah Şener, and Ali Beklen (2010). *Financial Business Cloud for High-Frequency Trading* (cit. on p. 20).
- Aït-Sahalia, Yacine and Celso Brunetti (July 2020). “High frequency traders and the price process”. *Journal of Econometrics* 217, pp. 20–45. DOI: 10.1016/j.jeconom.2019.11.005 (cit. on p. 13).
- Aldridge, Irene (2013). *High-frequency trading : a practical guide to algorithmic strategies and trading systems*. Wiley (cit. on pp. 1, 11).
- Andersen, Torben G. and Tim Bollerslev (June 1997). “Intraday periodicity and volatility persistence in financial markets”. *Journal of Empirical Finance* 4, pp. 115–158. DOI: 10.1016/S0927-5398(97)00004-2. URL: <https://www.sciencedirect.com/science/article/pii/S0927539897000042> (cit. on pp. 17, 18).
- (Nov. 1998). “Answering the Skeptics: Yes, Standard Volatility Models do Provide Accurate Forecasts”. *International Economic Review* 39, p. 885. DOI: 10.2307/2527343 (cit. on pp. 2, 25, 47).
- Andersen, Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys (Dec. 2000). “Exchange Rate Returns Standardized by Realized Volatility are (Nearly) Gaussian”. *Multinational Finance Journal* 4, pp. 159–179. DOI: 10.17578/4-3/4-2 (cit. on p. 14).

Andersen, Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys (Mar. 2003). “Modeling and Forecasting Realized Volatility”. *Econometrica* 71, pp. 579–625. DOI: 10.1111/1468-0262.00418 (cit. on p. 14).

Andersen, Torben G., Oleg Bondarenko, Albert S. Kyle, and Anna A. Obizhaeva (Aug. 2018). *Intraday Trading Invariance in the E-Mini SP 500 Futures Market*. papers.ssrn.com. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2693810 (cit. on p. 55).

Angel, James J. (Apr. 2014). “When Finance Meets Physics: The Impact of the Speed of Light on Financial Markets and Their Regulation”. *Financial Review* 49, pp. 271–281. DOI: 10.1111/fire.12035 (cit. on p. 12).

Areal, Nelson M. P. C. and Stephen J. Taylor (May 2002). “The realized volatility of FTSE-100 futures prices”. *Journal of Futures Markets* 22, pp. 627–648. DOI: 10.1002/fut.10018 (cit. on p. 17).

BANDI, F. M. and J. R. RUSSELL (Apr. 2008). “Microstructure Noise, Realized Variance, and Optimal Sampling”. *Review of Economic Studies* 75, pp. 339–369. DOI: 10.1111/j.1467-937x.2008.00474.x (cit. on p. 14).

Baron, Matthew, Jonathan Brogaard, Björn Hagströmer, and Andrei Kirilenko (Sept. 2018). “Risk and Return in High-Frequency Trading”. *Journal of Financial and Quantitative Analysis* 54, pp. 993–1024. DOI: 10.1017/s0022109018001096 (cit. on pp. 2, 12, 13).

Biais, Bruno, Thierry Foucault, and Sophie Moinas (May 2015). “Equilibrium fast trading”. *Journal of Financial Economics* 116, pp. 292–313. DOI: 10.1016/j.jfineco.2015.03.004 (cit. on p. 12).

Big Data, Oracle Ireland (2020). Oracle.com. URL: <https://www.oracle.com/ie/big-data/what-is-big-data.html> (cit. on p. 3).

Bluhm, Benjamin (2018). “Time Series Econometrics at Scale: A Practical Guide to Parallel Computing in (Py)Spark”. *SSRN Electronic Journal*. DOI: 10.2139/ssrn.3226976 (cit. on p. 20).

Bollerslev, Tim (Apr. 1986). “Generalized autoregressive conditional heteroskedasticity”. *Journal of Econometrics* 31, pp. 307–327. DOI: 10.1016/0304-4076(86)90063-1 (cit. on p. 49).

Bollerslev, Tim, Ray Y. Chou, and Kenneth F. Kroner (Apr. 1992). “ARCH modeling in finance”. *Journal of Econometrics* 52, pp. 5–59. DOI: 10.1016/0304-4076(92)90064-x (cit. on p. 15).

Brogaard, Jonathan, Terrence Hendershott, Stefan Hunt, and Carla Ysusi (Apr. 2014). “High-Frequency Trading and the Execution Costs of Institutional Investors”. *Financial Review* 49, pp. 345–369. DOI: 10.1111/fire.12039 (cit. on p. 12).

Brogaard, Jonathan and Kevin Roshak (2015). “Prices and Price Limits”. *SSRN Electronic Journal*. DOI: 10.2139/ssrn.2667104 (cit. on p. 13).

CHABOUD, ALAIN P., BENJAMIN CHIQUOINE, ERIK HJALMARSSON, and CLARA VEGA (Sept. 2014). “Rise of the Machines: Algorithmic Trading in the Foreign Exchange Market”. *The Journal of Finance* 69, pp. 2045–2084. DOI: 10.1111/jofi.12186 (cit. on p. 12).

Chebbi, I, W Boulila, N Mellouli, M Lamolle, and I Farah (2018). *A Comparison of Big Remote Sensing Data Processing with Hadoop MapReduce and Spark* (cit. on p. 20).

Dacorogna, M.M., U.A. Muller, R. Olsen, and O.V. Pictet (2001). “An Introduction to High-Frequency Finance”. *Academic Press: San Diego, CA* (cit. on pp. 14, 15).

De Luca, Giovanni (2006). “FORECASTING VOLATILITY USING HIGH-FREQUENCY DATA”. *Statistica Applicata* 18 (cit. on p. 1).

- Engle, Robert F. (July 1982). “Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation”. *Econometrica* 50, p. 987. DOI: 10.2307/1912773 (cit. on pp. 24, 48).
- (Jan. 2000). “The Econometrics of Ultra-high-frequency Data”. *Econometrica* 68, pp. 1–22. DOI: 10.1111/1468-0262.00091 (cit. on p. 14).
- Figlewski, Stephen (Feb. 1997). “Forecasting Volatility”. *Financial Markets, Institutions and Instruments* 6, pp. 1–88. DOI: 10.1111/1468-0416.00009 (cit. on p. 25).
- FOUCAULT, THIERRY, JOHAN HOMBERT, and IOANID ROȘU (Jan. 2016). “News Trading and Speed”. *The Journal of Finance* 71, pp. 335–382. DOI: 10.1111/jofi.12302 (cit. on p. 12).
- France, Trav. (1792). *The Library of Trinity College Dublin - Off Campus Access to e-Resources* -. login.elib.tcd.ie. URL: <https://www-oed-com.elib.tcd.ie/view/Entry/224401?redirectedFrom=volatility#eid> (cit. on p. 2).
- Gartner (2012). *Big Data*. Gartner. URL: <https://www.gartner.com/en/information-technology/glossary/big-data> (cit. on p. 3).
- Goldstein, Michael A., Pavitra Kumar, and Frank C. Graves (Apr. 2014). “Computerized and High-Frequency Trading”. *Financial Review* 49, pp. 177–202. DOI: 10.1111/fire.12031 (cit. on pp. 11, 12).
- Goodhart, C.A.E and M. O’Hara (1997). *Goodhart And O’hara-High Frequency Data In Financial Markets Issues And Applications*. calameo.com. URL: <https://en.calameo.com/read/00031869140022d425a8d> (cit. on p. 14).
- Goodhart, C.A.E. and L. Figliuoli (Mar. 1991). “Every minute counts in financial markets”. *Journal of International Money and Finance* 10, pp. 23–52. DOI: 10.1016/0261-5606(91)90025-f (cit. on p. 17).

Gourieroux, Christian and Joann Jasiak (Dec. 2001). *Financial Econometrics: Problems, Models, and Methods*. Princeton University Press (cit. on p. 14).

Guardian (Sept. 2008). *The Library of Trinity College Dublin - Off Campus Access to e-Resources* -. login.elib.tcd.ie. URL: <https://www-oed-com.elib.tcd.ie/view/Entry/34689?redirectedFrom=cloud+computing#eid189443962> (cit. on p. 4).

Hasbrouck, Joel and Gideon Saar (Nov. 2013). “Low-latency trading”. *Journal of Financial Markets* 16, pp. 646–679. DOI: 10.1016/j.finmar.2013.05.003. URL: <http://people.stern.nyu.edu/jhasbrou/Research/LowLatencyTradingJFM.pdf> (cit. on p. 12).

Hoffmann, Peter (July 2014). “A dynamic limit order market with fast and slow traders”. *Journal of Financial Economics* 113, pp. 156–169. DOI: 10.1016/j.jfineco.2014.04.002 (cit. on pp. 13, 25).

Huttunen, Jennifer, Jaana Jauhiainen, Laura Lehti, Annina Nylund, Minna Martikainen, and Othmar Lehner (2019). “BIG DATA, CLOUD COMPUTING AND DATA SCIENCE APPLICATIONS IN FINANCE AND ACCOUNTING”. *ACRN Journal of Finance and Risk Perspectives* 8, pp. 16–30 (cit. on p. 4).

Jiang, Wei, Qingsong Ruan, Jianfeng Li, and Ye Li (June 2018). “Modeling returns volatility: Realized GARCH incorporating realized risk measure”. *Physica A: Statistical Mechanics and its Applications* 500, pp. 249–258. DOI: 10.1016/j.physa.2018.02.018 (cit. on p. 55).

JORION, PHILIPPE (June 1995). “Predicting Volatility in the Foreign Exchange Market”. *The Journal of Finance* 50, pp. 507–528. DOI: 10.1111/j.1540-6261.1995.tb04793.x (cit. on p. 25).

Khurshid, Ahmad (Aug. 2019). *A note on high-frequency and big data modelling and analysis* (cit. on p. 19).

- Kirilenko, Andrei A., Albert S. Kyle, Mehrdad Samadi, and Tugkan Tuzun (2011). “The Flash Crash: The Impact of High Frequency Trading on an Electronic Market”. *SSRN Electronic Journal*. DOI: 10.2139/ssrn.1686004 (cit. on p. 11).
- Kurov, Alexander and Dennis J. Lasser (2004). “Price Dynamics in the Regular and E-Mini Futures Markets”. *The Journal of Financial and Quantitative Analysis* 39, pp. 365–384 (cit. on p. 55).
- Ladley, Daniel (July 2020). “The high frequency trade off between speed and sophistication”. *Journal of Economic Dynamics and Control* 116, p. 103912. DOI: 10.1016/j.jedc.2020.103912 (cit. on p. 11).
- Ladokhin, Sergiy (July 2009). “Volatility modeling in financial markets”. PhD thesis (cit. on pp. 2, 22).
- Laughlin, Gregory, Anthony Aguirre, and Joseph Grundfest (Apr. 2014). “Information Transmission between Financial Markets in Chicago and New York”. *Financial Review* 49, pp. 283–312. DOI: 10.1111/fire.12036 (cit. on p. 12).
- MacDonald, A. (2007). “LSE leads race for quicker trades”. *The Wall Street Journal Europe, June 19* (cit. on p. 12).
- Marinelli, C., S.T. Rachev, and R. Roll (Nov. 2001). “Subordinated exchange rate models: evidence for heavy tailed distributions and long-range dependence”. *Mathematical and Computer Modelling* 34, pp. 955–1001. DOI: 10.1016/s0895-7177(01)00113-3 (cit. on p. 16).
- Martens, Martin P.E., Yuan-Chen Chang, and Stephen J. Taylor (2001). “A Comparison of Seasonal Adjustment Methods When Forecasting Intraday Volatility”. *SSRN Electronic Journal*. DOI: 10.2139/ssrn.283030 (cit. on p. 17).

Merton, Robert C. (Dec. 1980). “On estimating the expected return on the market”. *Journal of Financial Economics* 8, pp. 323–361. DOI: 10.1016/0304-405x(80)90007-0 (cit. on p. 22).

Mitnik, Stefan, Marc S. Paoletta, and Svetlozar T. Rachev (Jan. 2002). “Stationarity of stable power-GARCH processes”. *Journal of Econometrics* 106, pp. 97–107. DOI: 10.1016/s0304-4076(01)00089-6 (cit. on p. 16).

O’Driscoll, Aisling, Jurate Daugelaite, and Roy D. Sleator (Oct. 2013). “‘Big data’, Hadoop and cloud computing in genomics”. *Journal of Biomedical Informatics* 46, pp. 774–781. DOI: 10.1016/j.jbi.2013.07.001 (cit. on p. 5).

Oxford Dictionary (1928). URL: <https://www-oed-com.elib.tcd.ie/> (cit. on p. 24).

Peng, Zhihao (Jan. 2019). “Stocks Analysis and Prediction Using Big Data Analytics”. *2019 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*. DOI: 10.1109/icitbs.2019.00081 (cit. on p. 3).

Shamgar, Ido (Mar. 2014). *Smart Trading: Adding Precision to Big Data-Inspired Trading Strategies / SAP Blogs*. blogs.sap.com. URL: <https://blogs.sap.com/2014/03/25/smart-trading-adding-precision-to-big-data-inspired-trading-strategies/> (cit. on p. 1).

Shapiro, Carl and Hal R Varian (2013). *Information rules : a strategic guide to the network economy*. Harvard Business School Press (cit. on p. 19).

Sun, M (2007). *Quantitative Methods in High-Frequency Financial Econometrics: Modeling Univariate and Multivariate Time Series* (cit. on p. 16).

Sun, Wei, Svetlozar Rachev, and Frank J. Fabozzi (Nov. 2007). “Fractals or I.I.D.: Evidence of long-range dependence and heavy tailedness from modeling German equity market returns”. *Journal of Economics and Business* 59, pp. 575–595. DOI: 10.1016/j.jeconbus.2007.02.001 (cit. on p. 15).

- Taylor, Stephen J (2007). *Asset price dynamics, volatility, and prediction*. Princeton University Press, , Cop (cit. on pp. [2](#), [13](#), [16](#), [22](#), [43](#), [47](#), [77](#)).
- Tian, Xinhui, Rui Han, Lei Wang, Gang Lu, and Jianfeng Zhan (Dec. 2015). “Latency critical big data computing in finance”. *The Journal of Finance and Data Science* 1, pp. 33–41. DOI: [10.1016/j.jfds.2015.07.002](https://doi.org/10.1016/j.jfds.2015.07.002) (cit. on pp. [18](#), [19](#)).
- Varia, Jinesh and Sajee Mathew (2014). *Amazon Web Services -Overview of Amazon Web Services*. URL: http://cabibbo.dia.uniroma3.it/asw-2014-2015/altrui/AWS_Overview.pdf (cit. on p. [5](#)).
- Varian, Hal R, Joseph Farrell, and Carl Shapiro (2004). *The economics of information technology : an introduction*. Cambridge University Press (cit. on p. [19](#)).
- Verma, Ankush, Ashik Mansuri, and Neelesh Jain (2016). *Big Data Management Processing with Hadoop MapReduce and Spark Technology: A Comparison* (cit. on p. [20](#)).
- Wermers, Russ and Jinming Xue (2015). *Intraday ETF Trading and the Volatility of the Underlying A Research Paper Prepared for Lyxor Asset Management* (cit. on p. [54](#)).
- WOOD, ROBERT A., THOMAS H. McINISH, and J. KEITH ORD (July 1985). “An Investigation of Transactions Data for NYSE Stocks”. *The Journal of Finance* 40, pp. 723–739. DOI: [10.1111/j.1540-6261.1985.tb04996.x](https://doi.org/10.1111/j.1540-6261.1985.tb04996.x). URL: <http://onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.1985.tb04996.x/full> (cit. on pp. [16](#), [44](#)).
- Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, Michael Franklin, Scott Shenker, and Ion Stoica (2012). *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing* (cit. on p. [19](#)).