Impact of Low-Latency Architecture on High-Frequency Big Data

Sneha Srivastava

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Dr Khurshid Ahmad

August 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Sneha Srivastava

September 11, 2020

Permission to Lend and/or Copy

Acknowledgments

I would like to express my immense gratitude towards my supervisor Professor Khurshid Ahmad without whose compassion, guidance, insights and advice, this research work had not been possible. He supported me every step of the way and kept me motivated in these very unusual circumstances.

I would also like to express heartfelt gratitude towards my parents Mrs. Madhu Srivastava, Mr. Sudhir Kumar Srivastava and my sister Sonal, without whose love and support, I wouldn't be where I am today. A special thanks to my friends V Anand Shankar, and Divyanshu Marwah for their help, suggestions and feedback.

Sneha Srivastava

University of Dublin, Trinity College
August 2020

Impact of Low-Latency Architecture on High-Frequency Big Data

Sneha Srivastava, Master of Science in Computer Science

University of Dublin, Trinity College, 2020

Supervisor: Dr Khurshid Ahmad

ABSTRACT

Finance was among one of the earliest domains to deal with Big Data. Stock market traders work in "millisecond environment" where financial decisions need to be made in milliseconds to be able to deal with the innate volatility of the stock market. With the ever-increasing velocity, frequency and volume of big data in finance due to rapidly rising use of algorithmic trading, a specialised low-latency computing architecture is needed to be able to store, process and extract information from this data with minimal decision latency. Currently, stock traders make use of physical, local computing systems and geographical closeness to the trading venue to reduce the latency of their financial decision-making.

Big Data technologies are used to provide distributed computing and parallel processing capabilities. They provide a way to process huge quantities of data efficiently, increase responsiveness to the changes in the data and are scalable.

Through this research, I aimed to address the gap between real-time data processing and market risk estimation of financial high frequency big data and big data architecture. I evaluated the impact on latency for market risk estimation of 3 high frequency stock market datasets. I used statistical methods with financial market algorithms in conjunction with Big Data architecture - Spark and Hadoop, for this study. The latency study showed that Hadoop Map-Reduce and Spark both outperformed the baseline system in decision latency and resources utilization by 40.6% and 20% respectively while Hadoop outperformed Spark in each of these aspects on all datasets. The market study showed that over a period, equity firms start behaving like the overall market.

Summary

The major objectives of this research are two-fold. Firstly, to use the parallel processing capabilities of big data architecture in financial market's risk estimation, using high frequency big data, and study its impact on the decision latency. Secondly, to compare the market risk of a specific sector, i.e., US airlines sector, with the overall market risk.

My research suggests that current low-latency computing solutions in finance use physical machines near stock market trading venues in order to reduce this decision latency. Usage of distributed big data processing systems which allows quick retrieval of large historical data and parallel high-speed computation addresses the issue of low-latency and real time decision-making in finance.

I have used publicly available high frequency intraday, 1-minute stock market big data for 3 US equity firms and low frequency data for a US stock market index. The problems being addressed here are real-time data processing and market risk estimation. I am dealing with decision latency of fused big data (of 3 firms), which have 73,476 rows total and 21 columns each (huge volume) and has high frequency. In this environment, decisions, both arithmetic and computational, have to be made within maximum 59 seconds before the next record is received at the 60th second. In this duration, market risk estimation using linear and nonlinear models, which adds to the latency, also needs to be computed.

I have developed an algorithm for stock market risk estimation which could be executed in parallel, thus, making use of the distributed and parallel processing capabilities provided by big data architecture - Spark and Hadoop. This algorithm uses econometrics and advanced statistical processing to not only predict the market risk of the 3 US equity firms but also, compare it with the overall market risk. For the latency study, I have compared the performances and resources utilization of Apache Spark and Apache Hadoop with each other and with my personal computer.

Hadoop Map-Reduce and Spark both outperformed the baseline system (my personal computer) in decision latency and resources utilization while Hadoop outperformed Spark in each of these aspects on all the datasets. The market study revealed that at high frequencies, the firms' data was highly volatile. As the frequency was decreased from 1-minute to daily, then weekly, then monthly, the volatility reduced further and further and market smoothed out showing that over a period of time the firms behaved like the overall market.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Latency refers to the total time taken between learning about the changes in a quantity of interest and response generated to those changes [8]. In this age of technology and internet, more data is being generated per second than was generated in a year. Data is not only being generated at high volume but also at high velocity which has made its storage, extraction and processing even more challenging. A quantitative measure of latency is by identifying the processor, disk, memory and network (in case of distributed cloud-based architecture) throughput of the computing systems.

Some delay-sensitive applications have an inherent need of low-latency to get real-time control of their decision making process. One such example is weather systems in aircraft. Aircraft weather radar systems are used to alert aircraft pilots, of weather hazards that may affect the aircraft. In order to be accurate, they need to get this data from other aircraft and ground-based weather monitoring systems, in near real-time. For this, they need highly precise low-latency computational systems to make a timely decision on the route of their airplane [9]. Another example is stock market trading in finance. Speed, in absolute mathematical terms, is essential to traders not only due to the innate volatility of the stock market but also due to the need to be faster than the other traders to make faster decisions based on a change in the market activity or the news. In order to generate a profit, the traders also need to evaluate the market risks which arise due to intense variability and dynamic nature of the stock market [10].

This need of low-latency and the complexities of dealing with it especially with big data has given rise to many high powered computing (HPC) systems. Decrease in latency could be achieved by both hardware and software improvements. One way to achieve low latency is to add more processing power, like CPUs, GPUs, RAM, etc to the existing hardware which could be as simple as a desktop or as complex as a super-computer. However, this isn't always very cost effective or feasible. Another way is to use distributed development frameworks which provide the capabilities of parallelism and distributed architecture to allow high-speed computation along with fault tolerance and high availability [5]. Some of the most popular open source distributed computing big data technologies in the market today are Apache Hadoop, Apache Spark, Apache Flink, Apache Storm, etc. They not only provide capability of high-speed computation but are also easier and in most cases, less costly to scale horizontally and vertically than physical systems.

1.2 Research Problem

In finance domain, stock market traders deal with very high frequency big data and they need to make investment decisions, i.e., buy stocks, sell stocks or remain neutral, after assessing the market risk, all in a few hundred milliseconds. In other words, they have a need of low-latency computing system. The problem with their current methods of achieving this is dependency on physical systems, e.g., powerful desktops with high-end configurations, and on geographical closeness to the stock market trading venues. There exists a gap between the computing methodologies and systems used by financial traders and big data engineering technologies which provide parallel and distributed computing capabilities.

1.3 Research Question

My Research question for this dissertation is: To devise an algorithm which could be executed in parallel on big data engineering architectures to predict financial market risk and study its impact on decision latency.

1.4 Research Objective

In an attempt to address this gap in computing in stock market trading , I have chosen big data engineering technologies - Apache Hadoop and Apache Spark which provide distributed parallel computing capabilities and could be scaled as data increases. I have formulated an algorithm which utilises the parallel execution abilities of the above mentioned technologies in conjunction with the calculation of statistical moments of distribution, financial market trading rules and linear modeling to predict the market risk of a specific sector ,i.e., U.S. airlines sector with respect to the overall market as well as predicting investment decisions, i.e, buy stock, sell stock or remain neutral. For this research, I have used 3 high frequency stock market datasets for 3 US stock equity firms in the airline sector and 1 dataset with 3 different low frequency durations for a US stock market index. The results have two aspects - latency study and market study. For latency study, I have assumed my personal computer to be a baseline model. Experiments with big data technologies show significant improvement in latency study while market studies show that over a period of time, the market smoothed out.

1.5 Contribution

The key contributions of this research are as follows:-

- An algorithm which combines advanced statistical methods and econometrics with Big Data architecture for market risk prediction.
- Evaluation of market risk for a specific sector and comparison with overall market. This requires a parallel computation architecture to reduce execution latency involving more memory as frequency of data increases.
- Comparison between 2 major architectures (Hadoop and Spark) for latency study. I have looked at total execution time, CPU utilization and memory utilization trade-off for reducing latency.
- Addressed the gap between finance high frequency big data processing and big data engineering architecture.
- A research paper on this topic is in the works to be published.

Although I am using finance stock market data for this case study, this method for achieving low-latency could be extrapolated to other big data applications which have inherent requirement of low decision latency, e.g., healthcare, weather systems, environmental studies, etc.

Chapter 2

Background

2.1 Big Data

My research is focused on reducing the decision latency in processing of high-frequency big data. I am using historical stock market data to study latency and for market risk estimation. Big Data has been defined as "very large sets of data that are produced by people using the internet, and that can only be stored, understood, and used with the help of special tools and methods" [11]. This data could come from varied sources, at high velocity and maybe heterogeneous in format. Businesses often want to assess this big data and turn it into meaningful, actionable information.

In order to categorise data as big data, it should have these 5 most important characteristics [12] :-

- **Volume** - Volume refers to the amount of data which is generated by organizations. This is the most important characteristic of big data. This data could be collected from multiple sources, e.g., over the network in an organization, social media, news, sensors etc. There are organizations, like finance, which generate huge volume of data every second. Storing and processing this data in a timely manner to make decisions becomes a challenge.

- **Velocity** - Velocity refers to the speed at which the data is created and with which it moves around. In order for a data to be characterised as big data, not only should it be of large volume but also come in at high velocity. This speed makes real-time processing harder than on data with low velocity, especially for applications which are latency-critical and should make decisions on the data as soon as they are obtained.

- **Variety** - Variety refers to the heterogeneity of the data. This data could be obtained from varied sources and could be in different formats. It could be structured and relational, which is stored in relational databases or unstructured, which needs special schema design for storage. Some examples are email, financial transactions, text documents, audio, video, etc. The complexity of the data increases with increasing variety.
- **Veracity** - Veracity refers to the "truthfulness" of the data. It points to the quality and the accuracy of the data collected along with the entities, both internal - like the machines collecting and processing the data, and external - like the source of the data, which might influence the quality of this data. This becomes increasingly complex as the volume, velocity and variety of the data increases.
- **Value** - The data collected should should have "value" for the organization for which it is collected. High value data, for an organization, provides valuable business insights and helps them formulate better business strategies.

Volume
Velocity
Variety
Veracity
Value
Big Data

Figure 2.1: The 5 Vs of Big Data [1]

These characteristics are pictorially represented in figure 2.1

2.2 Distributed Computing Technologies

Big data analytics refers to the analysis of big data to generate insights. The problem in analysing arises when this data is voluminous, comes in with a high velocity and is heterogeneous. It leads to computing issues with the existing systems which cannot be scaled up to deal with these complexities of this in-flowing data. Another problem which arises is to indemnify the veracity and responsiveness of this data. This is critical in the competitive world today, where delay of a few minutes could cost large sums of money, especially, in areas such as finance. Thus, arises the need for high performance computing systems to deal with these problems.

High Performing Computing systems could be used to deal with this problem. They are essentially computer systems (e.g. a desktop), to which large number of CPUs, GPUs, RAMs, are added. But with the magnitude of the data today, this solution is not very scalable and cost effective. Another solution, which is widely popular, is the usage of cluster-based commodity setup which provides *distributed computing*. The underlying concept of both the solutions is to provide bigger storage and faster processing of data. One common way in which this is achieved is through *parallelism*.

2.3 Map Reduce

The most common programming model used in distributed computing is the **Map Reduce** (MR) model. It works on the concept of divide and conquer. As the name suggests, this model consists of two stages - Map and Reduce. The former stage is responsible for splitting up of data into smaller parts and the latter is responsible for fetching and aggregating it. The primary data structure used for this is the (key,value) pair at all the stages of data processing [5].

The MR paradigm works in the following way (figure 2.2):-

- The Map stage gets the data and converts it into key-value pairs and forwards it to the next phase. This conversion does not follow a sequence, thus, is executed in parallel.

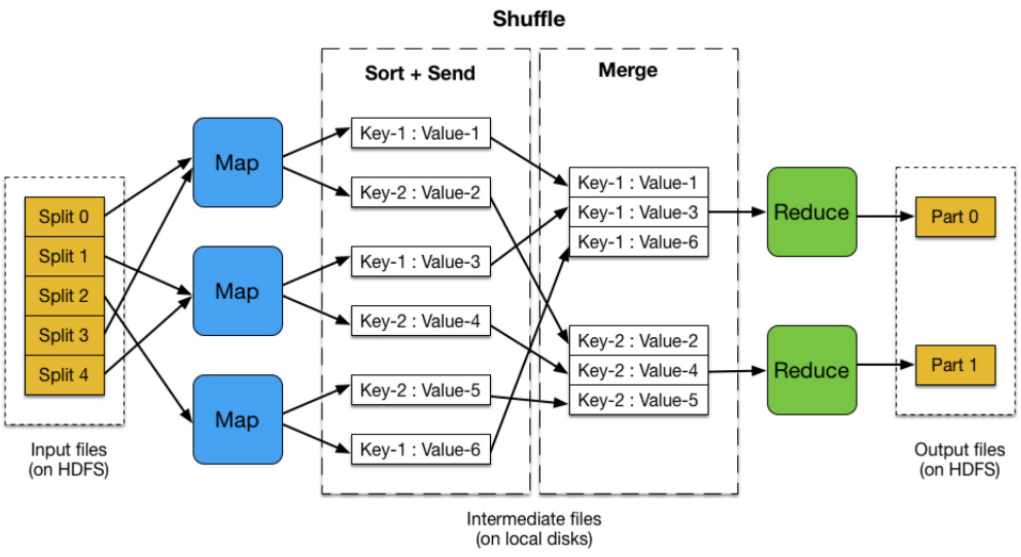


Figure 2.2: The Map Reduce Architecture [2]

- The intermediate phase receives these pairs of data and groups together the similar keys which provides a list of values. Partitions are formed on the keys and sent to reduce stage. These partitions depend on the algorithm defined by the user.

- The reduce stage obtains the data pairs from the previous phase and performs key-wise or total fusion of the data and sends the final output.

This partitioning and fusion technique not only helps in splitting the data into smaller parts so that individual parts may be processed in parallel, thus, providing faster computing speed, it also decreases the amount of data sent over network. In this way it reduces latency.

2.4 Hadoop

Apache Hadoop is the most widely used technology for big data analysis. It is an open source implementation of Map Reduce [12]. It follows distributed computing paradigm with cluster-based setup of commodity hardware. Due to this kind of setup, it is scalable, both horizontally (scale-out) or vertically (scale-up) in an efficient, cost-effective manner and based on requirements. It has an automated built-in capacity for providing disaster recovery of data, fault tolerance, job scheduling, etc. It was developed for batch processing of data. But in recent years, it has been extended to perform processing on live streaming data as well.

Hadoop has been mainly written using Java Programming Language but it provides streaming capabilities in the form of Java Archive Resources (.jar) files which could be integrated with Python Programming Language, which has been used in this research. Hadoop Architecture consists of three main components:-

1. **Storage - Hadoop Distributed File System** or HDFS, as the name suggests, is a distributed file system used for storing large amounts of data in a distributed manner, when the data gets too huge to be stored on a single file system. This data is replicated and is available for all processing systems on the Hadoop cluster. The files on the HDFS are stored in blocks with default size as 64 Mb. There are 2 types of nodes in the HDFS cluster (figure 2.3):-

   (a) **Namenode** - This is responsible for management of all files and data in the file system namespace. It also controls and manages the mapping between the data and its storage block.

   (b) **Datanode** - This stores the actual data in blocks and reports it to the namenode so that the latter could retrieve the data for processing.

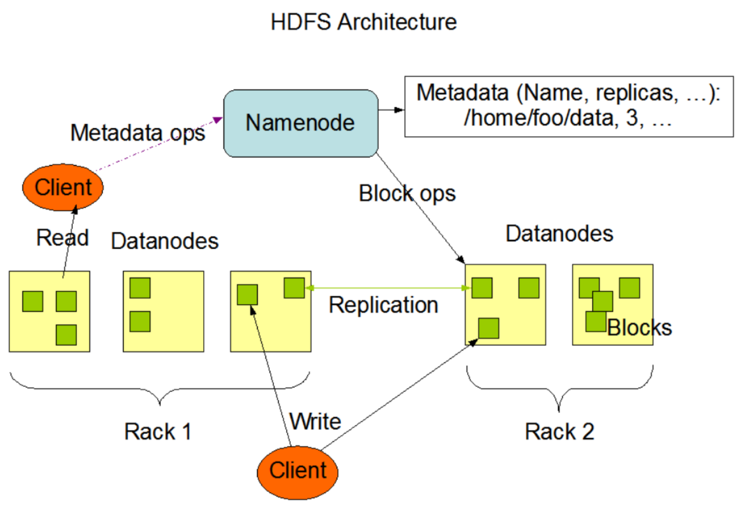


Figure 2.3: HDFS - Namenode and Datanode architecture [3]

2. **Processing** - Here the **Map Reduce** programming paradigm is used to process the data stored in HDFS. As mentioned in the previous section of this chapter, the actual tasks are split into the primitives mapper and reducer functions for parallel processing. Map "maps" the data into a group of tuples of (key,value) pairs. It produces an intermediate output which shuffles and sorts the keys and forms a list of their corresponding values which are then partitioned and sent to reduce for fusion and aggregation. Since, the data is distributed across nodes in the Hadoop cluster setup, the processing map-reduce algorithm runs in parallel across each node, but on different datasets at a time, thus, decreasing the processing time and reducing latency. The parallel computation and its complexities are controlled by Hadoop using distributed file system and message exchange protocols.

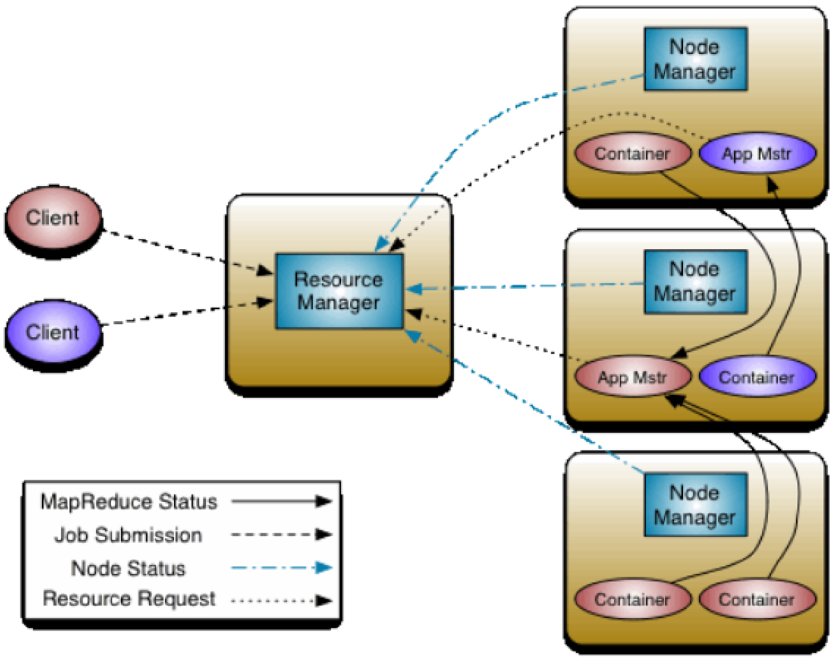


Figure 2.4: YARN architecture - scheduler and resource manager [4]

3. **Resource Manager** - The **YARN** or Yet Another Resource Negotiator is responsible for maintenance and management of resources in the Hadoop cluster. It has 2 components - Scheduler and Resource Manager (figure 2.4). It provides huge advantages in terms of reuse of resources in parallel for applications which are alike. It also allows multiple applications to run in parallel at the same time

on the cluster without needing to move data.

2.5 Spark

Apache Spark Framework is a distributed processing system developed to perform computations on big data in a short period of time. It was built as an extension of Hadoop Framework to deal with very disk I/O intensive, interactive and iterative problems which could not be handled by Hadoop, by using memory-based operations. Its main focus is processing live streaming data, however, it can also be used for batch processing. It has its own processing engine [13].

Spark works on **Master-Slave** setup where the master acts as a resource manager, allocating tasks and resources to slaves and slave/s perform the actual computing tasks. It can be a standalone single node setup, with 1 master and 1 slave or could be a cluster setup with one/many master/s and many slaves. They help provide data replication and fault tolerant capabilities in the cluster setup.

Spark provides a native language support for Scala. However, it extends its API support to languages such as Python Programming Language, R and Java [14].

Spark employs a strict generalization of the Map Reduce paradigm called **Directed Acyclic Graphs** or DAG which splits the dataset into smaller atomic tasks. The edges in this graph represent the exchange of information and the vertices represent parallel tasks (figure 2.5). Note that the MR has 2 specific vertices whereas DAG can have many. This supports parallelism by allowed the same data to be used by multiple vertices or same vertex to use multiple datasets. The edges work by using shared memory or disk.

The heart of Spark is a data structure called **Resilient Distributed Dataset** or RDD. It is a collection or a group of partitions of data spread across various data nodes in a Spark cluster. It provides high-level functions for distribution and transformation of data across the cluster. RDDs provide programmers the capability for defining their own methods and schemas for the data partitioning and for controlling the data format.

Spark provides other high-level abstractions such as **DataFrames** and **Dataset APIs**. See figure 2.6. The DataFrames are formal schemas in RDDs which are used for distribution of groups of structured and relational datasets by dividing them into named columns. The concept is similar to relational databases. "Spark Catalyst

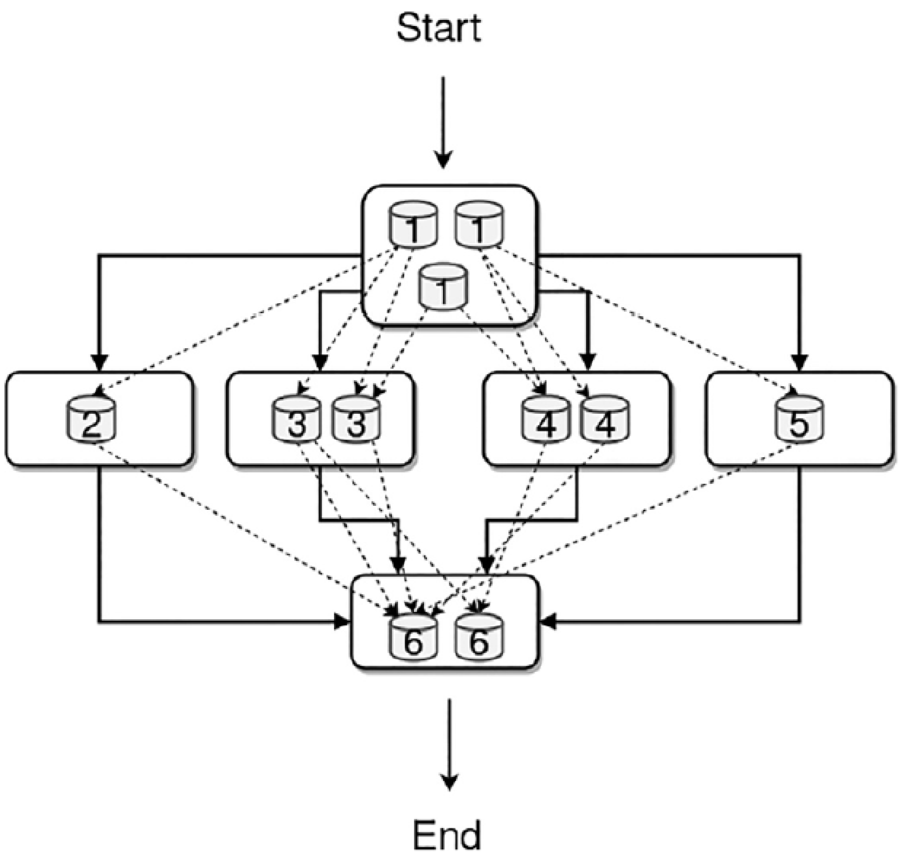


Figure 2.5: Directed Acyclic Graph Parallel Processing. The square boxes represent the vertices (tasks), the solid lines represent edges (data transfer), the dotted lines represent dependencies between cylinders (data blocks). [5]

Optimizer" optimises the querying of these DataFrames. The Dataset APIs are a collection of strongly-typed objects with connection to a relational schema. They achieve a compromise between control over structure of data provided by DataFrame and optimization provided by the Catalyst. These abstractions provide greater control of the data and in turn the design of algorithm for computing which are made parallel internally by RDDs for reduction of latency and computing time.

2.6 Differences between Hadoop and Spark

Hadoop and Spark, both are open-source distributed computing technologies which can be deployed in cluster or standalone mode. They provide parallelism capabilities

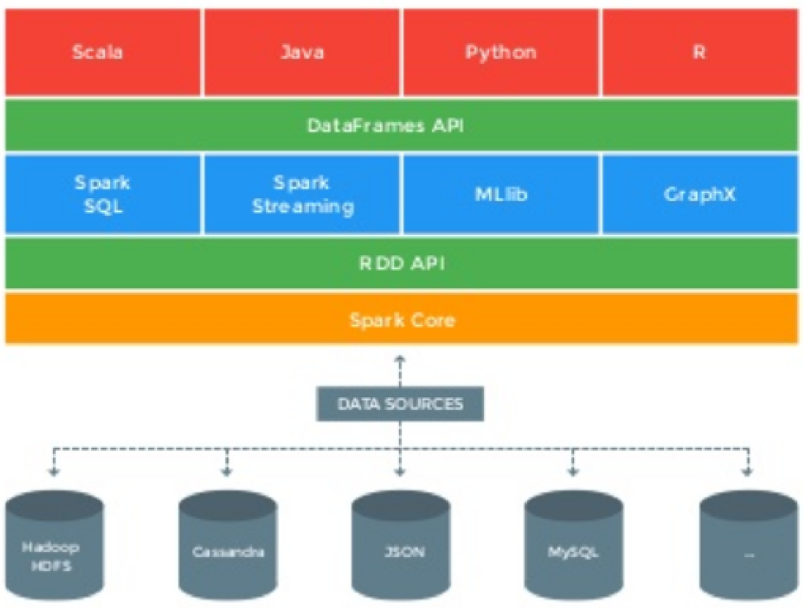


Figure 2.6: Spark Framework

which greatly help in reducing processing time and decision latency. They also provide fault tolerance mechanism and high availability. The main difference between Hadoop and Spark lies in the design philosophy.

Some of the major differences are listed below:-

Hadoop	**Spark**
Has a separate storage (HDFS) with high read throughput	No separate storage. In-memory data computation
Mainly used for Batch Processing	Stream and Batch Processing
Disk I/O overhead more but lesser RAM usage	Lazy evaluation reduces disk I/O cycles but more RAM needed
Complex Map-Reduce model. Low-level APIs involved	Highly abstract Map-Reduce model

Table 2.1: Differences between Hadoop and Spark

As the table suggests, Spark uses a technique called **Lazy Evaluation** on the dataset where the intermediate tasks of an algorithm persist in-memory and are not processed until the a transformation function is applied like ”reduce” or ”collect” and the output is requested. This drives up the RAM usage for Spark ecosystem. Hadoop, on the other hand, uses more disk I/O cycles since it reads data from HDFS in every iteration of the algorithm but uses lesser RAM than Spark.

Even though Hadoop was primarily developed for Batch Processing and Spark for Stream Processing, both can be used for both the tasks, however, their speciality lies with one task.

Chapter 3

Motivation and Literature Review

In this chapter, I talk about the work done so far in the field of financial analytics and low-latency architectures. The chapter begins by analysing the literature related to high frequency trading in stock market and the need and impact of low-latency in this field. I, also, talk about the algorithmic methods which have been used in stock market for the forecasting of risk. This is followed by the study of low-latency methods extant in finance.

Next I discuss the big data processing systems and their usages in low-latency applications. Finally, I talk about my motivation to do this research whose basis shall be clear from literature review.

3.1 Low-latency in Finance

With the introduction of algorithmic stock market trading, the data generated in the stock market is rising by leaps and bounds. This data not only has huge volume but also comes in at high velocity. Financial analysts and stock market traders have to make use of this data, both current and historical, to predict future trends in the prices of stock market and estimate the market risks. In order to beat the competition and gain profitability for their customers, the traders have to reduce the latency time of placing their orders to buy, sell stocks.

[8] give the definition of low-latency activity in financial market as the ”strategies which respond to market events in millisecond environment”. This paper studies the

impact of low-latency, high-frequency algorithmic trading on the market quality in general and proposes new measure of this impact. They conclude that the low-latency activities benefit the markets not only when they are stable but also during uncertain times, in short-term, by reducing the spread and volatility of data, which in turn reduces risk. They also talk about the methods for reducing latency which a lot of stock exchanges offer, known as "co-location" facilities, where the traders co-locate or place their computer systems in close proximity to exchanges. The traders also use high-end computing systems with more processing power, like CPU, memory, etc., for the same purpose.

[10] talk about the impact of latency on arbitrage opportunities. They document the impact of the number of co-location cabinets provided by the Australian Stock Exchange on the arbitrage profitability of the market. Arbitrage is the buying and selling of financial assets by exploiting the differences between the prices of similar or same assets across forms or markets. According to the authors, the increase in high frequency trading and low latency systems due to the rise in "co-location connections" , results in rise of arbitrage opportunities. This is because the market risk and volatility increase as the mispricing of assets increase due to increase in velocity to the data. This study contradicts the study in [8] about the market quality but one thing is certain that increasing volume and frequency of stock market data and low-latency trading, complicate the prediction of risk of the market.

The advantage of co-location is that it reduces the network latency and decreases transmission time down to milliseconds but the biggest drawback lies in the fact that physical proximity is needed to the exchanges to be able to reduce latency which is not always possible especially in current circumstances where everything is closed down and the global world is adopting online methods to solve their business needs. Another drawback is the dependency on hardware systems which is not always cost effective nor scalable. With the growth of data, in volume and velocity, easy and cost-effective scalability is the need of the hour.

The work of [14] comes close to my research in terms of using parallelism in hardware and software to achieve low-latency in high-frequency intraday data to calculate portfolio risks. They use different financial methods to mine to calculate portfolio risks and their parallelism involves usage of parallel R with Intel Math Kernel Library (MKL), and automatic offloading to Intel Phi co-processor. They use the capabilities offered by the Intel MKL to speed up application performance for mathematical calculations by using multiple processor cores. The offloading to Phi co-processor is done by using environment configuration variables on the operating system. While this study shows significant improvement in the latency of computation, the software design for this system requires knowledge of CUDA coding and the concept of multi-threading in processor cores.

3.2 Market Risk Estimation

The market demands and the application objectives of finance, marketing and economics are one of the most intriguing big data applications to draw the attention of researchers studying quantitative data models. This is also due to the fact that the data in financial domain is fairly structured. But doing market risk analysis in finance is a difficult problem because an economic system - containing both consumer market and financial market, is an "open-loop" system where behaviour of human beings is also an input factor. Therefore, problem solvers in this domain work with various assumptions which cannot be verified quantitatively. One of the biggest assumptions is that future market risks can be predicted using historical time series. For this reason, statistical methods need to employed to study the distributions of the data and the results for the purpose of model verification [15].

There are various methods of forecasting the market risks. The commonly used methods are ARMA models and ARIMA models [2]. They are used to model returns from the return time series which is calculated from price series. Some of the other models include sequential Monte Carlo methods, Markov models, Kalman filtering, etc [16].

For my research, I am forecasting investment risks using return series and studying the general trend of the firms with respect to the market. The most popular statistical model for this is called the *ARCH model*. It has some drawbacks when it comes to explaining the differences between average returns of buy and sell volatility of the market [17] which is an important parameter for predicting investment risk. I am using trading rules as one of the financial methods to predict the risks related to market. According to [17], there is sufficient evidence that these trading rules could successfully predict distributions of future returns and the differences between buy and sell volatility. Trading rules have been able to provide profitable investment decisions particularly for the US stock equity markets and firms. These rules have also been tested on currencies. The 2 trading rule which have been most popular with technical analysts are the channel rule and the moving average rule. Academic researchers also use the filter rule. Also, usually the transaction costs for trading stocks and making investment decisions are less than the profits generated from the trading rules. Apart from this method, moments of distributions are extremely useful when trying to understand the statistical properties of the time series. The combination of both these methods help identifying the general market risk.

3.3 Big Data Technologies

With the rise in volume and velocity of data, one of the biggest challenges is the capability to process, store and retrieve this data in a timely manner. Implementations of algorithms to run in a serial, sequential manner, doesn't help with our problem of low-latency. This has given rise to the concept of parallelism. It can be achieved by technologies like using multi-core, many-core CPUs, GPUs, etc., which use the concept of parallelism in a way. However, there are trade-offs in the form of cost, fault-tolerance, data loss, etc. A more efficient way is to use distributed cluster-based computing systems which not only provide high-speed computation by using parallelism but also provide capabilities such as data replication, fault tolerance, prevention of data loss, data recovery and most importantly are very scalable [13].

[18] have looked at the usage of various distributed computing architectures and their advantages and disadvantages in big data analytics like Hadoop MapReduce, Spark, Hive, Storm and Impala. While promising, the focus of their work is more on building an efficient data management system for big data analytics than studying the impact of latency on current financial methods using these big data technologies.

Apache Hadoop and Spark are two of the most popular big data frameworks when it comes to batch processing. That is one of the reason, I am using these two, in my project. The other reasons are their easy installation, ample documentation and integration with Python.

3.4 Motivation

Finance is dealing with very high velocity and high volumes of data which is only going to rise due to algorithmic trading. Low-latency decision-making is very important here as the traders make profits for their customers according to their speed and accuracy of decision-making. A few milliseconds of difference can gain (lose) 100-million dollars per year in profits for a high-speed firm [19]. Thus, high speed computation is needed for reducing decision-latency. While stock exchanges provide their own solution to solve this problem (co-location cabinets), this addresses only the issue of network latency. For computation, they still rely on physical servers and high-end computing systems which are not scalable and cost-effective, with the growing data. Big data technologies providing distributed computing and parallel processing capabilities can solve this problem.

There is also considerable research on financial methods for forecasting of market risks using high-frequency and low-frequency big data. But from the literature review, it is clear that there is a noticeable gap when it comes to usage of these financial algorithms with big data processing systems for reduction in latency of market risk estimation. This is my primary motivation for this research.

I am using Python Programming language for this project. Python has become the language of choice for data scientists and data engineers due its dedicated libraries for handling big data (Pandas [20]), and statistics (scipy [21]). I am also using big data technologies and python integrates well with those frameworks.

Chapter 4

Econometrics

Due to the dynamic nature of stock market, it is important for traders and financial analysts to do risk estimation of the market before they can make investment decisions for themselves or their customers. A slightest error can cause loss of millions of dollars. Thus, it is essential to understand the changes in the prices in the market. Technical Analysis of the market is the most common method for analysing these changes and making informed decisions. These methods are described in the next sections of this chapter.

4.1 Statistical Moments of Distribution

-

The Log Returns time series of almost all stock markets follow certain sets of general rules. One of such rules is that it does not follow a normal distribution curve [17]. Thus, to study the statistical properties of the return series, which is done in order to identify the market risk, we use these characteristics, called moments of distribution:-

$$Mean(r) = \frac{1}{n}\sum_{t=1}^{n} r_t \tag{4.1}$$

The mean (r), called the first moment of distribution, in the case of distribution of returns, is zero or close to it, since, in an efficient market, every positive return is followed by a negative return. If the mean is negative (positive), it means that the down-tick (up-tick) in returns is not balanced out. This could be the earliest indicator of an unstable market or in other words, market risk.

$$StandardDeviation(s^2) = \frac{1}{n-1}\sum_{t=1}^{n}(r_t - r)^2 \tag{4.2}$$

The standard deviation gives the second moment of distribution. It is an indicator of volatility or the dynamism of the market and hence, the risk. According to financial experts, stock markets, inherently, are very volatile.

$$Skewness(b) = \frac{1}{n-1}\sum_{t=1}^{n}\frac{(r_t - r)^3}{s^3} \tag{4.3}$$

Skewness is the third moment of distribution. It tells about the asymmetry in the distribution function of returns along with the direction of outliers. It again indicates the market risk. If the distribution is postively skewed, it means that the number of negative returns are more than the positive returns and vice-versa.

$$Kurtosis(k) = \frac{1}{n-1}\sum_{t=1}^{n}\frac{(r_t - r)^4}{s^4} \tag{4.4}$$

Kurtosis is the fourth moment of distribution. It tells about the frequency of returns accumulated in the peaks and the tails of distribution function. A normal distribution has 99.99% data between 3 standard deviations of the mean. Kurtosis value exceeding 3 standard deviations implies that the distribution is not normal and hence, risk calculation of the market is more complex than what it would be in a normal distribution.

These moments of distribution help in studying the distribution of the returns of prices which reveal information about our dataset. Since, our return values (r) are discreet random variables in nature, we use summation here. This sign would change to integration in case of continuous random variables. One important thing to note here is that, these moments of distribution are sensitive to outliers in the datasets. We do not remove the outliers in our dataset because any anomalies related to it could be a good indicator of the market risk.

The fact that returns distribution is not normal, adds up to the complexity of calculation of market risk and hence, the latency.

4.2 Trading Rules

- Trading rules are mathematical rules which take log returns of historic stock market time-series, as inputs, and turn them into investment decisions, i.e., whether to buy stocks, sell stocks or remain neutral [17]. The rule which I have used is the **Moving average rule**. It is a classification algorithm which compares the difference between moving averages of two lengths of time - S, i.e., the short term moving average and L, i.e., the long term moving average to classify days or next moment in time as buy, sell or neutral. It also uses a hyper-parameter called Bandwidth B, to take a certain amount of buffer between Buy and Sell days. The formulae are: -

$$a_t, S = \frac{1}{S} \sum_{j=1}^{S} (p_t - S + j) \tag{4.5}$$

$$a_t, L = \frac{1}{L} \sum_{j=1}^{S} (p_t - L + j) \tag{4.6}$$

$$R_t = \frac{(a_t, S - a_t, L)}{a_t, L} \tag{4.7}$$

According to [17], the most commonly used values of S is 1, L is 50 and B is zero. I have used these values in my research.

If the value of short-term moving average is higher than the value of long-term moving averages, it means that the stock price is following upward trend and if the value of short-term moving average is lower than long-term moving average, it means that the price is following downward trend. When the two values are same, it means we do not have enough information to predict the trend of prices. Based on this, if we have the value of log return at the time t, we could classify the time $t + 1$ as either Buy, Sell or Neutral based on the below formulae:-

$$R_t > B, Buy, \tag{4.8}$$

$$R_t < B, Sell, \tag{4.9}$$

$$-B <= R_t <= B, Neutral \tag{4.10}$$

Here R_t is the difference between short-term and long-term moving averages as given by equation 4.7. These rules give information about statistical properties and hence, help reduce the market risks, especially, when large volumes of data are involved (like in this research). They could be applied to both low and high-frequency trading datasets. This rule was used here as it has low systemic risks in the case of stock market asset predictability.

We have used these advanced statistical methods with Hadoop and Spark to study latency while estimating market risks whose details are given in the next chapter.

Chapter 5

Methods

The entire study was divided into 2 stages:-

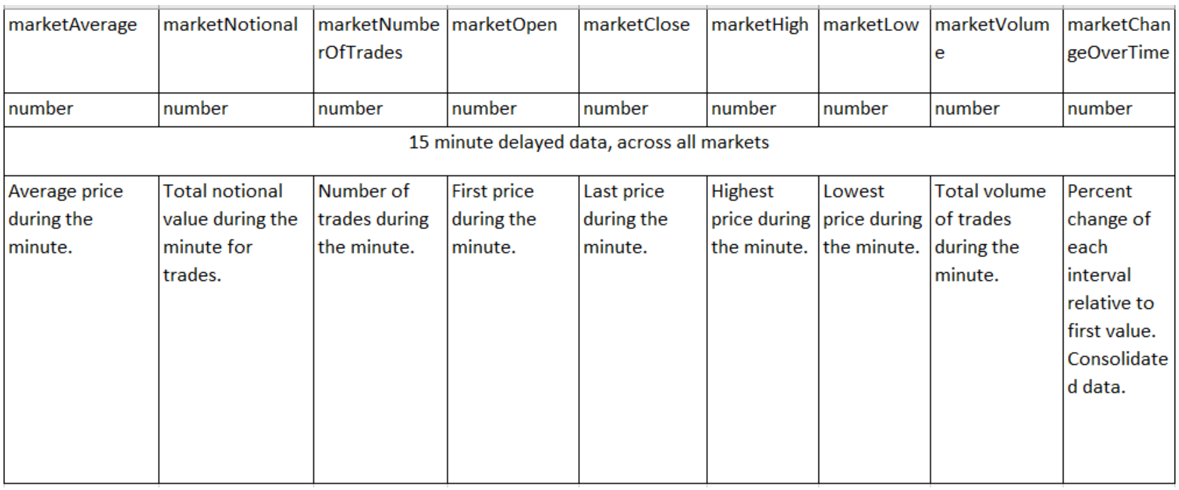
- Data acquisition and pre-processing
- Statistical modelling

Each of the stages are described in detail in the subsequent sections.

5.1 Dataset

I am using publicly available stock market data. They are of 2 categories:-

- **High Frequency financial data** - For the latency study, I am using 1-minute intraday historical data of duration -3 months, from 1st October 2019 to 31st December 2019, for 3 US equity firms. These firms - American Airlines, Delta Airlines and United Airlines belong to the US airlines sector. The total number of rows are 73,476 and each file has 21 columns each. This data was downloaded from the IEX Cloud website [6] using their publicly exposed REST API using a Python script developed by me. These APIs work on the standard HTTP request-response structure and send the output in JSON encoded format. I converted this JSON output into standard Pandas dataframe format and stored it in the form of .csv (comma separated values) files. Pandas is a specialised library of Python which makes handling of huge datasets easier. The columns in these 3 datasets and their interpretations are as given in figure 5.1:-



marketAverage	marketNotional	marketNumberOfTrades	marketOpen	marketClose	marketHigh	marketLow	marketVolume	marketChangeOverTime
number	number	number	number	number	number	number	number	number
15 minute delayed data, across all markets								
Average price during the minute.	Total notional value during the minute for trades.	Number of trades during the minute.	First price during the minute.	Last price during the minute.	Highest price during the minute.	Lowest price during the minute.	Total volume of trades during the minute.	Percent change of each interval relative to first value. Consolidated data.

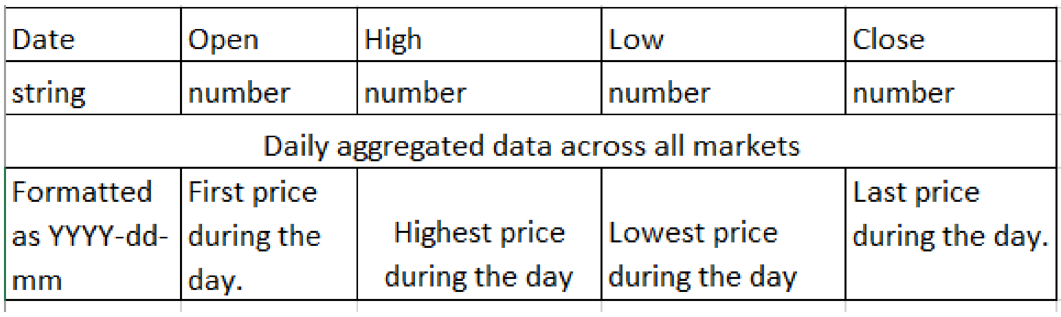
Figure 5.1: Description of the important columns in High Frequency Dataset [6]. The most important columns for us are date, time, market close value.

The value we are interested in is the 'market close'. 'Market close' is the closing price of stocks during the 1-minute duration for high frequency data. I drop the rows containing missing/null values in the aforementioned column. This market close value is used further to convert the raw stock market price time-series into return series. Also, a new column 'company' is added here which helps us in generating (key,value) pair in mapper phase.

- **Low Frequency financial data** - For the market study and market risk estimation, I am using daily, weekly and monthly data for the year 2019 (historical data) for a US stock market index - S and P 500. This stock market index is the weighted average of all stocks listed in the stock market for top 500 companies [7]. It tells us the overall drift and the trend of the market. These 3 duration, low-frequency stock market trading datasets were downloaded from Yahoo! Finance using the UI on their website, manually and stored in .csv files. Yahoo! Finance is a property of Yahoo! Network. It gives information such as news related to finance, financial reports, etc.

  The columns in the Low frequency dataset for daily data are given in the figure 5.2:-

  As is the case for high frequency data, the values we are interested in is the



Date	Open	High	Low	Close
string	number	number	number	number
Daily aggregated data across all markets				
Formatted as YYYY-dd-mm	First price during the day.	Highest price during the day	Lowest price during the day	Last price during the day.

Figure 5.2: Description of important columns in Low Frequency Daily Dataset [7]. The most important columns for us are date, time, close value.

'close' value. 'Close' is the closing price of stocks during the daily, weekly and monthly duration for low frequency data. This 'close' value is used to convert the raw stock market price time-series into return series which is used further to do regression analysis in market study. This study studied the volatility of the stock market and the general trends as the duration of the data (daily, weekly, monthly) increased.

Along with this, there are a few calculated datasets which are used for market analysis. I aggregate values of AAL, DAL and UAL datasets over an hour and 3 hours to find correlations and market trend over different time gaps among them.

5.2 System Architecture

I am using publicly available stock market data. The data for the 3 US equity firms - American Airlines, Delta Airlines and United Airlines, for 3 months, is high frequency intraday, 1-minute data obtained from IEX Cloud website. The low frequency, i.e., daily, weekly and monthly datasets for US stock index - S and P 500 are obtained from Yahoo! Finance website. These datasets required cleanup of missing values.

After data cleaning and data imputation, the refined datasets were passed through Hadoop Map-Reduce and Spark SQL separately to perform further calculations. Here, the algorithm which was used for statistical modelling and prediction of market risk was divided into 2 phases:-

- **Mapper** - In this phase, the stock market price time-series was converted into log return time-series, This step was important in order to do statistical modeling. The log return time-series was sent to reducer for further calculations

- **Reducer** - In this phase, statistical moments of distribution and market trading rules were applied to the log returns time-series obtained from the mapper in the last phase. The output of this phase is the investment risk prediction for the firms.

Splitting the algorithm into mapper and reducer phases helped in achieving parallelism of the complex mathematical computations for the 3 high frequency datasets (See figure 5.3) when using big data engineering technologies described in the next sections.

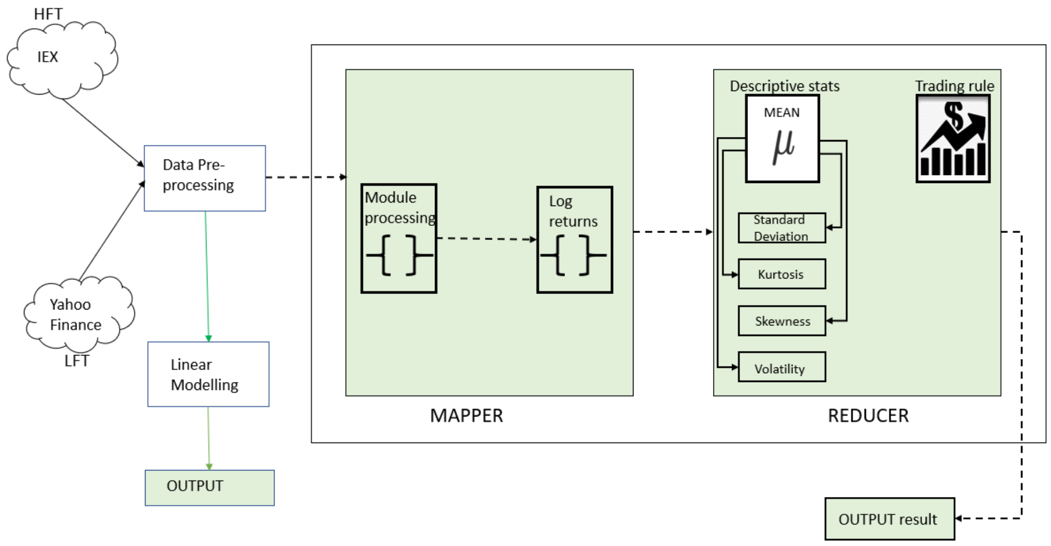


Figure 5.3: The project Pipeline. Here HFT stands for high frequency trading data downloaded from IEX Cloud and LFT stands for low frequency trading data downloaded from Yahoo! Finance. HFT data is pre-processed and sent to big data architecture for calculation of investment decisions. LFT data is pre-processed and is used for regression analysis to study correlations between market and firm data

5.3 Hadoop Implementation

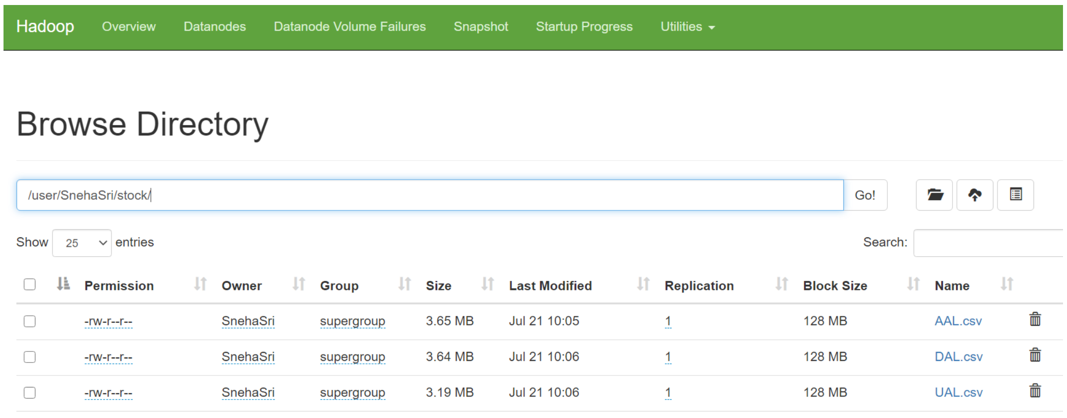
For this project, one of the distributed computing architecture I used was Apache Hadoop with Map Reduce. Hadoop cluster could be setup in one of the three ways:-

1. **Standalone Mode** - This setup uses 1 Namenode and 1 Datanode but doesn't use HDFS. The dataset is used directly from local storage for this kind of setup. This is a debug mode for running Hadoop.

2. **Pseudo-Distributed Mode** - This setup uses 1 Namenode and 1 Datanode and also uses HDFS for loading and distributing data for processing. This uses all of the Hadoop components, i.e., the storage, the map reduce processing and the YARN and emulates the cluster setup most closely. It is used the most for academic research.

3. **Fully-Distributed Mode** - This is the full cluster setup of Hadoop where one/many Namenodes and many Datanodes are used. This is used the most in enterprise settings where high availability of an application is needed along with data replication and fault tolerance. It can be scaled-up and scaled-out.

I used pseudo-distributed setup of Hadoop Map Reduce architecture for this research as it closely resembles the fully distributed deployment. After the installation and setup on my local computer, the YARN resource allocater was brought up first, followed by the Namenode and the Datanode. The 3 high-frequency dataset files were 11 Mb in size and were copied to HDFS which took approximately 4 seconds to get copied (figure 5.4).



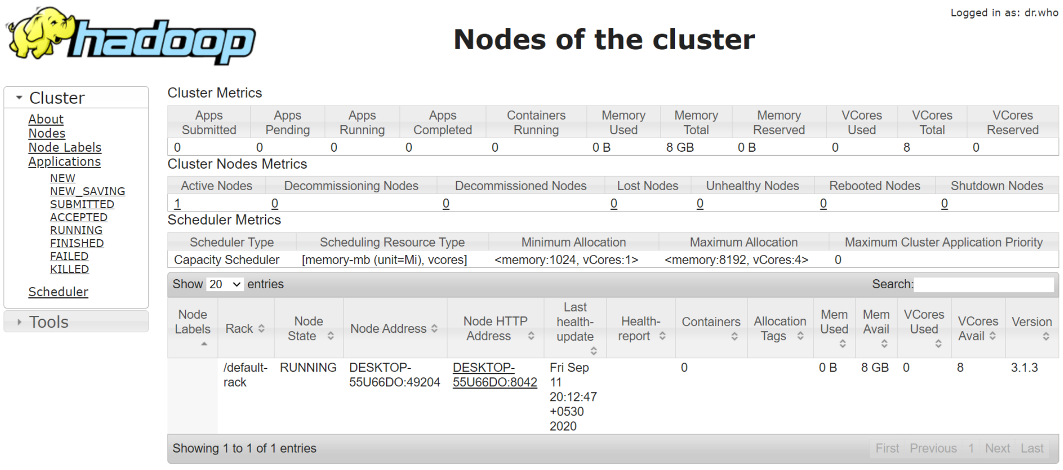
Browse Directory

/user/SnehaSri/stock/

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	SnehaSri	supergroup	3.65 MB	Jul 21 10:05	1	128 MB	AAL.csv
☐	-rw-r--r--	SnehaSri	supergroup	3.64 MB	Jul 21 10:06	1	128 MB	DAL.csv
☐	-rw-r--r--	SnehaSri	supergroup	3.19 MB	Jul 21 10:06	1	128 MB	UAL.csv

Figure 5.4: This figure shows the high frequency datasets uploaded on HDFS. The block size allocated here is 128 Mb and replication factor is 1 since this is running in pseudo-distributed mode

For processing, I developed an algorithm which would utilise the parallel processing capabilities of Map Reduce. The algorithm was written in Python and Hadoop's streaming API was used to run this algorithm. It consists of two phases - Mapper and Reducer. After cleaning the datasets and dropping null values, the datasets are fed to the Mapper phase. This reads the files from HDFS using the "standard input" functionality of Python, line by line. The conversion of stock market time series to log return time series happens here for 'closing prices' of the market for each minute for every equity firm. This mapper phase follows the concept of 'Map' stage in Map Reduce described in Chapter 2 and forms a collection of tuples of (key,value) pairs for each each dataset. These collections are passed to the next phase of the algorithm called the "reducer" phase which works on the same concept as 'Reduce' of the Map Reduce paradigm. This is achieved by using "standard output" on tuples in mapper and reading the output stream line by line in reducer. This input/output stream used by mapper and reducer gives Hadoop the illusion of streaming data.



Nodes of the cluster

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	8 GB	0 B	0	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default-rack	RUNNING	DESKTOP-55U66DO:49204	DESKTOP-55U66DO:8042	Fri Sep 11 20:12:47 +0530 2020		0		0 B	8 GB	0	8	3.1.3

Showing 1 to 1 of 1 entries First Previous 1 Next Last

Figure 5.5: This figure shows the resource allocation to YARN and total utilization when an application runs.

In the reducer phase, the values achieved are equity name, date, time, log return values of prices. Point to be noted here is that these values are all in "String" format being read from streaming input functionality of Python and are converted into floating point numbers before further calculations. It is reducer phase, where heavy computations and calculations are done. The statistical moments of distribution are calculated for each row of tuples obtained and trading rules are applied to estimate market risk and give the investment decisions of buying, selling stocks or remaining neutral for each minute before the next row is read. Latency is calculated for these computations on each row. This latency has to be less than a minute otherwise the decision would be delayed which would be of no use to the trader.

Most of the heavy computations are done in the Reducer phase, and the Mapper phase servers as a pipeline to transform the existing time-series into another form. It also forms the (key,value) pairs for each of the datasets and keeps sorting, shuffling, grouping and sending them simultaneously to the reducer. Both of these phases help in achieving parallelism and reducing the decision latency for estimation of market risk. YARN manages the resource allocation and spawning of map and reduce tasks for each application running 5.5. Since, I am using 3 datasets at a time, YARN automatically assigns 3 Maps and 1 Reduce instance to perform the computations. These numbers could be increased to achieve greater degree of parallelism.

This algorithm design was implemented in Hadoop and it helps in achieving lower latency for the market risk estimation of high frequency big data.

5.4 Spark Implementation

This research work used Spark as the second choice of distributed computing architecture, to study the impact on latency of processing high frequency data. Spark can be setup in multiple modes depending on the scheduler (internal or external) required to run Spark applications [22]:-

1. **Standalone Mode - Single Node** - This setup uses 1 Master and 1 slave node to run applications. The scheduler it uses comes as a part of Spark installation package and does not need an external one.

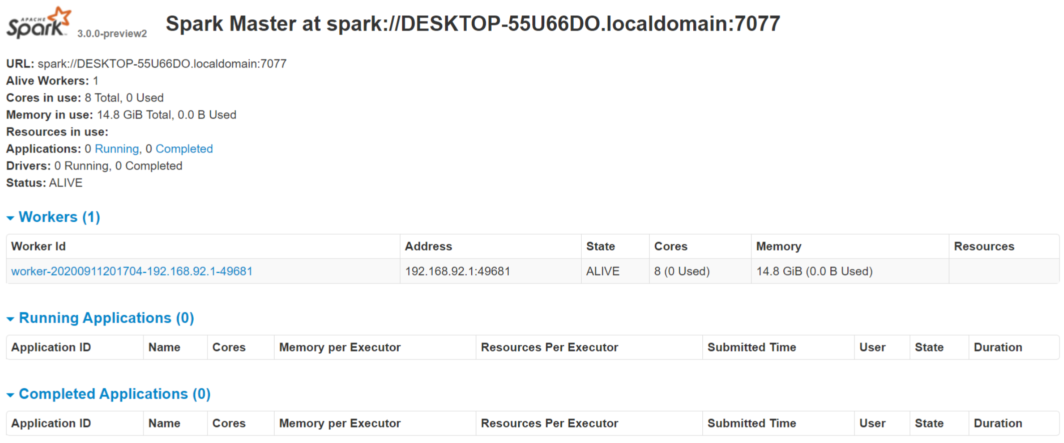
2. **Standalone Mode - Multi Node** - This setup does not require any external scheduler as well to run applications on Spark but has many master-slave nodes and is fully distributed.

3. **Cluster Mode** - This mode could has fully distributed master-slave setup but uses an external scheduler to run applications on Spark.

Spark also has two deployment modes - **local** and **cluster** which have to be specified at run time to let the Spark driver know where the application should run.

For this project, I have used standalone Spark - Single Node setup on my local computer. After installation of Spark, the master and the worker (slave) nodes have to be started to complete the setup. Also, I have used Python Programming language to write the algorithm in Spark for which a special module needed to be used called **PySpark**. [23] It is a wrapper library for Spark, written in Python, which provides all the functionalities available in native Scala language (Spark is written in Scala).

As mentioned in chapter 2, Spark uses a generalised form of Map Reduce called Directed Acyclic Graphs which is implemented by using the core of Spark - Resilient Distributed Datasets or RDDs. Spark provides many high-level APIs for achieving our tasks. Since, my datasets are structured and relational, I have used **Spark SQL APIs** for this task. "Spark SQL is a new module in Apache Spark that integrates relational processing with Spark's functional programming API" [24]. It provides query in both batch and streaming mode. while Spark provides this high level abstraction, the core concept is still RDDs. Sparks splits the data and algorithm into various RDDs automatically and parallelises the computations on them.



3.0.0-preview2 **Spark Master at spark://DESKTOP-55U66DO.localdomain:7077**

URL: spark://DESKTOP-55U66DO.localdomain:7077
Alive Workers: 1
Cores in use: 8 Total, 0 Used
Memory in use: 14.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20200911201704-192.168.92.1-49681	192.168.92.1:49681	ALIVE	8 (0 Used)	14.8 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Figure 5.6: This figure shows the resource allocation to Spark master and worker nodes and the resources used when an application runs.

The concept of algorithm used here is the same as Hadoop. I split the algorithm into two phases - Mapper and Reducer. Mapper reads the csv files of the datasets into spark Dataset APIs and converts stock market time series into log return time-series for each equity which is then sent to reducer to calculate statistical moments of distribution and investment decisions of buying, selling stocks or remaining neutral within 1 minute, before the next row is read. For all the arithmetic computations, I have used the **Windows Aggregation** function provided by Spark SQL library, in Python. These functions take a group of rows, called a "window" and return a single computed/aggregated value for every row in that group. The "lag" function is used to provide a rolling window for calculation of moving average.

Point to be noted here is that Spark works on Lazy Evaluation technique and persists intermediate outputs in memory and does not process them until an action is requested on them. In our case, this action is "standard output" on the screen. If I choose to display the investment decisions for every row in all the 3 datasets, the latency is much higher than if I choose to display only 100 rows.

All the computations are done in a distributed manner and in parallel which helps in reducing the decision latency.

As mentioned, the main algorithm for both Hadoop and Spark is divided into 2 phases,i.e, Mapper and Reducer, which helps parallelise and speed up the execution of the modeling and computations. For statistical moments of distribution, once mean is calculated, rest of the calculations could be parallelised. Trading rules, has to be executed sequentially which adds to the latency. Pseudo Code is given below:

```
1. Mapper
    (a) for i = 1 to n:
    (b) Calculate LogReturns[equity]
    (c) Send to reducer
2. Reducer
    (a) for i = 1 to n:
    (b) calculate mean[equity]
    (c) calculate std[equity], kurtosis[equity], skewness[equity]
    (d) calculate $L$[equity] , $S$[equity]
    (e) compare $S$ with $L$, classify[equity]
```

(f) output

(g) Latency per iteration calculated

The same algorithm is run on Hadoop, Spark and my local system and the latency results, with respect to total execution time, CPU and memory utilization are compared. Point to be noted here is that network latency is not within the scope of this project.

5.5 Conclusion

High frequency and low frequency datasets were downloaded from different sites, analysed and pre-processed. An algorithm was formulated for both Hadoop and Spark for econometrics and computations of market risks. The algorithm was run on the three systems - my local computer, Hadoop and Spark and resource utilization and decision latency were compared for the algorithm run. The results are discussed in the next chapter.

5.6 Key Challenges

I faced these challenges while working on this research:-

1. Finding high-frequency stock market datasets for free was the biggest challenge. There are many finance websites which provide stock market data for free but they are all low-frequency data. Initially, I had daily data, on which I was applying inverse transform sampling to extract intraday data. After looking through many websites, I finally discovered IEX Cloud website from where I got my final data.

2. I had hardware limitations while implementing Spark and Hadoop. Since, I was deploying these on my laptop, I had to restrict the CPU cores to 4 out of 8 and RAM to 8 GB out of 16 GB to prevent my computer from going out of resource.

3. Finding an appropriate baseline to compare the latency results with, was a big challenge. There is not much literature on this topic in the field of finance and stock market trading. So, I used my computer as baseline model.

Chapter 6

Case Study and Results

As mentioned in chapter 1, the main objectives of this research are two-fold:-

- To study the impact of big data processing systems on the latency of high frequency stock market big data
- To estimate and predict market risk

So, the results for these two objectives are discussed in separate sections.

6.1 Latency Study

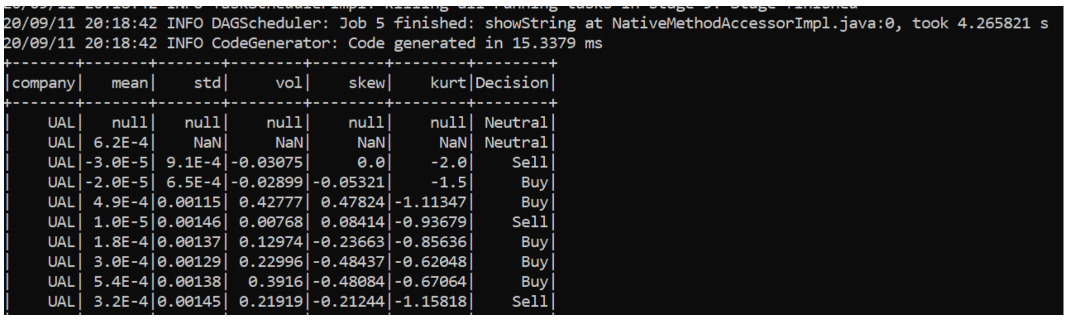
As mentioned in Chapter 3, the baseline model for this latency study is my personal laptop. For the reduction of latency and comparison of outputs, I have used Hadoop Map Reduce in Pseudo-distributed mode and Spark in Standalone mode. The big data frameworks used were setup on my laptop. The configurations of my system are mentioned in the table 6.1:-

CPU Type	CPU speed(Ghz)	CPU Cores	RAM (GB)	OS
Intel core i7	1.99	8	16	windows 10 (64 bit)

Table 6.1: Local computer's configurations

The algorithm mentioned in chapter 4 was run on all the three systems. The final output of the algorithm are the investment decisions of buying/selling the stocks or remaining neutral for every minute of data (as we have 1-minute data). The maximum latency of execution time allowed here is 59 seconds before the next data record is received and delayed decision, post 1 minute would be of no use to the traders.

Output obtained by spark is given in the figure 6.1.



```
20/09/11 20:18:42 INFO DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0, took 4.265821 s
20/09/11 20:18:42 INFO CodeGenerator: Code generated in 15.3379 ms
+-------+-------+-------+--------+--------+--------+--------+
|company|   mean|    std|     vol|    skew|    kurt|Decision|
+-------+-------+-------+--------+--------+--------+--------+
|    UAL|   null|   null|    null|    null|    null| Neutral|
|    UAL| 6.2E-4|    NaN|     NaN|     NaN|     NaN| Neutral|
|    UAL|-3.0E-5| 9.1E-4|-0.03075|     0.0|    -2.0|    Sell|
|    UAL|-2.0E-5| 6.5E-4|-0.02899|-0.05321|    -1.5|     Buy|
|    UAL| 4.9E-4|0.00115| 0.42777| 0.47824|-1.11347|     Buy|
|    UAL| 1.0E-5|0.00146| 0.00768| 0.08414|-0.93679|    Sell|
|    UAL| 1.8E-4|0.00137| 0.12974|-0.23663|-0.85636|     Buy|
|    UAL| 3.0E-4|0.00129| 0.22996|-0.48437|-0.62048|     Buy|
|    UAL| 5.4E-4|0.00138|  0.3916|-0.48084|-0.67064|     Buy|
|    UAL| 3.2E-4|0.00145| 0.21919|-0.21244|-1.15818|    Sell|
```

Figure 6.1: This figure shows the output obtained in spark. Note that spark works on lazy evaluation. Since, it is printing only top 50 rows, the execution time is much less than the total execution time of printing all rows.

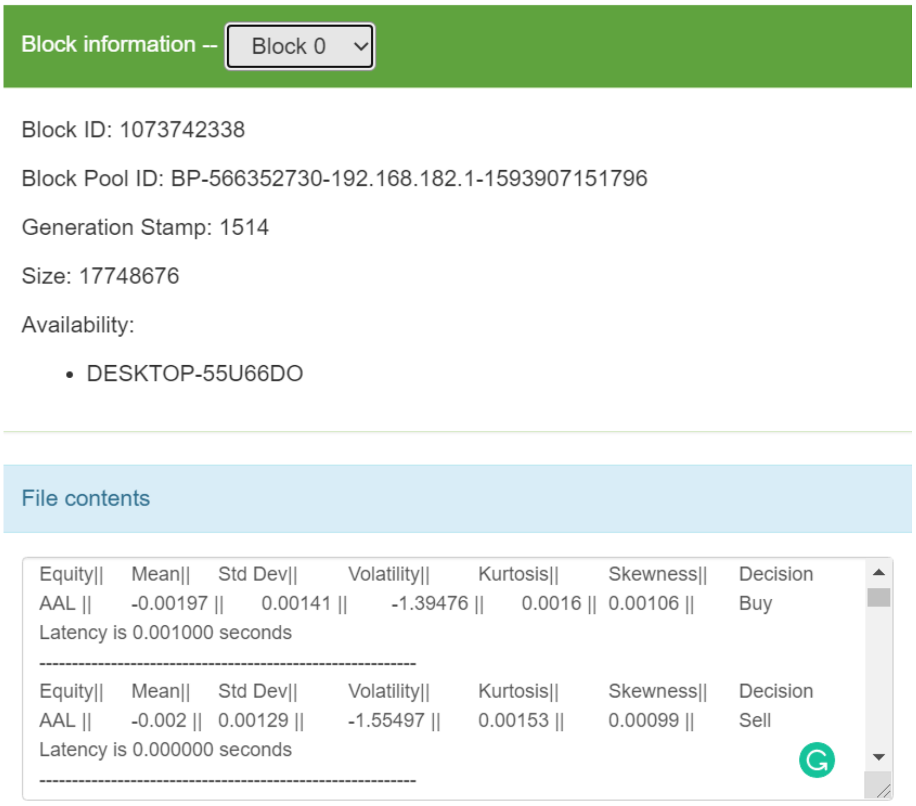
Output obtained by Hadoop is given in the figure 6.2. This file is stored in HDFS.

A comprehensive comparison of the resources' utilisation (for latency study) is given in the figure 6.3.

As seen from the figure 6.3 , Hadoop outperformed both Spark and my computer in terms of resource utilization and execution time. It reduced the latter by 40.6% of the time taken by my computer and by 20% than that by Spark. It reduced the memory utilization to 63% from 70% as used by my computer and 77% as used by Spark. Spark reduced the total execution time with respect to my computer by 25% but increased the memory utilization by 10%.

Hadoop and Spark are using parallelism as explained in chapter 3 to reduce total execution time while my laptop runs the algorithm sequentially. [5] mentions that Hadoop Map Reduce caters well to batch processing jobs while Spark was mainly developed for streaming jobs and was extended further to support batch processing. This could be one reason why Hadoop fared better in reducing the latency (total execution time) than Spark. Spark might also work better with the usage of message ingestion services like Kafka since, it does not use dedicated storage to read files unlike Hadoop's HDFS.

Reason for more memory usage by Spark as opposed to Hadoop and my laptop is due to the fact that Spark uses *Lazy Evaluation* technique. It means that the processing

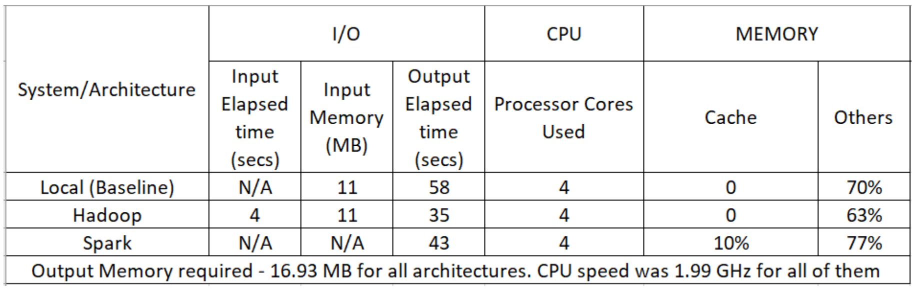


```
Block ID: 1073742338
Block Pool ID: BP-566352730-192.168.182.1-1593907151796
Generation Stamp: 1514
Size: 17748676
Availability:
  • DESKTOP-55U66DO
```

```
Equity||   Mean||      Std Dev||     Volatility||    Kurtosis||    Skewness||    Decision
AAL ||     -0.00197 ||    0.00141 ||    -1.39476 ||    0.0016 ||  0.00106 ||     Buy
Latency is 0.001000 seconds
------------------------------------------------------------
Equity||   Mean||      Std Dev||     Volatility||    Kurtosis||    Skewness||    Decision
AAL ||     -0.002 ||   0.00129 ||    -1.55497 ||     0.00153 ||    0.00099 ||    Sell
Latency is 0.000000 seconds
------------------------------------------------------------
```

Figure 6.2: This figure shows two rows of output of the algorithm on Hadoop UI. It displays equity name which is American Airlines (AAL) in this case, statistical moments of distribution along with the investment decisions and latency per row rounded to 6 places of decimal. The value is 0.001000 seconds means the latency is of the order of milliseconds

of intermediate results of any program running on Spark is not done until the final output is requested by the program. Instead, all the intermediate results are stored in the memory. Only, when an output is requested on the screen or by the program to send to another process, does Spark retrieve the cached intermediate results from the memory and processes them. Generally, Hadoop has more disk I/O overhead than Spark since it uses a dedicated storage - HDFS to store the input, read the input files and store the output, but in processing of my datasets, I did not observe this behaviour. Further experimentation with bigger volume of the datasets or with more time series might reveal the increased disk I/O there.



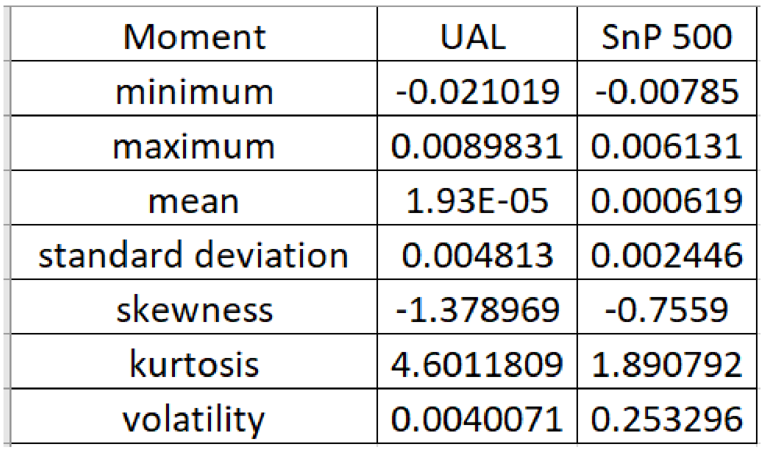
System/Architecture	I/O			CPU	MEMORY	
	Input Elapsed time (secs)	Input Memory (MB)	Output Elapsed time (secs)	Processor Cores Used	Cache	Others
Local (Baseline)	N/A	11	58	4	0	70%
Hadoop	4	11	35	4	0	63%
Spark	N/A	N/A	43	4	10%	77%
Output Memory required - 16.93 MB for all architectures. CPU speed was 1.99 GHz for all of them						

Figure 6.3: Comparison between resources utilization and latency time among my local computer, Hadoop and Spark. Since, the data is intraday 1 minute data, we need to make a decision within 59 seconds. My computer is taking 58 seconds total execution time. So, the critical latency time left is just 1 second to make a decision. With Hadoop 25 seconds are left and with Spark, 15 seconds to make a decision. This is important because if we added another time series or more data, my computer would not be able to cope up within desired time.

6.2 Market Study

The market study was done from the point of view of evaluating market risks. The existing market algorithms were formulated in a novel way to make them usable on big data processing frameworks in order to achieve parallelism and reduce latency. The traders use algorithmic trading for buying and selling stocks in the stock market but don't make use of available big data processing frameworks. The current markets follow a "millisecond environment" where a delayed decision-making of even a few milliseconds could cost the traders huge amounts of money. There are many methods currently for the evaluation of market risks, e.g., machine learning modeling, statistical modeling using moments of distribution, trading rules, etc. The statistical moments of distribution and trading rules were employed in this study to figure out the investment risks and decide when to buy/sell stocks. Moments of distribution give massive insights into the stock market data and the trend of the market in general.

The stock market index is the weighted average of all stocks in the market. As mentioned before, the stock prices of both index and firms are highly correlated and returns of the prices are not completely independent but have very little correlation between them. Thus, return time series were used for further calculations. The descriptive statistics of the stock equity firm - United Airlines(UAL) and stock index - SnP 500 are given in the table 6.4



Moment	UAL	SnP 500
minimum	-0.021019	-0.00785
maximum	0.0089831	0.006131
mean	1.93E-05	0.000619
standard deviation	0.004813	0.002446
skewness	-1.378969	-0.7559
kurtosis	4.6011809	1.890792
volatility	0.0040071	0.253296

Figure 6.4: Descriptive statistics of daily data for UAL and SnP 500

The mean is almost zero which validates the theory of an efficient market where all increases in stock prices are balanced out by decreases in stock prices and hence, market remains stable. The rest of the moments indicate that the distribution of returns of prices is not normal. Skewness values of -1.3 and -0.75 indicate and the data has a fat tail on right side, so more data is concentrated on right side and there are more positive returns than negative. The kurtosis values of 4.6 and 1.89 indicate that the values lie beyond three standard deviations of the mean. The trading rule used here is the moving average rule. This acted as a data smoothing process, which reduced the impact of outliers on the accurate forecasting of the market risk, thus, helping the same, and smoothed out the variations over a period of time. It places equal importance or weight on current as well as historic stock market prices (figure 6.5).

The stock market is highly volatile which makes risk prediction difficult. However, as the frequency, which is the shortest duration for which data is available, of the data decreases, the volatility also reduces.

According to [25], systematic risk of market changes with changes in frequency of the data. So, the risk of minute-by-minute data is different from that of daily data, to weekly, to monthly data. Figures 6.6, 6.7, 6.8 show the changes in systematic or *beta* risk with the different frequency data. I have employed linear regression algorithm using the following formula to calculate the value of beta.

$$UAL(T) = Beta * SnP500(t) + Alpha \tag{6.1}$$

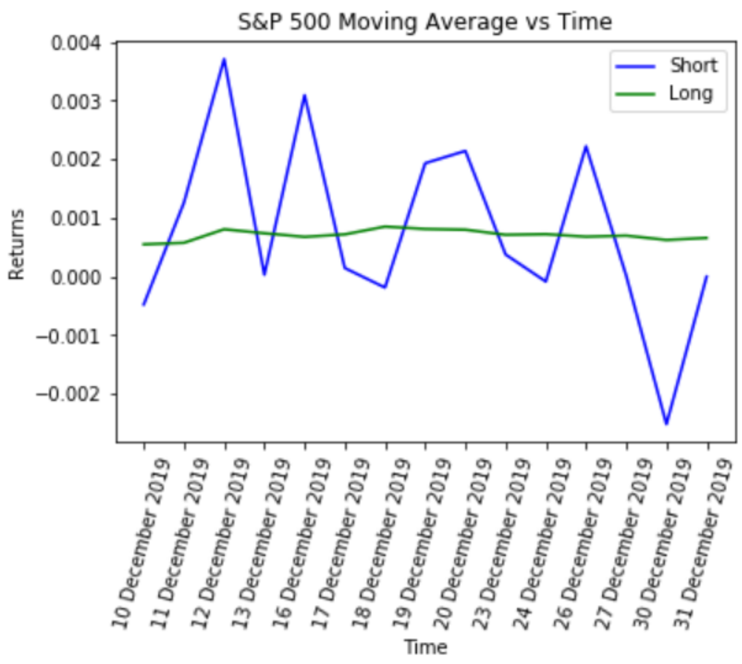


Figure 6.5: Short term average (in blue) vs Long term moving average (in green) values of daily returns. The short term moving averages has higher fluctuations than long term moving averages

Here, $UAL(T)$ refers to the return values of equity firm (UAL), and is the dependent variable in this equation. $SnP500(t)$ refers to the return values of SnP 500 and is the independent variable since it is a stock market index. $Beta$ refers to the slope of the line which signify systematic market risk and $Alpha$, is the intercept. Note that the value of Beta increases from 1.23 to 1.84 (table 6.2) as the time duration is increased from daily to monthly (figures 6.6, 6.7, 6.8). This is because as the duration of return prices are increased, it results in less number of sampled data which results in loss of information.

We also want to check if the high frequency behavior of the three equity firms in the same market (NYSE) and same sector (Aviation) is different to each other. For this study, we employ the same equation 6.1. (Correlation shown in Table 6.3)

If the stocks are highly correlated then we will have no difference in the market risk; but if the stock are not quite correlated then there will be an advantage in investing in one rather than the the other. If beta is close to one then one stock is just as good as another. The beta values for figures 6.6, 6.7, 6.8 are given in the table 6.2. These

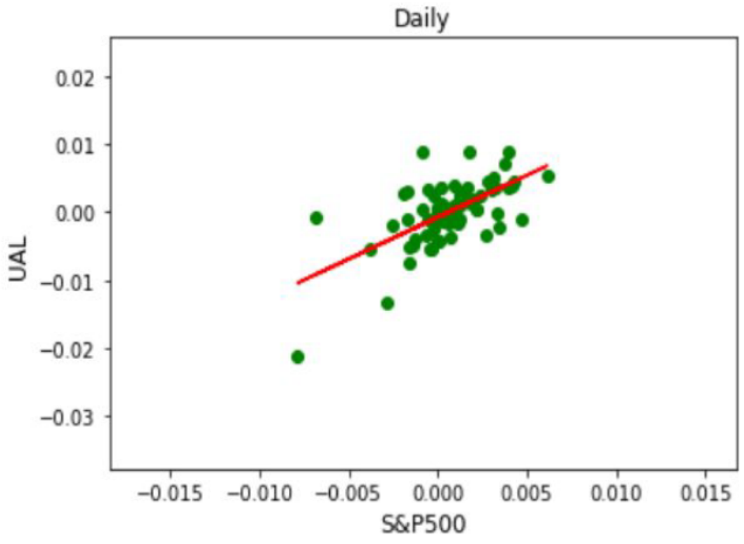


Figure 6.6: Regression Analysis between daily log return values of UAL (firm) and SnP500 (Index) shows correlation between the two.

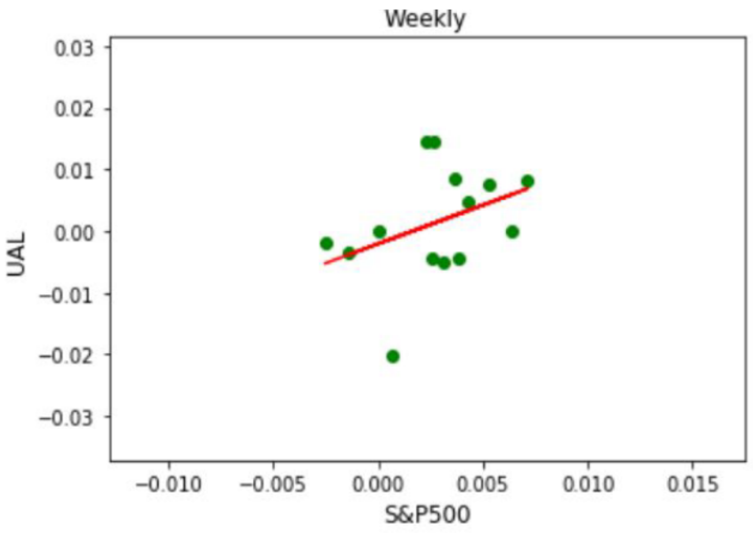


Figure 6.7: Regression Analysis between weekly log return values UAL (firm) and SnP500 (Index) shows correlation between the two.

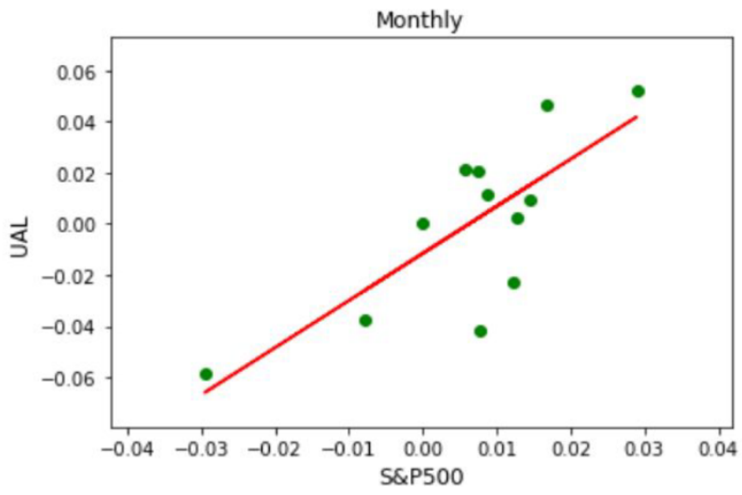


Figure 6.8: Regression Analysis between monthly log return values UAL (firm) and SnP500 (Index) shows correlation between the two.

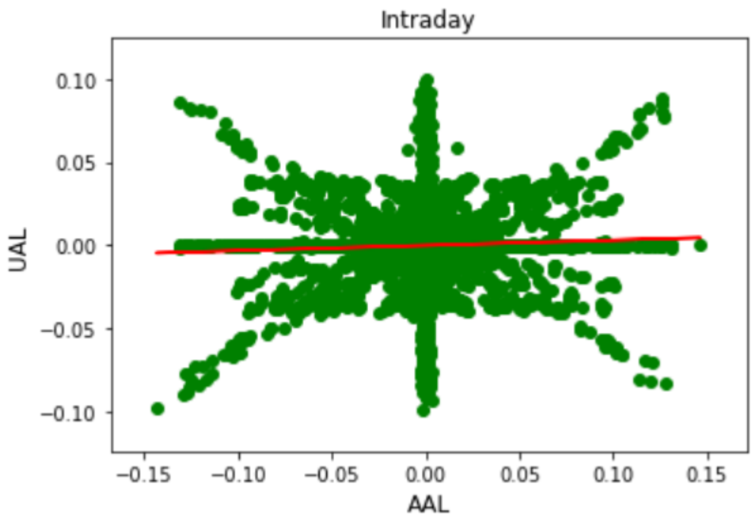


Figure 6.9: Regression Analysis between monthly log return values UAL (firm) and AAL (firm) shows correlation between the two.The data is very scattered, showing that there is almost no correlation

.

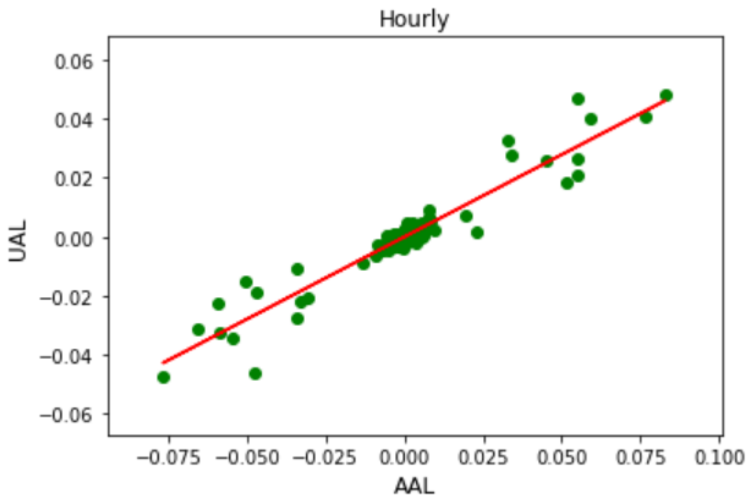


Figure 6.10: Regression Analysis between hourly log return values UAL (firm) and AAL (firm) shows correlation between the two.

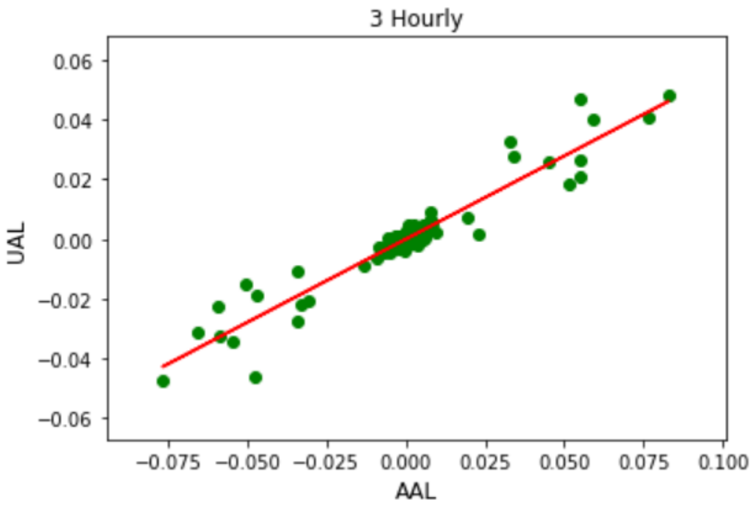


Figure 6.11: Regression Analysis between 3 hours log return values UAL (firm) and AAL (firm) shows correlation between the two.

Frequency	**Beta**	**Alpha**
Daily	1.2316****	-0.0007*
Weekly	1.2528	-0.0020
Monthly	1.8409****	-0.0115*

Table 6.2: Correlation between **UAL (firm)** and **market (SnP 500)**. 1 asterisk represents confidence interval is greater than 85%. 2 asterisks represent confidence interval is greater than 90%, 3 asterisks represent confidence interval greater than 95%, 4 asterisks represent confidence interval greater than 99.99%

Frequency	**Beta**	**Alpha**
Intraday	0.3392	0.0002
Hourly	0.5706****	-0.0003
3 Hourly	0.5706****	-0.0003

Table 6.3: Correlation between **UAL (firm)** and **firm (AAL)**. 1 asterisk represents confidence interval is greater than 85%. 2 asterisks represent confidence interval is greater than 90%, 3 asterisks represent confidence interval greater than 95%, 4 asterisks represent confidence interval greater than 99.99%

compare the firm United Airlines with the market SnP 500. The beta values for figures 6.9, 6.10, 6.11 are given in the table 6.3. These compare the firm United Airlines with the firm American Airlines. Based on these values the trader could make an informed decision about which equity he wants to invest in.

6.3 Conclusion

This section discussed the results of the two case studies undertaken for this project - the latency study and the market study, in detail.

Chapter 7

Conclusion and Future Work

This chapter concludes this research and talks about the limitations of it and future work which could be done to improve upon and extend this research work further.

7.1 Conclusion

There are applications in the world which have an intrinsic requirement for low decision latency in order to have real-time control over their decision-making. They try to reduce the latency by using high-powered computing systems. The evolution of technology and rise in big data has given popularity to distributed computing and parallel processing.

The main aim of this study was to find the effect of low-latency architecture on high frequency stock market data. Literature review showed that there is a gap between the computing systems used for processing and storage of high frequency big data used in trading in finance and the usage of big data computing systems. This research aimed to address this gap and reduce the decision latency of market risk estimation and forecasting using the parallel processing, efficient storage and retrieval capabilities on intraday 1-minute stock market data. The big data technologies used here are Hadoop MapReduce and Spark.

The data used for the case study was of two types - high frequency intraday 1-minute stock market data for 3 US stock equity firms (3 airlines) and low frequency daily, weekly and monthly data for a US stock market index. The high frequency data was collected from IEXCloud.io and low frequency data was collected from Yahoo!

finance. The data was cleaned up before application of the algorithm.

The research had two components - latency study and market study. An algorithm was developed for big data systems to take advantage of the distributed and parallel computing capabilities of pseudo-distributed deployment of Hadoop Map Reduce and standalone Spark, which were setup on my local computer. The algorithm was divided into 2 phases called Mapper and Reducer, and the final output obtained were the investment decisions of buying the stock, selling it or remaining neutral. Seeing the highly correlated stock prices, calculation of Log return series was done in the mapper phase to reduce this correlation. Statistical moments of distribution and financial trading rules were applied on the return time-series in the reducer phase to study the distribution of the data and to make the investment decisions. A further analysis of the United States airline sector was done by using regression analysis of the equity firm and the market index for the market study.

A comparison was done of latency performance among Hadoop , Spark and my local computer, where the latter acted as a baseline. The latency study here focused on the CPU, memory utilization and total execution time of the algorithm, on all three systems. The algorithm on my computer ran sequentially and took the longest time to run. Hadoop outperformed both Spark and my computer in terms of resource utilization and execution time. It reduced the latter by 40.6% of the time taken by my computer and by 20% than that by Spark. It reduced the memory utilization to 63% from 70% as used by my computer and 77% as used by Spark. Spark reduced the total execution time with respect to my computer by 25% but increased the memory utilization by 10%.

The results of the market study revealed that the more the time series is smoothed, the lesser the fluctuations and volatility in the stock market data and the more the equity firms start behaving like the market. This study shows that in the longer run, there is lesser market investment risk than in the short term.

The algorithm developed for latency study could be extrapolated to other applications like healthcare, weather systems, where there is need for low decision latency.

7.2 Future Work

While this research shows promising results for latency study when comparing the performances of Hadoop and Spark, there are a few limitations that could be improved upon.

Firstly, this research does not study network latency. Spark and Hadoop were deployed on my computer locally. Network latency is an important parameter while studying computing systems to reduce decision latency in an enterprise environment. In such kind of environment, fault tolerance, data recovery and no data loss are important factors for which fully-distributed mode of Hadoop and Spark would be used providing data redundancy and high availability. In order to provide these benefits, they would have to be deployed on different physical or virtual machines or enterprises would use cloud computing, which would add to network latency. Thus, dealing with it becomes important. Secondly, further research about the impact of using multiple master-slave nodes, for both Hadoop and Spark, on performance and latency, could be done.

For the datasets used in this research, Hadoop outperformed Spark. Testing could be done on other high frequency datasets and time series to solidify the results. Impact on live streaming data could also be studied using the big data technologies. It is said that Spark streaming performs better than Hadoop on streaming data.

In addition to these, more complex non-linear models and statistical methods could be used to estimate market risks. These methods would add to the latency of the calculations. Their effect could be studied on the decision latency using distributed big data systems.

Trading venues and stock market indices use co-location cabinets to reduce latency while doing algorithmic trading. A thorough comparison needs to be done with the latency results obtained by this method versus the one used in this research to be able to exploit it completely.

Bibliography

[1] “Image from: https://coservit.com/servicenav/en/big-data-the-5vs/.”

[2] “Image from: http://www.sunlab.org/teaching/cse8803/fall2016/lab/mapreduce-basic/.”

[3] “Hadoop architecture guide,” 2019.

[4] “Image from: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/yarn.html.”

[5] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, and F. Herrera, “Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce,” *Information Fusion*, vol. 42, pp. 51–61, 2018.

[6] “https://iexcloud.io/ (site visited 7 june 2020).”

[7] “https://in.finance.yahoo.com/ (site visited 18 june 2020).”

[8] J. Hasbrouck and G. Saar, “Low-latency trading,” *Journal of Financial Markets*, vol. 16, no. 4, pp. 646–679, 2013.

[9] A. E. Breiholz, K. M. Kronfeld, and K. L. Walling, “Weather radar system and method with latency compensation for data link weather information,” June 27 2017. US Patent 9,689,984.

[10] A. Frino, V. Mollica, R. I. Webb, and S. Zhang, “The impact of latency sensitive trading on high frequency arbitrage opportunities,” *Pacific-Basin Finance Journal*, vol. 45, pp. 91–102, 2017.

[11] "Definition from: https://dictionary.cambridge.org/dictionary/english/big-data (site visited 7th september 2020)."

[12] F. Almeida, "Big data: Concept, potentialities and vulnerabilities," *Emerging Science Journal*, vol. 2, no. 1, pp. 1–10, 2018.

[13] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita, "Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf.," in *INNS Conference on Big Data*, vol. 8, p. 121, 2015.

[14] W. Huang, L. Meng, D. Zhang, and W. Zhang, "In-memory parallel processing of massive remotely sensed data using an apache spark on hadoop yarn model," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 1, pp. 3–19, 2016.

[15] X.-P. S. Zhang and F. Wang, "Signal processing for finance, economics, and marketing: Concepts, framework, and big data applications," *IEEE Signal Processing Magazine*, vol. 34, no. 3, pp. 14–35, 2017.

[16] D. Creal, "A survey of sequential monte carlo methods for economics and finance," *Econometric reviews*, vol. 31, no. 3, pp. 245–296, 2012.

[17] S. J. Taylor, *Asset price dynamics, volatility, and prediction.* Princeton university press, 2011.

[18] X. Tian, R. Han, L. Wang, G. Lu, and J. Zhan, "Latency critical big data computing in finance," *The Journal of Finance and Data Science*, vol. 1, no. 1, pp. 33–41, 2015.

[19] "Taken from: https://blogs.sap.com/2014/03/25/smart-trading-adding-precision-to-big-data-inspired-trading-strategies/."

[20] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[21] "Taken from: https://www.scipy.org/."

[22] "https://spark.apache.org/docs/latest/cluster-overview.html."

[23] "https://spark.apache.org/docs/latest/api/python/index.html."

[24] "https://jaceklaskowski.gitbooks.io/mastering-spark-sql/content/spark-sql-functions-windows.html."

[25] R. Gençay, F. Selçuk, and B. Whitcher, "Systematic risk and timescales," *Quantitative Finance*, vol. 3, no. 2, pp. 108–116, 2003.