Real Time Fraud Detection in Financial Markets

Kaarthik Elangovan

September 2013

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Advanced Computing

Computing Science and Mathematics
University of Stirling
Abstract

Business Problem: The Financial Markets are seeing an increase in fraudulent activity every year and it costs millions of pounds for many institutions. In spite of various governing bodies enforcing strict regulations, fraudulent activities continue to happen. Most of this fraudulent activity are discovered many months or years later when the governing bodies conduct an audit on historical trades. Institutions/brokers who provide various trade services if found guilty of not detecting such activities end up paying huge penalties. The solution is to provide a software system that will detect fraudulent activity in a real time fashion to take action sooner than before and avoid such penalties.

Technology Problem: In today’s digital world, there is a burst in data and standalone technologies are reaching their limits to process data and keep up with the speed with which data is being generated. According to a survey conducted by IBM [18] ‘Every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone’

There is a need for big data technologies, which use the power of cluster computing using commodity hardware to process huge amounts of data at speeds which are impossible with single nodes. Financial markets have systems which generate huge amounts of data and financial services institutions are starting to migrate to the big data technology.

Objectives: To create a system that will use big data, cluster computing and in memory processing technologies and solve the business problem of detecting fraudulent activity on a real time basis.

Methodology: The system will use machine learning clustering algorithms to identify patterns in historical trades for each trader. The patterns/clusters will be used to determine any change in trading behaviour of each trader by comparing the clusters with the new trading activity of that trader on a real time basis and raise alerts to the business personal for them to determine the validity of alerts. The validation will be used to make the system learn again and make better decisions. The system should evolve on a regular basis.

Achievements: The built system is capable of learning from historical trades, identify patterns in the data. The system will create clusters for each trader and compare new trading activity against the clusters to give a verbose output on the differences real time.
Attestation

I understand the nature of plagiarism, and I am aware of the University’s policy on this.

I certify that this dissertation reports original work by me during my University project except for the following,

The EM and K-Means algorithm logic in the machine learning program is same as that of Weka [19] but implementation is done using Spark data structures and map/reduce functions to run the algorithm in a distributed environment. Weka implementation runs only on a single JVM in a standalone machine.

Signature                          Date
Acknowledgements

I would like to thank,

- My supervisor Kevin Swingler for his valuable inputs in carrying out this project. The most important of all is when Kevin suggested some techniques to create patterns in data and that gave direction for rest of the project as it was important to have pattern in data to test that the entire system worked as expected.

- Dr Mario Kolberg for his time and suggestions.

- My wife, who from India spoke to me regularly and reminded me to stay focused and finish the project on time.

- My family and friends for their support.
Table of Contents

Abstract .......................................................................................................................... i
Attestation .................................................................................................................. ii
Acknowledgements .................................................................................................. iii
Table of Contents ...................................................................................................... iv
List of Figures ............................................................................................................ vii

1 Introduction .......................................................................................................... 1
  1.1 Background and Context ................................................................................. 1
  1.1.1 Business Application .................................................................................. 1
  1.1.2 Technology Application ............................................................................. 1
  1.2 Scope and Objectives ....................................................................................... 2
  1.3 Achievements .................................................................................................. 2
    1.3.1 Generating Trading Data with Patterns ................................................. 2
    1.3.2 Setting up Big Data/Cluster computing frameworks in Standalone Mode . 3
    1.3.3 Trade Publisher to Message Queue ......................................................... 3
    1.3.4 Finding Patterns in Trading Activity (Machine Learning) ....................... 3
    1.3.5 Finding Differences in New Trading Activity (Fraud Detection) ............ 4
    1.3.6 Learning Outcome ................................................................................... 4
  1.4 Overview of Dissertation ................................................................................. 4

2 State-of-The-Art ................................................................................................... 5
  2.1 Introduction to Big Data and Cluster Computing ........................................... 5
    2.1.1 What is Big Data? ................................................................................... 5
    2.1.2 Why Big Data? ...................................................................................... 6
    2.1.3 Analysing Data ....................................................................................... 8
    2.1.4 Machine Learning in Data Analysis ....................................................... 8
    2.1.5 Combining Machine Learning and Big Data using Cluster Computing .... 8
    2.1.6 Overview of Hadoop, Spark and Kafka ................................................. 10
    2.1.7 Hadoop ................................................................................................. 10
    2.1.8 HDFS ................................................................................................... 11
    2.1.9 Spark ..................................................................................................... 13
    2.1.10 Why Spark? ....................................................................................... 14
    2.1.11 Kafka .................................................................................................. 15
    2.1.12 Big Data in Financial Markets ............................................................ 16
    2.1.13 Clustering (Unsupervised Machine Learning) in Financial Markets .... 17
    2.1.14 Fraud detection in Financial Markets using Clustering ...................... 17

- iv -
List of Figures

Figure 1. Big Data .................................................................................................................. 6
Figure 2. Scale of Big Data .............................................................................................. 7
Figure 3. Map/Reduce Operations .................................................................................. 11
Figure 4. Hadoop Architecture ..................................................................................... 12
Figure 5. Spark Architecture ......................................................................................... 13
Figure 6. Spark/Hadoop Machine Learning Experiment ............................................. 15
Figure 7. Apache Kafka Architecture ........................................................................... 16
Figure 8. Use Case Diagram .......................................................................................... 24
Figure 9. High Level System Design ........................................................................... 27
Figure 10. FMTradePublisher Class Diagram ............................................................. 28
Figure 11. FMTradeSubscriber Class Diagram ............................................................. 28
Figure 12. FMMachineLearning Class Diagram ............................................................ 29
Figure 13. FMFraudDetection Class Diagram ............................................................... 30
Figure 14. Banzai GUI ..................................................................................................... 32
Figure 15. FIXimulator GUI ............................................................................................ 33
Figure 16. FMTradeSimulator Flow-Chart .................................................................. 33
Figure 17. FMTradePublisher Flow-Chart ................................................................. 34
Figure 18. FMTradeSubscriber Flow-Chart ................................................................. 36
Figure 19. FMMachineLearning Flow-Chart ................................................................ 38
Figure 20. FMFraudDetection Flow-Chart .................................................................. 40
Figure 21. Normal Distribution of Order Quantity ....................................................... 41
Figure 22. HDFS Project Directory Structure .............................................................. 43
Figure 23. HDFS Trade File ........................................................................................... 43
Figure 24. System Configuration .................................................................................... 52
Figure 25. System Resources .......................................................................................... 52
1 Introduction

1.1 Background and Context

1.1.1 Business Application

Fraud is a multi-million pounds business. It costs many firms millions of pounds in losses and fines every year and it is increasing all the time. There are efforts being made by the investment banks, brokerage firms, hedge funds and other regulatory bodies to utilize IT systems to determine any suspected transactions and also monitor traders risk limits. There are still rogue trading activities which bypass these checks and costs massive losses and fines. The use of IT systems to monitor such activity and raise alerts at the earliest is the major business goal for many firms when comes to investing on fraud detection systems. Since the market meltdown in 2008, there has been an increase in usage of advanced techniques and better fraud detection mechanisms. The paper on fraud detection published in 2012, ref here [37], shows that there has been an increase in research on the topic of fraud detection in financial markets since 2008. One of the major effects of fraud is on the world economy as a whole. It is a domino effect, lots of big economies in the world have major dependency on the financial sector and any impact on this sector affects the economy causing job loses/increased unemployment rates across the economy. For E.g. this doc [36] shows the importance of investment banks in German economy. The other effect it has on individual firms that are involved in such activity is the loss of reputation and trust from its clients. Almost all major investment banks on the street have been affected by this over the years. This shows the importance of building a better fraud detection system. These links show a list of fraudulent activities in the financial markets through trading [28], [29], [30], [31], [32], [33], [34], [35].

1.1.2 Technology Application

Technology innovations continue to prove some assumptions made over the years are wrong. One such is Cluster Computing, some tasks once assumed that it was possible to execute only on a main frame or super computer now can be achieved by putting together a collection of commodity hardware into a cluster to achieve the same task with low cost and low processing power. There has been an increase in the open source and proprietary frameworks that utilize the cluster computing technology. This technology has given way to do many types of data analytics which was simply ignored previously because of the processing power that was required. The increase in the data burst lately also has brought in many innovations in data management with the introduction of Big Data and NoSQL databases for managing huge volumes of data.
In the financial sector, the amounts of transactions are increasing, with many investment banks making global markets available to local customers. There are many algorithmic and high frequency trading platforms which are capable of generating many thousands of transactions per second. There are exchanges which can handle many millions of transactions per second. This requires usage of Big Data and Cluster Computing technologies.

1.2 Scope and Objectives

The scope of the project is to create an IT system that can detect fraud in the financial market in real time. The system should be able to assess historical trading activity of traders and find patterns. Once the patterns are determined, it will be used to make decisions on new trading activity of those traders to check if the activities fit the pattern or there is deviation from the pattern. The system should raise an alarm on finding differences and send it to a business analyst who can further analyse to determine the validity of the alert. The decision from the business analyst will be used to make the system relearn and make better decisions in future. The system should be capable of continuous learning and configurable to add finer checks to detect fraud.

1.3 Achievements

1.3.1 Generating Trading Data with Patterns

The unavailability of real data required generating trading activity using a trade generator. On doing research, the Banzai/FIXimulator tool was chosen; it acts like a front end trading interface (Banzai) and a stock market simulator (FIXimulator). It is a Java based buy/sell side FIX trading application built on open source QuickFIX/J FIX engine. FIX (Financial Information Exchange) is a protocol used by many financial institutions to exchange data. This is mainly used as the messaging protocol to send orders/receive trades from many stock exchanges. The Banzai/FIXimulator was released under GNU/GPL license, it is distributed with source and free to modify/re-distribute. The Banzai GUI connects to the FIXimulator, the FIXimulator can be configured to auto execute orders randomly.

For the purpose of this project, this tool was modified to do the following,

- Support for Account FIX field in both Banzai and FIXimulator
  
  The Account field will be used as the unique identifier for a trader.

- Setting up products XML and loading in Banzai for Automatic Order Generation
  
  The products file will be used to generate random orders on selected exchange/sector.

- Saving/Loading Account Pattern Settings in Banzai
The randomization settings for each account can be saved as a file and loaded back again to regenerate orders.

- Support for Random Order Generation with following Fields in Banzai, Exchange and Sector – Percentage Split
  Order Quantity – Mean/Standard Deviation
  Price – Min/Max

  These values can be set in the randomization section of the Banzai GUI to generate random orders with the specified settings

- Adding Drop Copy Session to FIXimulator

  This session will be used to take the FIX messages and publish them on a message queue for real time processing.

1.3.2 Setting up Big Data/Cluster computing frameworks in Standalone Mode

The Hadoop, Spark and Kafka frameworks that are used in this project all need to be configured to run in a standalone mode.

A Linux user id “hduser” was created to setup all the frameworks that are required for this project work.

More on Hadoop, Spark and Kafka can be found here 2.1.6 and some references to do the installation of these frameworks here [5], [6], [13].

1.3.3 Trade Publisher to Message Queue

The messaging queue system publisher was created to connect to the FIXimulator as a drop copy session and get all the FIX trade messages. These messages are published to the message queue for real time access and historical trade generation. Details on the Fix Connector and Trade Publisher can be found here 5.2.

1.3.4 Finding Patterns in Trading Activity (Machine Learning)

The historical trading activity generated was stored on the Distributed File System using a Trade Subscriber/Aggregator application, the trading activity for every trader was then loaded by the application to run machine learning algorithms and find patterns successfully. The pattern that was generated using the trade generator was exactly determined by the machine learning application. Further details on the Machine Learning implementation can be found here 5.4.
1.3.5 Finding Differences in New Trading Activity (Fraud Detection)

The machine learning output for each trader was stored on the distributed file system and used by the real time streaming consumer application to check differences in the real time trades against the machine learning output for each trader. It successfully displays a verbose output of the differences in the new trading activity against the previously determined pattern for that trader.

1.3.6 Learning Outcome

- Some introduction to Scala/Functional programming concepts.

1.4 Overview of Dissertation

Chapter 1: Introduction

A brief background to the importance of fraud detection systems and big data technologies is provided. It also defines the scope of the design and outlines the achievements.

Chapter 2: State-of-The-Art

Introduction to various big data, cluster computing technologies, machine learning used in the project.

Chapter 3: Project Approach

It covers the approach taken to implement the project work.

Chapter 4: Design

Presents use cases for the system and associated overview of the class diagrams corresponding to various applications in the system.

Chapter 5: Implementation

Highlights the building blocks of the system and shows how each application is implemented technically using various frameworks and also a higher level flow is described.

Chapter 6: Testing

The testing of various applications and approach is described in this chapter.

Chapter 7: Conclusion

Outlines what has been achieved and how the design can be developed in the future.
2 State-of-The-Art

2.1 Introduction to Big Data and Cluster Computing

2.1.1 What is Big Data?

Data is the lowest level of abstraction from which information and then knowledge are derived. It is a major part of every organization and plays very important role in innovation and growth. Data being captured is on the raise because of variety in sources of data and the storage is cheaper. It is increasingly difficult to derive knowledge from this data due to the size. The traditional RDBMS technology when dealing with such variety of data and high volume affects its performance. Such limitations in turn affect some major organizations in the internet search, finance and business informatics sector who deal with lots of data. Big Data was the solution to this. Big data is large amounts of structured/unstructured data that can be captured, stored, analysed and communicated. The size of Big Data is beyond the ability of a typical database software tool to process. This is a moving target, as the technology advances, the size of the data to be classified as Big Data also will increase.

Big data usually is described using the “3V’s” model,

- Volume
- Velocity
- Variety

These simply correspond to the characteristics which Big Data possesses. The volume is the amounts of data, velocity corresponds to the rate at which this volume of data is generated and variety is multiple sources from which both structured and unstructured data is captured.

The volume of data is growing at an exponential rate; the figure below shows some statistics on the amount of data that is out there,
2.1.2 Why Big Data?

The traditional use of IT in the industry has contributed to productivity growth over the years and Big Data is considered as the next frontier. Big Data adds a lot of value to the business, most importantly,

- The amount of information that many companies process is increasing

  The customer base for many companies is growing and it is increasing the necessity for a competitive advantage over its peers.

- Adding transparency to the business, relevant data easily accessible across multiple business units.
Big Data is stored both in unstructured and structured ways and the data stores are huge making it easier to access from a single point of source instead of keeping multiple copies of the same data in different formats across an organization.

- Customize the actions to various segments of the populations.

Big data provides ways to segment large customer base and thus makes it easier to customize actions for various segments.

- Applying many automated algorithms, reduce human decision making.

The human decision making is reduced since the human capability to visualize or to make decisions on the amount of data is limited, many automated algorithms mainly Machine Learning algorithms are applied to make better decisions.

- Helps in creating new business models, services and products.

The knowledge derived from Big Data can be used to create new business models, services and new products.

- It also adds an edge to the business over its competitors.

The use of Big Data in an organization also gives an edge over its competitors, since the potential to extract value from it and reach out to customers with more knowledge from data is huge. Scale of Big Data over the years shown below,

![Big Data Will Scale To Exabytes](http://www.syoncloud.com/big_data_technology_overview)

**Figure 2. Scale of Big Data**

(Courtesy: [http://www.syoncloud.com/big_data_technology_overview](http://www.syoncloud.com/big_data_technology_overview))
For Big data to add value to the business, the most important thing is to extract knowledge through analysis.

2.1.3 Analysing Data

There are two major techniques to analyse data,

- Statistics [17]
  
  It is a branch of mathematics that is concerned with collection, organization and interpretation of data.

- Machine Learning [17]
  
  It is a branch of computer science/artificial intelligence that is concerned with the construction and study of systems that can learn from data.

2.1.4 Machine Learning in Data Analysis

Machine learning can be further divided into two types,

- Supervised Learning [17]
  
  Set of machine learning techniques that infer a function or relationship from a set of training data. An example of supervised learning is “Classification”

  Classification is a technique to identify the categories in which new data points belong based on a training set containing data points that have already been categorized.

- Unsupervised Learning [17]
  
  Set of machine learning techniques that find hidden structure in unlabelled data. An example of unsupervised learning is “Clustering”

  Clustering is a technique of classifying objects that splits a diverse group into smaller groups of similar objects whose characteristics of similarity not known in advance.

2.1.5 Combining Machine Learning and Big Data using Cluster Computing

With the technology getting cheaper, the storage of Big Data is not a concern anymore. The issue is only with the analysis of large amounts of data using machine learning/statistical methods in a manageable time. The solution was to distribute the work load and data to speed up the processing. Internet giant Google introduced the concepts of GFS (Google File System), Big Table and MapReduce process, which is based on distributing/parallel
programming model. The queries/data are split and distributed across parallel nodes stored in Big Table on GFS and processed in parallel (Map). The results are then gathered and delivered (Reduce). This innovation led to many distributed/cluster computing technologies.

Some of the big data and cluster computing technologies that are widely used in the industry are provided by both open source communities like Apache and many proprietary companies, Propriety solutions include Big Data and Cluster Computing systems from,

- EMC
- Oracle
- Terradata
- SAP
- SAS
- IBM
- Pentaho

Open source Cluster Computing technologies from Apache and other communities include,

- Spark (University of California, Berkeley)
- Kafka (Apache)
- Storm (Twitter)
- Akka (Typesafe)

Some Big Data databases include,

- Hadoop (Apache)
- Cassandra (Apache)
- Mongo DB (10gen)

There are currently over 150 Big Data databases which do not fall under the traditional RDBMS databases. Hadoop and Cassandra are wide column stores, wide column databases stores data tables as sections of columns of data rather than as rows of data.

Similarly Mongo DB is a document store, in which common semi structured documents like XML, JSON formatted data is stored. All of these technologies have very similar MapReduce concepts and distributed parallel computing techniques.
Other type of Big Data databases include Key-Value stores, Graph databases, Multimodal databases and Object databases.

Based on the problem at hand, various technologies can be chosen and the problem could be solved. There are no perfect technologies, but the one that solves most of the problem in an efficient way can be chosen as the best for the problem. Since there are many open source and proprietary software’s, carrying out performance tests using multiple technologies is another way of determining what works best. Many of these technologies have emerged in the last couple of years or so, and thus these are still being adopted by the industry and being tried out by many companies.

2.1.6 Overview of Hadoop, Spark and Kafka

Most of these open source projects are developed using Java. Since these open source projects are memory intense due to the parallel computing nature and also in general have a large code base, Java provides various advantages like,

- Easier code management using Object Oriented Principles
- Automatic memory management with Garbage Collection
- Many built in and other open source libraries.
- Larger community of Java developers.
- Ability to combine other core languages like C/C++ using JNI for system processes.

Also, there are many functional programming languages like Clojure and Scala, which support object oriented programming, are built on top of the Java Virtual Machine. The programming is made simpler with flexible syntax, loosely typed variables and other many functional programming concepts like anonymous functions, closures etc. The code gets compiled to Java Byte code; this can then be run on any JVM. The reach of such languages which combines the power of object oriented and functional programming concepts is huge and there by pulling in a bigger community of developers contributing to such open source projects.

2.1.7 Hadoop

Hadoop [11] is a framework for running applications on large clusters built of commodity hardware. It was started by Yahoo and a clone to Google's Map Reduce process. This was later adopted by Apache.

Commodity hardware is computer system that is easily and affordably available. Typically it is a low performance system that is capable of running common operating systems without requesting any special devices or equipment.
Clusters are a set of loosely connected or tightly connected commodity hardware systems that work together so that it can be viewed as a single system.

The Hadoop framework provides the applications with both reliability and motion in data. Hadoop implements a paradigm called as MapReduce, where the application is split into many fragments of work, each of which may be executed or re-executed on any node in the cluster.

2.1.8 HDFS

In addition, it provides a distributed file system (HDFS) [11] that stores data in blocks of predefined size and split into chunks across the nodes, providing very high bandwidth access across the cluster. Both Map/Reduce and the distributed file system are designed so that node failures are automatically handled by the framework.

Map/Reduce is a set of code and systems for parsing and building large data sets. A map function generates a bunch of key value pairs from the input data and this data is then reduced by a reduce function to merge all the values associated with the equivalent keys. Programs are automatically parallelized and executed on a run time system that manages partitioning of the input data, scheduling execution of map/reduce tasks and managing communication including recovery from node failures. Also, this mechanism is done under the hood by the Hadoop core system, helping programmers who have no experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

The Map/Reduce process shown below,

![Map/Reduce Operations](http://spark.incubator.apache.org/talks/overview.pdf)
The below figure shows the overall Hadoop architecture,

![Hadoop Architecture](http://en.wikipedia.org/wiki/File:Hadoop_1.png)

**Figure 4. Hadoop Architecture**


There are 5 main processes in Hadoop,

- Job tracker
- Task tracker
- Name node
- Secondary name node
- Data node

In this, the Job Tracker and Task Tracker are part of the Map Reduce processing. Name Node, Secondary Name Node and Data Nodes are part of the HDFS.

The Job Tracker keeps track of the Map Reduce programs, the entire map reduce program.

The Task Tracker keeps track of the individual tasks which are the Map and Reduce tasks.

The Name node and data nodes are used for managing data on the HDFS. The data is split into blocks or predefined size (E.g. 64 MB) and distributed across the cluster and managed by HDFS.
The Hadoop workflow includes,

- Loading data into the cluster on HDFS
- Analysing the data using Map Reduce programs
- Storing the results back in the cluster on HDFS
- Finally reading the results from the HDFS.

And in order to store data on HDFS, the process looking to store data on HDFS will consult the Name node, write blocks of data to one Data node, and then the data node replicates the block for redundancy and recovery.

Hadoop can be used for both running MapReduce programs for batch processing of large data sets or just a data store using HDFS. With the help of other frameworks like Spark, various data analytics/mining can be done much faster.

2.1.9 **Spark**

Spark was initially developed by the Algorithms Machines People Lab at the University of California, Berkeley. This is now an incubator project under Apache.

Spark is a cluster computing framework that makes data analytics faster to run and also to write to distributed file systems. Spark offers a general execution model that can optimize and support in memory computing. This allows it to query data faster than other disk based engines like the Hadoop. Spark was mainly developed to run iterative algorithms like those used in Machine Learning and also interactive Data Mining applications.

The figure below shows the architecture of the Spark framework,

![Spark Architecture](http://spark.incubator.apache.org/talks/overview.pdf)
The Spark cluster of nodes can be configured to run as a separate cluster or can be run on other clustering frameworks like Hadoop. Once the spark cluster is configured with number of worker nodes, the spark cluster can be started. The master/driver process is the one that launches multiple worker programs. They read data blocks from a distributed file system like HDFS and then cache them in memory to run any iterative machine learning algorithms. The driver program defines one or more objects called as RDDs (Resilient Distributed Datasets).

Resilient Distributed Datasets is a read only, partitioned collection of records. RDDs can be only created through deterministic operations on either a dataset in stable storage or from other existing RDDs. These operations are called transformations to differentiate them from other operations that can be applied on RDDs. Some example Transformations includes methods like map, reduce, filter, groupBy and join. There are two aspects of RDDs that can be controlled namely caching and partitioning. RDDs can be cached in memory making iterative operations on these objects faster. Also RDDs partition information can be specified which allows to group records in a RDD closer to each other on the same machine to speed up operations.

The Spark programs use a very similar map/reduce task that are used in Hadoop to run analytics on the large data sets. The difference is that Hadoop map/reduce programs are mainly used to run in batch mode, whereas Spark programs can be used to run on real time data as well as batch mode since it uses the concepts of caching data in memory. Spark infrastructure also has mechanism built in to make sure CPU cores/memory on all the nodes in a cluster is efficiently utilized to improve performance.

2.1.10 Why Spark?

Experiments conducted on Spark applications have shown that similar MapReduce programs in Hadoop take longer time to run than that of Spark. One of the experiments was running logistic regression (Supervised Machine Learning Algorithm) on two set of points to find the best line separating them. This algorithm is iteratively run multiple times to get the best solution and it shows that on Hadoop with the increase in number of iterations the time taken also increases.

The data set size is 29 GB; the test was carried out on a 20 nodes cluster with 4 cores on each node. With Hadoop, each iterations takes about 127 seconds, whereas on Spark the first iteration took 174 seconds and the subsequent iterations (the data once cached) took only 6 seconds.
The below diagram shows the result,

![Diagram showing comparison between Hadoop and Spark](https://example.com/diagram.png)

**Figure 6. Spark/Hadoop Machine Learning Experiment**


Based on this comparison, the Spark framework seems to outdo Hadoop by a large extent. Spark can simply be used as a framework to process data/run machine learning algorithms and HDFS can be used to store the data. Combining these two frameworks together provides opportunities to solve lot of big data problems.

Another good example of Spark applications outdoing Hadoop map/reduce programs is video streams optimization and data mining done by a firm named Conviva [16]. The firm had originally written the data mining process in Hadoop and it took about 20 hours to process all data they have, it reduced to 30 minutes when they switched to Spark.

Spark is not only for iterative algorithms and data mining, but can also be used for any general processing of Big Data.

Spark framework also has a Streaming API, this API can be used to connect to major open source message queue infrastructures to subscribe to messages real time and apply Spark transformations/actions.

2.1.11 **Kafka**

Apache Kafka [4] is a “publish-subscribe” high throughput distributed messaging system. It is a messaging system which utilizes the power of cluster computing for producers to distribute, publish the messages and for the subscribers to consume.

Some terminologies used in the Kafka framework are,

- It maintains feeds of messages in categories called “Topic”
• The process that publishes messages to a Kafka topic is called “Producer”
• The process that subscribes the published message is called “Consumer”
• It is run as a cluster which is comprised of one or more servers (nodes) each of which is called a “Broker”.

At a high level, producers send messages over the network to the Kafka cluster which in turn serves them up to consumers like this below,

![Apache Kafka Architecture](http://kafka.apache.org/documentation.html#introduction)

**Figure 7. Apache Kafka Architecture**

Messaging traditionally has two models namely queuing and publish-subscribe [4]. In a queue, a set of consumers may read from a server and each message goes to one of them. In publish-subscribe the message is broadcast to all consumers.

Kafka framework can be used in many distributed and high volume of data systems. As such the messages are persisted on disk and replicated within cluster to prevent data loss. Each broker can handle terabytes of messages without performance impact.

Kafka uses Zookeeper server for configuration management. The Zookeeper server is a configuration management server which can act as a central process for client processes (both producers and consumers) to connect to and read configuration information about the Kafka brokers. Also, the message publish-subscribe offsets can be maintained in the Zookeeper for the producers/consumers to read/write. This is used to keep track of the messages published and consumed.

2.1.12 **Big Data in Financial Markets**

In the financial services sector, increasing strict market regulations require the organizations to capture and store massive amounts of data, process them both in real time and batch mode to determine useful information like customer buy/sell trends, new opportunities, fraud activity etc. This requires the introduction of Big Data in the financial sector.
2.1.13  **Clustering (Unsupervised Machine Learning) in Financial Markets**

As described here 2.1.4, Clustering is a technique of classifying objects that splits a diverse group into smaller groups of similar objects whose characteristics of similarity not known in advance. Using clustering technique, trades that belong to each trader will be analysed to group similar trading activity together and understand every trader's pattern of trading. This pattern can then be used to find better opportunities for the traders or any kind of fraudulent activity by the trader.

2.1.14  **Fraud detection in Financial Markets using Clustering**

The clustering information of traders can be used to find any outliers in new trading activity. This is a data mining/machine learning application. Other techniques like a rule engine/expert system in which quantity threshold, frequency threshold of products traded, total volume traded, price, markets can be specified. Both these systems can be used together to do better fraud detection.

Some of the common fraudulent activity like below can also be configured as rules in the expert system,

- **Microcap Fraud**

  In microcap fraud, stocks of small companies of under £160 million market capitalization are deceptively promoted, then sold to public who are unaware of such companies. This type of fraud has been estimated to cost investors £640 million–£2 billion annually.

- **Short Selling Abuses**

  Abusive short selling are ones that drive down securities prices, the stocks are sold without being borrowed and also without intent to borrow.

- **Penny Stock Fraud**

  Penny stocks are those who per share value is less than or equal to £3, usually buying such products at lower prices and then selling at inflated prices frequently to make profit is considered penny stock fraud.

The main approach to this is to find patterns in the trading activity for each trader; this technique has already been applied to credit card fraud detection system (FALCON [25]). The application uses very similar approach of finding patterns in cardholder behaviour and then detects fraud if any future trading activity for that customer does not match the behaviour. This system currently protects more than 2.1 billion active accounts world-wide [26].
3 Project Approach

3.1 Applying Clustering Technique

Assuming that account is a unique identifier for each trader, the historical trading activity for every account will be used to run clustering machine learning algorithms. The two main algorithms that will be used in this are,

- EM (Expectation Maximization) Algorithm [21]
  
  It is used for finding maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved variables. EM alternates between an expectation steps which computes the expectation of the likelihood assuming the variables as observed. The maximization steps, which computes the maximum likelihood estimates of the data points by maximizing the expected likelihood found on the expectation step. The parameters found on the maximization step are then used to run another expectation step and the process is repeated until the algorithm converges which is when the log likelihood starts decreases.

- K-Means Algorithm [22]
  
  It aims to partition n observations of k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the clusters. K-means has two iterative steps, the assignment step in which the data points are assigned to the clusters (based on distance from sample centroids – mean data point of each cluster selected randomly), then the update step which is used to calculate the new centroids based on the assignment step, these are carried out iteratively until the assignment step has no changes causing the algorithm to converge.

These two algorithms are used together to find the best clusters/pattern in the trading activity of each trader. The K-means algorithm requires the number of clusters in the data points to be specified upfront, this means the data needs to be plotted or pre-processed to determine the number of clusters. This requires additional user intervention, but to automate this process, the EM algorithm can be used in combination to determine the best number of clusters in the data set. With the help of cross validation step the number of clusters can be determined. The data set is split in to training and testing data, the training data can be used to run the K-Means algorithm iteratively with increasing number of clusters. The EM steps are carried out iteratively as well to find the log likelihood for the training and testing data, this determines the best number of clusters. It can then be used again to the run the K-means algorithm to get the final set of clusters and its properties.
The algorithms will determine patterns in the trading activity and group them together based on similarity among the trades for each trader.

3.2 **Issue getting access to Real Data**

The system requires historical trading activity for traders to run the algorithms in order to determine patterns. Getting access to real trader data is not viable; this information is held completely in private by many investment firms and not shared with external entity other than for audit/regulatory purposes.

For the purpose of this system, trading activity needs to be generated to run the machine learning algorithms.

In order to create a system that will prove the clustering works and also the fraud detection works, the data can be generated with some predefined pattern and thus cross verifying that the pattern is determined systematically by the machine learning process.

3.2.1 **Trading Pattern**

Usually traders maintain a portfolio with predefined pattern like trading a certain volume/quantity on certain exchanges and on a set of products belonging to particular sectors. For E.g., a trader could configure his/her portfolio to trade on London Stock Exchange and Technology sector products (like IBM) 50% of the times and then configure the other 50% to trade on a different exchange/sector. The trader could also specify other parameters like the order quantity, price ranges, and trade value for various products as part of the portfolio settings.

3.3 **Assumptions in type of trading order and variables for Machine Learning**

In a real trading environment, there are many types of trading activity like Algorithmic Orders, Smart Orders, High Frequency Orders, Direct Market Access Orders and Sales Orders. Some of these types generate a large number of smaller orders for one big order and so there are multiple trades for a single order. This is also generated at a very high frequency. As one of the use cases for this system is to build a framework that will handle historical trading orders to run machine learning algorithm and then detect uncharacteristic behaviour in the trading activity, the system will be simulated to run machine learning from executed orders on the final state of the executed order (instead of individual trades for each order). Dealing with individual trades would mean grouping them all together based on order id to do the learning process. In order to get directly to the machine learning process, the system will assume trade quantity for the orders and not for individual trades. For the purpose of this system, the account will be assumed to be unique. The profiling information for each trader will be based
on the account number, in theory the trading behaviour is tied to the account rather than the
trader itself. Even if the trader uses multiple accounts to trade on different exchanges/products,
the machine learning should work as expected.
This approach will be taken using a data generator (trade generator) to generate historical
trading activity for various trading accounts. Initially there will be three variables taken to run
the machine learning, namely the “Trade Quantity”, “Exchange” and “Sector”. This is to prove
the system works as expected with the pattern chosen for a sensible set of variables, ref 3.2.1.
The system will be configurable to add more variables as required for advanced machine
learning purposes and also to detect fraud.
Once this data is generated, the pattern/portfolio settings specified by the trader will be
available in the form of raw data; this data will then be used in the machine learning process to
create the clustering information. The clustering information can be used to cross check if the
machine learning process worked as expected by verifying the pattern.

3.4 Why Clustering?

The credit card fraud detection has shown that the unsupervised learning technique works well
to determine patterns in cardholder activity. The FALCON Fraud Manager [25] is a very big
credit card fraud detection system, it currently manages 2.1 billion accounts, it protects about
85% credit cards used in US and 65% credit cards used worldwide. It uses profiling
technology to identify transaction behaviours and spending patterns for each service, account
and customer to ensure rapid recognition of uncharacteristic expenditures. This proves that this
approach has worked successfully in the industry. Although the credit card system and the
trading system are two separate entities, the same approach can be applied to determine
patterns in trading activity and any uncharacteristic trading behaviour.

3.5 Real Time Fraud Detection

Once the clustering information for each trader is available, this can then be loaded on to a real
time streaming application to check new trading activity against this model, any differences in
the usual behaviour in the trader activity can be raised as an alert for further investigation by a
business analyst. The same approach that is used in clustering can also be used to find the
distance between the new data points (trading activity) to that of the clusters centroids (mean
of each cluster) to determine the differences and raise an alert.

3.6 Why Real Time?

There are some systems which are built to run such analytics end of day to determine unusual
patterns in the trading activity. The real time approach helps to determine such behaviours
sooner as they happen during the trading day and thus prevent any additional financial damage that the fraud may cause until the end of the trading day.

3.7 Using Hadoop, Spark and Kafka to implement the solution

Hadoop will be used to store the trading activity for each trader (account); the trading activity can be stored in semi structured format (CSV) on HDFS.
HDFS will also be used to store the clustering information for each trader.
The Spark application will run the EM and K-means algorithm on the historical trading activity of each trader to determine the clusters. The obtained clustering information for each trader is then persisted in HDFS.
Apache Kafka will be the messaging infrastructure for creating a publish-subscribe model of storing the real time trades in memory, this will then be consumed by a Streaming Spark application to compare the real time trades for each trader against his/her corresponding clustering information to determine any uncharacteristic behaviour.

3.8 Why Hadoop, Spark and Kafka?

The amount of trading that happens around the world and the data generated are massive.
Some of the big investment banks and brokerage firms in the market trade huge volumes and have close to billion transactions a day and many thousand transactions per second. In order to process such high volume of data and also to store such data at a central location to run various batch analytics or real time analytics these need a highly robust distributed computing infrastructure.
Hadoop solves the purpose of storing the trading activity of each trader in the HDFS and can be accessed from a central location making it easier to run many analytics. Also, due to the storage being cheaper and all the nodes in the cluster use commodity hardware; historical data may never need to be tape archived reducing some infrastructure costs.
Spark on the other hand can be used to run faster analytics combined with Hadoop, the historical trading activity can be accessed from HDFS and the machine learning algorithms can be run against this data in Spark, the clustering output can then be stored back in HDFS.
The clustering information can be then be retrieved by a Spark streaming application to run real time check on new trading activity.
Apache Kafka is a good match for such high volume real time data handling, the huge transactions per second nature of the securities trading require Kafka like infrastructure to store messages in memory/replication across clusters to make sure no messages are lost for processing (very important from both regulatory/audit perspective and daily transaction processing).
The frameworks provide the right mix of Big Data storage and Cluster Processing techniques to work under a high volume trading environment. Above all, these frameworks are open source, so no licensing fees is required to use any of these frameworks. There is a larger community of users and developers to get help from as well.

3.9 **Running the system on a real cluster of nodes**

As all the frameworks used in the system like Hadoop, Spark and Kafka run on a cluster of nodes. There is a requirement to setup a cluster with some nodes to run all these frameworks and the applications to see a reasonable performance in the execution of the entire system.

Setting up a cluster is not financially viable personally for the purpose of this project, in a real world scenario; any investment bank which wishes to use this can setup a cluster and run this system.

Having said that, during the initial learning stages of the cluster computing, running a Spark application on a 6 large Amazon Elastic Computing Nodes Cluster cost about $11/hour!

07/05/2013 07/07/2013 Sale Amazon Web Services $11.52

3.10 **Issue running the system in standalone mode**

Spark applications consume a lot system memory due to RDDs caching and other memory intense operations of splitting the raw data across a Master/Worker, accumulating operations of various results on the Master process. This framework is meant to run on a cluster of nodes and the core is programmed to utilize the resources of the cluster in an optimal way and does not fare well when run in a standalone mode. Also, with the Java processes, even if care is taken to take back any unused memory by assigning “null” to the Java objects so the JVM Garbage collector cleans up some unused heap memory, the system memory requested by the Spark application from the Operating System is not freed up. This means the memory is accumulated as the RDDs become complex when more clusters are created during the iterations running EM and K-means algorithms.

The result is that on a standalone PC of 3.7 GB RAM and 4 Cores, a Spark application running EM and K-means algorithms on a data set of 1000 orders for one account with 4 clusters (machine learning)/different patterns in the set takes about 1.5 GB of RAM. Apart from this the PC is also configured to run the frameworks Hadoop, Spark and Kafka. It also requires the trade generator to run alongside. The Spark clusters by default takes about 1 GB less than the actual memory on the PC to run, this is about 2.7 GB. It is also configured to take up 1.5 GB for the Worker process. So once the 1.5 GB limit is reached by the Spark application, the process starts swapping memory to disk, this means the process takes up a lot of CPU causing
the PC to not respond even with some memory left on the PC. Increasing the memory limit for
the Worker also results in the Spark framework to not start up the application because it is
configured internally to have a threshold on memory left on the PC/node to start any
application. If it cannot assign a particular amount of virtual memory to the application, the
application will not start up.
4 Design

4.1 Use Cases

The system has two main features,

- Running Machine Learning Algorithms on Historical Trades:
  The machine learning on the historical trades is done by entering trades using a trading user interface; this will also involve storing these trades in Hadoop file system (HDFS).

- Fraud Detection on Real Time Trades:
  The other main objective of the system is to detect fraudulent activity real time in trades entered by the traders. This involves publishing trades on a message queue to manage real time streaming of trades in memory and also for clustering information to be available in HDFS.

The use case will show both the user and system actions together to show the relationship between the system actions and its dependency on the user actions. Extensions to those actions are shown as “includes” in the diagram. A combination of business and system use case.

Figure 8. Use Case Diagram
### 4.1.1 Trade Entry with Pattern

**Description:** Entering trades through a trade entry interface with a pre-defined pattern to create historical trades in the system.

**Includes:** Storing Trades in Hadoop File System. All entered trades for various accounts are stored in Hadoop.

**Actor:** Trader

### 4.1.1.1 Sub Use Case: Machine Learning

**Description:** System process will start machine learning clustering algorithms on the trades stored in HDFS to form clusters (find patterns).

**Includes:** Finding clusters in the trades from HDFS for each account.

**Depends:** Trades should be available in the HDFS for each account. Clusters will be written back to HDFS once determined.

**Actor:** System

### 4.1.2 Real Time Trade Entry

**Description:** Entering trades through a trade entry interface real time.

**Includes:** All entered trades by the trader will be published on to a message queue to manage streaming trades’ real time in-memory.

**Actor:** Trader

### 4.1.2.1 Sub Use Case: Fraud Detection

**Description:** System process will start fraud detection on the real time streaming trades.

**Depends:** Trades published on a Message Queue and available for subscription. Also, the clusters for each account should be available in HDFS.

**Actor:** System
4.2 **Class Diagrams**

The project has been split into five different applications. This is to better manage the different aspects of the project. Also based on good software engineering and object oriented principles, keeping the classes in the system decoupled allows for reuse and lower impact on the entire system during changes. This also helps during testing various units in the system separately.

4.2.1 **Applications in the System**

The following gives an overview of each application,

- **Trade Generator**

  The Trade Generator application of the project is used to generate historical trades and also to generate real-time trades. This application is called **FMFixSimulator** (Financial Market FIX Simulator). It consists of the modified FIXimulator and Banzai GUIs. Detailed class diagram here 4.2.2.

- **Trade Publisher**

  The Trade Publisher application will connect to the FIXimulator, consume FIX messages and publish trade objects on a message queue. The application is called **FMTradePublisher** (Financial Market Trade Publisher). Class diagram here 4.2.2.

- **Trade Subscriber**

  The Trade Subscriber application will connect to the message queue topic that the Trade Publisher application published messages to and consume these messages. It will then store these messages on to a distributed file system. The distributed file system will also require trade aggregation using a Trade Aggregator. This application is called **FMTradeSubscriber** (Financial Market Trade Subscriber). Class diagram here 4.2.2.

- **Machine Learning**

  The Machine Learning application will read trades list of every trader account from a distributed file system (DFS) and create/find clustering/patterns information. This information will be stored back into the DFS. It runs both the EM and K-Means machine learning algorithms to find patterns. This application is called **FMMachineLearning** (Financial Market Machine Learning). Class diagram here 4.2.4.

- **Fraud Detection**

  The Fraud Detection application reads the clustering information from the DFS. It also consumes real-time trades from the Trade Publisher application. It then
compares the new trades with the clustering information for the corresponding trader account and gives a verbose output on the differences. This application is called FMFraudDetection (Financial Market Fraud Detection). Class diagram here 4.2.5.

The following diagram shows all the applications described above working together,

![High Level System Design Diagram](image)

**Figure 9. High Level System Design**

4.2.2 **FMFixSimulator and FMTradePublisher**

The Use Case 4.1.1 corresponds to the historical trading data generation for machine learning purposes.

The FIXimulator and Banzai GUIs are from [1]; this tool is used as the trade entry and stock exchange simulator.

The below class diagram shows the FMTradePublisher application,
The FixConnector connects to the FIXimulator core and the FixApplication handles all FIX messages. The TradePublisher will take the FIX messages convert them to Trade objects and publish to a message queue topic. The FIXimulator and Banzai tools will be modified to handle randomization of order generation using some predefined pattern. There will also be provision in the FIXimulator to accept additional FIX connections from a message queue publisher application.

4.2.3 FMTradeSubscriber

As part of the Use Case 4.1.1, the trade objects that are published by the FMTradePublisher will be consumed from the FMTradeSubscriber application and persisted on to a distributed file system.

The below class diagram shows the FMTradeSubscriber application,
The application will also be configurable to take any set of variables from the trade object to store in the distributed file system. This is required to support additional attributes during the machine learning process.

4.2.4 FMMachineLearning

The Sub Use Case 4.1.1.1 is about the machine learning from the historical trading activity from the distributed file system. It also deals with storing clustering information from the machine learning process back into the distributed file system.

The below class diagram shows the FMMachineLearning application,

![FMMachineLearning Class Diagram](image)

**Figure 12. FMMachineLearning Class Diagram**

The FMMachineLearning is used to start the EM machine learning algorithm. The EM algorithm runs on the trades list from distributed file system for each trader account and creates clustering information which is then stored back in to the distributed file system. The EM algorithm is mainly used for determining the number of clusters in the trade list; this uses the K-means algorithm to find the exact clustering information. The application is capable of passing the trade files to any algorithm. In future, any additional algorithms can be implemented and passed these trade files for machine learning.
4.2.5  FMFraudDetection

The Use Case 4.1.2 and Sub Use Case 4.1.2.1 deal with the fraud detection process of the project. These require a real time stream of trades and clustering information of individual accounts from the distributed file system.

The below class diagram shows the FMFraudDetection application,

![FMFraudDetection Class Diagram](image)

The clustering information from the distributed file system are read by the FMFDMain class and passed on to the FMFraudDetection class. It maintains a list of clusters for each trader account. New trade objects are consumed real time from a message queue by the FMFraudDetection class and compared against the clusters to give a verbose output.
5 Implementation

The approach taken to implement this project is shown in chapter 3. That chapter describes some of the challenges and decisions made to carry out the project work.

Implementation details of each of the applications described in the section 4.2.1 will be discussed below.

5.1 FMFixSimulator

The data generation was the first major challenge of the project. Due to the unavailability of real data the first step in the project was to generate simulated data.

First option to generate data was manual entry of some orders and start with the machine learning process. This approach was not taken as the data required some pattern in it so the same pattern can be determined in the machine learning process to prove the system works as expected. To create pattern in the trading activity, there was a requirement of an automated order entry mechanism. FIXimulator/Banzai [1] GUIs was chosen to be best for trade generation. The same tool is used for both historical trade activity generation and real time trade entry. This also gives a simulated setup of a real world trade entry/stock exchange.

The basic FIXimulator and Banzai GUIs are Java Swing/AWT based and have functionality to send single orders and automated executor functionality which behaves like a simulated exchange. The implementation uses QuickFIX/J [2], which is an open source version of the official FIX protocol [3]. FIX (Financial Information Exchange) is a high level messaging protocol used to exchange financial information between financial sector applications both within an organization and between organizations.

In order to create pattern in the data, the Banzai GUI was changed to add randomization variables and generate orders with the specified pattern. There is also an option added to save these randomization settings to generate more orders again in future.

The Banzai and FIXimulator were changed to support “Account, Exchange and Sector” FIX fields. This is required as Account is chosen as the unique identifier for each trader. Exchange and Sector variables along with Trade Quantity are the three variables that were chosen to do the machine learning process initially.

Apart from this, there were changes to add support to load stocks from a XML file which contains a list of instruments and some instrument properties like its symbol, exchange, sector etc. These will be used to get appropriate stocks to send orders based on the randomization criteria specified in the Banzai GUI.
Sample instruments XML is shown in Appendix 3 Code Snippet [1].

The below diagram shows an overview of the modified trade entry GUI Banzai,

![Banzai GUI Diagram]

**Figure 14. Banzai GUI**

The Fiximulator GUI was modified to support drop copy sessions. The drop copy session is a concept in FIX where the process that is handling the FIX messages can be configured to drop copies of messages to a set of downstream FIX connections. This is required for streaming the FIX messages on to a message queue. The message queue framework chosen was Apache Kafka, overview here 2.1.11.
The FIXimulator GUI is shown below,

[Image: FIXimulator GUI

The high level flow chart shows the trade generation operation,

Figure 15. FIXimulator GUI

The high level flow chart shows the trade generation operation,

Figure 16. FMTradeSimulator Flow-Chart

- 33 -
5.2 **FMTradePublisher**

One of the requirements of the project is to process messages real time. This means the trade messages that are generated need to be stored in a message queue in-memory for the subscribers to not lose any messages. Since the implementation of the project uses big data and cluster computing technologies, Apache Kafka 2.1.11, which is a distributed messaging system is chosen as the message queue framework.

The FIXimulator drop copies execution messages to another FIX session. This is the Kafka drop copy session which takes FIX execution messages.

Some sample FIX Messages are shown in Appendix 3 Code Snippets [2].

The FMTradePublisher is divided into a FixConnector to receive drop copies from the FIXimulator and TradePublisher to convert FIX messages into Java “Trade” Objects and publish to a Kafka message topic “trades”.

The TradePublisher is also capable of reading FIX Messages from a flat file, parse and publish to the same Kafka message topic. This is just an added feature; any raw FIX Message files can be used for testing purposes as well. The high level flow chart shows the fix connector and trade publisher operations,

![Flow Chart](image)

*Figure 17. FMTradePublisher Flow-Chart*

5.3 **FMTradeSubscriber**

The Trade objects which are published on the Kafka message topic “trades” will be consumed by the FMTradeSubscriber application for storage in Hadoop Distributed File System.

To understand the implementation of the FMTradeSubscriber, the machine learning process needs to be understood. In order run the machine learning process, various trade attributes are required to run the EM and K-means algorithms. The attributes that were chosen to run the
machine learning process were “Trade Quantity”, “Exchange” and “Sector”. The aim of the system was to make it extendable. The “Trade” object contains various fields as shown here Appendix 3 Code Snippets [5]. The FMTradeSubscriber is configurable to add more attributes which are available in the “Trade” object. There are two types of fields that are supported, numerical and categorical. The category field also takes the set of valid values in that category. This is helpful to convert “String” category values in the data to “Numerical” values to do machine learning.

*It is important to understand this feature as it makes the system more extendable for further improvements. Additional fields will be automatically picked up by the FMTradeSubscriber and the values for these fields will be obtained from the Java “Trade” Object using the Java Reflections API.*

The sample trade Java properties file looks like the one shown here in Appendix 3 Code Snippets [3].

There are name and values in this file, the “fields” contains the attribute names. This is a set of column names; each category will be either set to 0 or 1 to indicate the existence of that category in any give trade. E.g. a trade on NYSE would mean that column gets a value of “1” and all other exchanges get a “0”

Apart from this trade property configuration, the FMTradeSubscriber also contains two other properties. The machine learning process is run on a per account basis, so the trades will be stored on Hadoop Distributed File System under a folder structure that corresponds to each individual account. The program is configurable to create new “Account” folders for any new account that starts trading and write to the existing account folder that already has traded previously. This is also part of the requirement to make the system extendable. The information about the accounts is held in a properties file.

One of the major advantages of using the Hadoop Distributed File System is it can handle large files efficiently. HDFS is meant for writing and reading large files, it does not support file appends or inserts. This means that whenever new trading activity starts for a given account, it will be written to individual files on HDFS. Storing many small files on HDFS is very inefficient; the memory overhead to access these files is high. In order to overcome this inefficiency, there is a TradeAggregator process which will run to aggregate all the trade files for a given account into one large file. This not only helps make file access efficient on HDFS but also helps the Spark machine learning process to read from one large file.
The high level flow chart below shows the FMTradeSubscriber application,

![FMTradeSubscriber Flow-Chart](image)

**Figure 18. FMTradeSubscriber Flow-Chart**

5.4 **FMMachineLearning**

The FMMachineLearning is one of the major aspects of the project. The idea behind finding patterns in the trading activity required use of machine learning. The methodology of finding patterns and then detection uncharacteristic behaviour is already proven in other systems. The same approach is taken for the fraud detection in financial markets as well.

For machine learning, there is a requirement to identify appropriate set of attributes in the data. The “Trade” object consists of many attributes. For the purpose of testing the system works as expected, three attributes were chosen, namely, “Trade Quantity”, “Exchange” and “Sector”. The idea behind this decision is shown in this section 3.3.

The machine learning algorithms chosen were EM and K-means. These algorithms are clustering algorithms, short definition here 2.1.4. The algorithms are mainly used to find patterns in the data and cluster similar data based on distance from a given point called “Centroid” which is the center of cluster of data or the mean of given set of points in a cluster. Running these algorithms on a given data set can be done using many existing tools; “Weka” [19] is one among them, it is a Java program. The “Weka” tool has support for both these algorithms. The sample data generated for a trader account was used in the Weka tool to get the clustering information and the output showed the right set of clusters that were expected.

At this point, the Weka tool is useful to run the algorithm only on a single JVM. With Big Data and distributed file systems, the processing is also required to be run on distributed systems.

For running the machine learning algorithms on a cluster of nodes, the Spark framework was chosen, some details on the framework here 2.1.9.
With Spark, raw data is loaded onto RDDs (Resilient Distributed Datasets); this is a mechanism to split the data across multiple nodes in a cluster. The Spark application runs on a Spark cluster and it has a Master (Driver) program and a set of Worker programs. The Master is responsible for splitting the task and sending/receiving intermediate results from all Workers in the cluster.

Spark is built in Scala programming language which is a JVM based language. The Scala code is compiled into Java Byte Code and run on a JVM. Spark framework also provides a Java API.

A Sample Java RDD is shown in Appendix 3 Code Snippet [5] it is a Map/Cache transformation in Spark.

This operation will pass a “Closure” to the map function on the HDFS file and assign the same to JavaRDD “instancesRDD” above; it is evaluated on the Spark cluster only when any actions or data collection on the RDD is done.

Similarly there are multiple Map/Reduce or Filter/GroupBy actions that can be done on a JavaRDD object to carry out various operations and get appropriate results in the EM/K-means algorithm implementation.

Some more Spark JavaRDD functions here Appendix 3 Code Snippet [5] is a Map/Filter transformation for creating “Training RDD” instances in the EM Algorithm,

Similarly a Map/Reduce transformation is shown here Appendix 3 Code Snippet [7] is used for calculating the log likelihood in the Expectation step of the EM Algorithm,

There are other transformations that are required to run the EM and K-means algorithms using the Spark framework. These transformations are done only for steps in these algorithms that work on a list of data. The list of data is assumed to be larger in size (Big Data) and require a cluster of nodes to handle them and process them.

There are some steps in the algorithms which require final results from aggregation of all chunks of data from all nodes. In order to support this, Spark provides another type of objects called “Accumulator”; these are used as a way to accumulate values from various Worker nodes and get the final value from the Master node. For EM algorithm, this is required for getting the normal estimate values for clusters in the Maximization step. In the K-means algorithm, this is required for checking for algorithm convergence, accumulating squared errors and also for cluster centroids.

The FMMachineLearning application initially connects to HDFS (Hadoop Distributed File System) to get a list of accounts (account folder in HDFS), and from each account folder the raw trades file will be passed to the EM algorithm. The Spark loads the application as soon as
the Spark Context is started; this is a mechanism to load the application into the cluster. The JAR files of the application are copied over to Worker nodes in this operation. The EM algorithm is used only for cross validation in this implementation. The cross validation step is useful to determine the appropriate number of clusters in the data set. The cross validation step in EM algorithm runs the K-means algorithm iteratively with varying number of clusters and verifies the log likelihood to determine the best set of clusters. The K-means algorithm provides the final set of clusters and its properties. The clustering information of each individual trader account is converted into a “Cluster” object. This is serialized and stored in HDFS.

The high level flow chart of the FMMachineLearning application is shown below,

![Figure 19. FMMachineLearning Flow-Chart](image)

5.5 FMMFraudDetection

The second major aspect of the project is Fraud Detection. Fraud is a very broader term and it means different kind of activities. Some common ones listed here 2.1.14. The approach taken for this project is finding patterns in the historical trading activity; create machine learning clusters and then verifying the difference between new trading activity and clusters. The difference check will determine if the new trading activity fits into the previously known pattern. If the new trading activity does not fit into the previously known pattern, the differences in the data will be output.

The initial idea of the fraud detection was to find differences and send them as alerts to a front end which will be monitored by a business analyst. The business analyst can then manually verify the validity of the differences/alerts. This information will feed back into the machine
learning process. This will make the fraud detection process even more accurate and avoid any false positives in future.

In the interest of time, the differences between the clusters and new trading activity will be shown on a terminal window.

The FMFraudDetection process initially starts up and reads all the cluster objects for various trader accounts from HDFS loads them into memory. It keeps a map of “Account” and a list of cluster objects.

In order to process the trades’ real time, the Spark framework also has a Spark Streaming API. Using the Spark Streaming API, messages from an Apache Kafka message topic can be directly consumed and processed in-memory. The Spark Streaming API has a different type of object called DStream and the Java version is called JavaDStream. It holds the list of individual messages from a stream.

The Sample JavaDStream looks like shown here in Appendix 3 Code Snippets [8].

The Kafka stream contains serialized “Trade” objects, this need to be de-serialized and converted into vectors for running difference checks with the cluster objects. The method to process Kafka messages and apply a decoder for individual messages in the Spark Streaming API had a bug as it is still in Alpha version; it assumed each message of type “Object” even after using Java Generics to specify them as “Trade” objects. The short term solution/hack was to hard code the “Trade” Encoder/Decoder class (TradeSerializer) into the Spark Streaming Kafka processing method. The API was directly changed, recompiled and used in the FMFraudDetection application.

This change also means that the “Trade” and the “TradeSerializer” class are now part of the Spark Streaming API and need to be referenced by other applications like the FMTradePublisher and FMTradeSubscriber to process the “Trade” objects appropriately.

The high level flow chart of the FMFraudDetection application is shown below,
5.5.1 Distance Calculation

Euclidean distance is used in the K-means algorithm to find the distance between a given instance vector and a centroid vector. The same approach is used in the distance calculation to find differences in the new trading activity against the corresponding centroids. The best centroid is chosen based on the shortest distance. The issue with this approach is that the categorical values in the data are converted to numerical values by giving a “0” or “1” value based on the existence of the appropriate category value in the data point. This will give a wrong distance for data points which have higher trade quantity and thus the actual weight of exchange and sector categories in this case will be ignored or very minimal. This will result in inappropriate cluster fit for the new trading activity. In order to overcome this issue, the distance calculation is extended to do equality checks for categorical value. It will verify the Euclidean distance for the numeric fields separately and then perform category equality checks to find the best cluster for the new trading activity. If there is no best cluster for the new trades, then the most probabilistic one is chosen and the differences between the new trade and the cluster chosen are shown. Also, for testing purposes, the output will also show differences between the new trade and all other clusters that belong to the trader account. The distance function is shown here in Appendix 3 Code Snippets [9].
6 Testing

6.1 Functionality Testing

6.1.1 FMFixSimulator

The FIXimulator and Banzai GUIs were modified to add randomization variables and automatic generation of orders. The logic to send orders using normal distribution on order quantity was tested by plotting the trade quantity and it showed that the normal distribution worked as expected. This is a normal distribution with mean quantity of 100 and a standard deviation of 25.

![Figure 21. Normal Distribution of Order Quantity](image)

In order to verify probability distribution on exchange and sector, the trades were stored in a raw text file and manual Linux “grep” functions were run to find the actual count. The code to generate random orders has iterations to check for probability of the occurrence of the Exchange and within that iteration to check for probability of a particular Sector. This means that even if the number of orders chosen to be sent is fixed, the randomization and this probability check results in either lesser or higher number of orders than chosen, which is expected.

There were 986 trades generated for account ABC123 with 50% of trades on NYSE exchange and 50% on LSE exchange. The same 50/50 split is done for BANKS and TECHNOLOGY sector. The “grep” results show the probability distribution worked as expected,
The columns are,

Account, Trade Quantity, Exchange, Sector

// Line count of trades in the temporary trades.csv file
hduser@localhost ~/MSCProject $ wc -l /tmp/trades.csv
986 /tmp/trades.csv

// Checking the percent split of NYSE exchange in the trades file
hduser@localhost ~/MSCProject $ grep NYSE /tmp/trades.csv | wc -l | awk '{ tmp = ($1/986) * 100; printf"%0.2f\n", tmp }'
50.10%

// Checking the percent split of LSE exchange in the trades file
hduser@localhost ~/MSCProject $ grep LSE /tmp/trades.csv | wc -l | awk '{ tmp = ($1/986) * 100; printf"%0.2f\n", tmp }'
49.90%

// Checking the percent split of TECHNOLOGY sector in the trades file
hduser@localhost ~/MSCProject $ grep TECHNOLOGY /tmp/trades.csv | wc -l | awk '{ tmp = ($1/986) * 100; printf"%0.2f\n", tmp }'
47.97%

// Checking the percent split of BANKS sector in the trades file
hduser@localhost ~/MSCProject $ grep BANKS /tmp/trades.csv | wc -l | awk '{ tmp = ($1/986) * 100; printf"%0.2f\n", tmp }'
52.03%

The above output shows that the randomization and probability distribution worked as expected with minor negligible errors.

6.1.2 FMTradePublisher

In order to test the FMTradePublisher, that consumes FIX messages from the FIXimulator as drop copies. The FIX Logs were verified to check for “outgoing” messages from the FIXimulator to both the Banzai GUI and the FMTradePublisher FixConnector process which is configured to be called as the KAFKA FIX session.

Message sent back to Banzai
3943 outgoing 20130829-09:47:25.888 ExecutionReport
  8=FIX.4.2|9=230|35=8|34=1975|49=FIXIMULATOR|52=20130829-09:47:25.888|56=BANZAI
  1=ABC123|6=33.79|11=1377769646870|14=99|17=E1377769647860|20=0|31=33.79|32=99|37=O1377769646873|38=99|39=2|54=1|55=ACN|75=20130829|107=TECHNOLOGY|150=2|151=0|207=NYSE|10=221

Same Message drop copied to Kafka (FixConnector)
3944 outgoing 20130829-09:47:25.889 ExecutionReport
  8=FIX.4.2|9=228|35=8|34=988|49=FIXIMULATOR|52=20130829-09:47:25.889|56=KAFKA
  1=ABC123|6=33.79|11=1377769646870|14=99|17=E1377769647860|20=0|31=33.79|32=99|37=O1377769646873|38=99|39=2|54=1|55=ACN|75=20130829|107=TECHNOLOGY|150=2|151=0|207=NYSE|10=097

The FMTradePublisher also converts the FIX messages into “Trade” objects, serialize the “Trade” objects and publish them to a Kafka message topic by the name “trades”.

The messages from the Kafka message topic needs to be consumed to verify that the serialization and publish operations worked as expected. This is done in the FMTradeSubscriber application.
6.1.3 FMTradeSubscriber

The FMTradeSubscriber consumes “Trade” objects from the Kafka message topic “trades” and de-serializes the objects. Based on the trade configuration in the “trade.properties” file, the fields to be retrieved from the “Trade” object is chosen, the values of these fields are laid out in comma separated format and stored in HDFS.

The files get flushed to the HDFS only when a certain memory threshold is reached. In this project for 986 trades generated, the data size is too small to get flushed to the HDFS file system immediately, the immediate flushing to disk can be achieved by keeping smaller block sizes for the file chunks, but it is very inefficient. Keeping the 64MB block size unchanged and testing the application shows that the FMTradeSubscriber application needs to be stopped for the Hadoop framework to write the data to disk.

Contents of directory /user/lhuser/MSCProject

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Replication</th>
<th>Block Size</th>
<th>Modification Time</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>dir</td>
<td>1.92</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>clusters</td>
<td>dir</td>
<td>2.100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trades</td>
<td>dir</td>
<td>2.193</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Go back to DFS home

Figure 22. HDFS Project Directory Structure

File: /user/lhuser/MSCProject/trades/ABC123/1.data

Go back to dir listing
Advanced view/download options

0,197,0,0,1,0,0,1,0,0,0,0
1,92,1,0,0,0,0,0,0,1
2,100,0,0,1,0,0,0,0,0,1
3,193,1,0,0,0,0,1,0,0,0,0
4,96,0,0,1,0,0,0,0,0,1
5,210,0,0,1,0,0,1,0,0,0,0

Figure 23. HDFS Trade File

Also, the TradeSubscriber writes to HDFS and creates smaller trade files for iterations of trade generation. This means there will be many small files in HDFS. In order to avoid this issue, after the TradeSubscriber creates these files, the TradeAggregator will aggregate all the small files into a large file; this was verified to be working fine as well.
6.1.4 FMMachineLearning

FMMachineLearning involves reading the trade files from HDFS and running the EM and K-means algorithm. The process has a logging framework which records each step that the algorithms perform; this verifies the file load from HDFS worked as expected.

The EM and K-means algorithm once complete running, they generate clusters for each account and these clusters are stored as serialized object back in HDFS.

The output of the algorithms was verified with the output from Weka to prove the algorithms ran successfully in Spark. The cluster information from Weka and that of Spark application matched exactly proving the Spark implementation was successful.

The following shows the output from Weka and Spark,

<table>
<thead>
<tr>
<th></th>
<th>Weka</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clusters selected by cross validation:</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cluster 0 - Size</td>
<td>234</td>
<td>234</td>
</tr>
<tr>
<td>QUANTITY - Mean</td>
<td>99.6239</td>
<td>99.62393</td>
</tr>
<tr>
<td>QUANTITY - Std Dev</td>
<td>4.9402</td>
<td>4.950790453</td>
</tr>
<tr>
<td>EXCHANGE_LSE - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXCHANGE_LSE - StdDev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>SECTOR_TECHNOLOGY - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SECTOR_TECHNOLOGY - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>Cluster 1 - Size</td>
<td>236</td>
<td>236</td>
</tr>
<tr>
<td>QUANTITY - Mean</td>
<td>199.6992</td>
<td>199.69915</td>
</tr>
<tr>
<td>QUANTITY - Std Dev</td>
<td>9.4011</td>
<td>9.421035727</td>
</tr>
<tr>
<td>EXCHANGE_NYSE - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXCHANGE_NYSE - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>SECTOR_BANKS - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SECTOR_BANKS - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>Cluster 2 - Size</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>QUANTITY - Mean</td>
<td>200.38</td>
<td>200.38</td>
</tr>
<tr>
<td>QUANTITY - Std Dev</td>
<td>10.6104</td>
<td>10.63163802</td>
</tr>
<tr>
<td>EXCHANGE_LSE - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXCHANGE_LSE - StdDev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>SECTOR_BANKS - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SECTOR_BANKS - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>Cluster 3 - Size</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>QUANTITY - Mean</td>
<td>100.3135</td>
<td>100.3134921</td>
</tr>
<tr>
<td>QUANTITY - Std Dev</td>
<td>4.9714</td>
<td>4.981333334</td>
</tr>
<tr>
<td>EXCHANGE_NYSE - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXCHANGE_NYSE - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
<tr>
<td>SECTOR_TECHNOLOGY - Mean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SECTOR_TECHNOLOGY - Std Dev</td>
<td>0.5003</td>
<td>0</td>
</tr>
</tbody>
</table>
The output shows that all the parameter values match, except for the standard deviations for the categorical fields, both Exchange and Sector. This field can only take a 0 or 1, so the standard deviation will not affect this field. Also, the clusters were not in the same order between Weka and Spark, this is not an issue as long as the properties of the clusters matched. This was verified to be good.

The testing was done on a standalone PC and not on a real cluster, the difficulties with that is explained here 3.9 and 3.10.

The output proved that the Spark application worked as expected and the right set of clusters from the data was found. This information will be used by the FMFraudDetection application to do the difference check with the new trades.

### 6.1.5 FMFraudDetection

For the FMFraudDetection algorithm, the original approach was to create a HTML5 front end and publish the differences between new trades and clusters. From the front end, the business analyst will be able to verify the validity of the differences to be regarded as suspicious activity or just new activity from a trader that is acceptable. In the interest of time, this was shortened to just give a verbose output on the differences between the new trade and the clusters.

The issue with de-serialization of “Trade” during the Spark Streaming process is explained here 5.5. This required recompiling of the Spark Streaming API. Once it was recompiled, the library was referenced back in the FMFraudDetection to de-serialize the “Trade” objects from the Kafka message topic. The de-serialization worked as expected once the “Trade” objects were converted into vectors with fields of interest. The distance calculation explained here 5.5.1 found the right differences between the new trades and the cluster centroids of the account.

The verbose output of the differences is shown below,

This is the output for a trade that fits rightly into the one of the clusters of that trading account,
All Clusters Read From HDFS File
===================================================================
[NEW TRADE]
Account : ABC123
Trade Attributes Processed :[QUANTITY, EXCHANGE_NYSE, EXCHANGE_NASD, EXCHANGE_LSE, EXCHANGE_HKFE, EXCHANGE_BVSP, SECTOR_BANKS, SECTOR_ENERGY, SECTOR_HEALTHCARE, SECTOR_MEDIA, SECTOR_TECHNOLOGY]
Trade Attributes Values :(100.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0)
Best Cluster Found :[99.62393162393163, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
Difference Between Trade And Cluster :2 =>
Cluster Info For : 2
QUANTITY : 99.62393162393163 Min : 85.0 Max : 114.0 Range : 29.0
EXCHANGE : LSE
SECTOR : TECHNOLOGY
QUANTITY : 10.0 Within Range 85.0 - 114.0
Traded On : EXCHANGE_LSE
Traded On : SECTOR_TECHNOLOGY
No Differences Found
===================================================================
It is a trade to Buy 100 IBM @ Market in account ABC123. The output shows the “Best Cluster Found” and then compares any differences between the cluster centroid and the trade point, in this case, there were no differences found.

Similarly, for a trade to Buy 1000 IBM @ Market in account ABC12, the output is as follows,

===================================================================
[NEW TRADE]
Account : ABC123
Trade Attributes Processed :[QUANTITY, EXCHANGE_NYSE, EXCHANGE_NASD, EXCHANGE_LSE, EXCHANGE_HKFE, EXCHANGE_BVSP, SECTOR_BANKS, SECTOR_ENERGY, SECTOR_HEALTHCARE, SECTOR_MEDIA, SECTOR_TECHNOLOGY]
Trade Attributes Values :(1000.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0)
Best Cluster Found :[99.62393162393163, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
Difference Between Trade And Cluster :2 =>
Cluster Info For : 2
QUANTITY : 99.62393162393163 Min : 85.0 Max : 114.0 Range : 29.0
EXCHANGE : LSE
SECTOR : TECHNOLOGY
QUANTITY : 1000.0 Greater Than Mean By 900.3760683760684
Traded On : EXCHANGE_LSE
Traded On : SECTOR_TECHNOLOGY
No Differences Found
===================================================================
The output shows that the closest cluster centroid that was found was cluster “2” for the account ABC123. This cluster matches with the trade because it could verify the “Exchange” as LSE and “Sector” as TECHNOLOGY, but the issue was with the “Trade Quantity”, in this cluster the mean is about 100, but the new trade quantity was 1000, this means the trade quantity was away from the mean by about 900, this is shown in the above output.
6.2 **System Testing**

The system testing was done for,

- Machine Learning
- Fraud Detection

In Machine Learning, the front end FMFixSimulator used for historical trade generation, FMTradePublisher, FMTradeSubscriber were started. The initial set of trades were stored in HDFS, all these were verified during the Functionality testing phase here 6.1. Once the data is available in HDFS, the FMMachineLearning application is started up. The data load and machine learning processes are verified through the logging process and finally the output is cross checked with the Weka output to prove the Spark machine learning application worked as expected.

In Fraud Detection, the same sets of applications are required to be running, FMFixSimulator, in this case, it is required for single trade entries or bulk entry is supported as well. FMTradePublisher and FMTradeSubscriber are required for streaming the trades’ real time. The FMMachineLearning need not be running at this time. The FMFraudDetection process is started up and verified through the output that it is consuming real time trades and also it has successfully processed the cluster information from HDFS.
7 Conclusion

7.1 Summary

The original idea was very broader and deeper in terms of implementation. This was reduced to an appropriate design. The approach was to have a thinner implementation of various aspects involved in an end to end trading environment to prove such a traditional setup using a Big Data environment and in a much higher scale will work as expected. The approach has helped visualize a trading environment with a front end, stock exchange, a data storage system for trades, an analytics application for finding patterns using machine learning and finally a real time streaming application to process and detect fraud.

7.2 Evaluation

The feedback about the implementation has been positive mostly. The project started with main focus on the fraud detection in financial markets, but also had other elements to it like the Machine Learning and usage of Big Data and cluster computing technologies. The project was then designed to focus on all these different aspects and provide an appropriate solution. The aim was also to present an end to end solution which will simulate a real trading environment. The final work has satisfied the aim to a good extent. It has a trading simulator which acts like a trade entry front end and a stock exchange.

Due to the unavailability of real trading data, the generation of data became a major part of the project. Lots of effort went into finding the appropriate tool for the data generation and once the data generation was successful, there was a requirement to generate data with some pattern. This pattern was essential to prove that the machine learning process will work fine. Additional effort was required to modify the trading simulator to randomize and add some pattern. At the end this worked out well for the overall execution of the project.

In order to create a real time trading system, the usage of message queuing system was important. This was achieved through the Apache Kafka framework. The effort again was to understand the framework, setting up the framework on a standalone machine, creating a producer, a consumer and a trade serializer. This again was a good effort to present a standalone message queue infrastructure for real time messaging.

The best effort was the implementation of EM and K-means algorithms in the Spark framework. The Spark framework follows a mix of functional and object oriented programming model. The challenge was to learn this new framework, understand the implementation of EM and K-means from Weka and finally implement in Spark.
In terms of the functionality, the system is extendable to add more variables for machine learning, this is a good feature to have if in future additional changes are required in this application, it can be done easily. The downside would be that it is limited to Numerical and Categorical data types, other types like Nominal and Date fields are not supported at this time.

Also, the Spark application runs very slowly on a standalone PC, the real throughput of the application is only visible on a real cluster of nodes. This was not done due to financial constraints and also in the interest of time to get some additional help from the department.

The fraud detection process is very limited in its functionality, it currently only shows the differences between the cluster centroid and the new trade as plain text. This could be better if the original idea was implemented which is to send these to a front end and rerun the machine learning to make better fraud detection in future. Currently, the output is for information purposes only, it is not useful to make the system better, but just acts like a prototype for further improvements. Also, since the fraud detection process is a broader domain with many types of approaches that can be taken, like having a rules engine to validate trades against a predefined set of rules. The set of rules could identify some common patterns of trading activity that was previously identified as fraudulent. This will make the project a combination of an expert system and a machine learning system. The rule engine could be made configurable to add rules dynamically.

### 7.3 Future Work

The system as mentioned is designed as a thinner version of a real fraud detection system. The functionality is still limited to get useful fraud information and process them. Currently this only gives a verbose output. Future work on it could make the output look prettier with a front end application showing the severity of the differences. The thresholds of these differences could be made configurable and appropriate alert can be shown on a front end, this could help the business analyst to get a better idea of the actual alert and its severity.

Apart from this, the system could be designed to have an API which will allow for other types of fraud detection systems to connect and extend the current application. The data is then evaluated from both the current system and then passed on to the add-on system for further checks. This could also be just a rule engine system or another type of machine learning system.

Also, the development effort will be better if the applications are tested on a real cluster to understand the efficiency.
References

FIX Trading

Apache Kafka

Apache Spark

Apache Hadoop & Big Data

Machine Learning
Fraud in Financial Markets


Tighter rules on Fraud activity fall short!

[28] http://www.ft.com/cms/s/0/b9dabde6-a678-11e2-bc0b-00144feabdc0.html

Some major trading loses/fines caused by Fraudulent trading activity

[29] http://www.ft.com/cms/s/0/cba5010a-ccd8-11e1-b78b-00144feabdc0.html
[33] http://www.ft.com/cms/s/0/f57df722-b0e9-11e2-80f9-00144feabdc0.html

A report showing the importance of Investment banking and the economy (Germany)


Paper showing increase in research since 2008 on Fraud detection in Securities Market


LSE can handle over a millions transactions per second

[38] http://www.zdnet.com/blog/open-source/the-london-stock-exchange-moves-to-novell-linux/8285

Miscellaneous

[40] http://docs.oracle.com/javase/tutorial/reflect/
Appendix 1 – Snapshot of System Details

Snapshot showing the PC OS/CPU/Memory details

![System Configuration](image1.png)

**Figure 24. System Configuration**
Snapshot showing System CPU spiking close to max on all cores during Machine Learning

![System Resources](image2.png)

**Figure 25. System Resources**
Appendix 2 – User Guide

In order to successfully run the project, the following steps need to be taken.

On a Linux machine,

Create a user id “hduser”.

The frameworks Hadoop, Spark and Kafka needs to be installed under this user, so they all have the right permissions to access each other’s data.

These tutorials help setup the framework successfully,


The Kafka and Spark frameworks are downloaded directly in the home directory of “hduser”

```
hduser@localhost ~ $ ls -ltr | egrep "kafka|spark"
drwxr-xr-x 12 hduser hadoop  4096 Aug  5 14:46 kafka-0.7.2
drwxr-xr-x 20 hduser hadoop  4096 Aug 16 11:36 spark-0.7.3
```

hduser@localhost ~ $ pwd
/home/hduser

The Hadoop installation defaults to /usr/local/hadoop folder on the Linux machine. The user id for this installation is also “hduser”.

Copy the “MSCProject” project folder into /home/hduser, this folder has a scripts directory which has both the “Compile” and “Run” scripts for all the individual applications except for the FIXimulator and Banzai. These two can be started from Eclipse.

The other applications can be run from the MSCProject folder by calling the corresponding “Run” scripts for various applications.
Appendix 3 – Code Snippets

[1] Sample Instruments.XML document

```xml
<instruments>
  <instrument name="TXU CORP" ticker="TXU" sedol="2885700" ric="TXU.N" cusip="873168108" price="58.56" exchange="LSE" sector="ENERGY"/>
  <instrument name="INTL BUSINESS MACHINES CORP" ticker="IBM" sedol="2005973" ric="IBM.N" cusip="459200101" price="92.07" exchange="LSE" sector="TECHNOLOGY"/>
  <instrument name="YAHOO! INC" ticker="YHOO" sedol="2986539" ric="YHOO.OQ" cusip="984332106" price="32.09" exchange="LSE" sector="MEDIA"/>
</instruments>
```

[2] Sample FIX messages

It is a set of Tag-Value pairs; this message is a FIX login message

```
8=FIX.4.2^9=72^35=A^34=1^49=FIXIMULATOR^52=20130828-20:10:27.443^56=BANZAI^98=0^108=30^10=005
```

FIX message for a trade,

```
8=FIX.4.2^9=230^35=8^34=1975^49=FIXIMULATOR^52=20130829-09:47:25.888^56=BANZAI^1=ABC123^6=33.79^11=1377769646870^14=99^17=E137769647860^20=0^31=33.79^32=99^37=O1377769646873^38=99^39=2^54=1^55=CAN^75=20130829^107=TECHNOLOGY^150=2^151=0^207=NYSE^10=221
```

[3] Trade Object

```java
public class Trade implements Serializable{
  private static final long serialVersionUID = 1L;
  private String symbol = null;
  private String symbolSector = null;
  private int orderQuantity = 0;
  private int tradeQuantity = 0;
  private String side = "Buy";
  private double price;
  private String ID = null;
  private String tradeID = null;
  private String orderID = null;
  private String account = null;
  private String exchange = null;
  private String sector = null;
  private String tradeDate = null;
  private double tradeValue;
}
```
Trade Properties file

The sample trade Java properties file looks like below,

```java
#Trade properties to parameterize exchange/sector
#Fields should be ordered in the same order as numeric and category below

#numeric fields are stored first and then the categorical fields
fields=QUANTITY,EXCHANGE_NYSE,EXCHANGE_NASD,EXCHANGE_LSE,EXCHANGE_HKFE,EXCHANGE_BVSP,SECTOR_BANKS,SECTOR_ENERGY,SECTOR_HEALTHCARE,SECTOR_MEDIA,SECTOR_TECHNOLOGY

#The numeric/category values should match the field names in the Trade object
numeric=tradeQuantity
category=exchange,sector

#All possible values for the categorical fields
exchange=NYSE,NASD,LSE,HKFE,BVSP
sector=BANKS,ENERGY,HEALTHCARE,MEDIA,TECHNOLOGY
```

Data Load from HDFS into Spark JavaRDD

```java
// Spark map function to read in data from HDFS and convert string representation from the HDFS file to a Java RDD (Resilient Distributed Dataset) of Scala Tuples of instance number and the corresponding instance.
JavaRDD<Tuple2<Integer,Vector>> instancesRDD =
    FMMachineLearning.sc.textFile(args[1]).map(
        new Function<String,Tuple2<Integer,Vector>>() {
            private static final long serialVersionUID = 1L;
            @Override
            public Tuple2<Integer,Vector> call(String arg0) throws Exception{
                return parseDataToTuple(arg0);
            }
        }).cache(); // Cache it so it is loaded into memory for faster performance.
```

Sample Map/Filter function to create Training JavaRDD instances in EM

```java
crossValidateTrainingRDD = instancesCopyRDD.map(
    new Function<Tuple2<Integer,Vector>, Vector>() {
        private static final long serialVersionUID = 1L;
        @Override
        public Vector call(Tuple2<Integer, Vector> arg0) throws Exception{
            if((arg0._1 >= 0 && arg0._1 < first) || (arg0._1 >= last)){
                return arg0._2;
            }
            return null;
        }
    }).filter(new Function<Vector,Boolean>() {
        private static final long serialVersionUID = 1L;
        @Override
        public Boolean call(Vector arg0) throws Exception{
            if(arg0 != null){
                return true;
            }else{
                return false;
            }
        }
    });
```
Sample Map/Reduce function to calculate log likelihood in E step of EM

```
logLikelihood = instancesRDD.map(
    new Function<Vector,Double>(){
        private static final long serialVersionUID = 1L;
        @Override
        public Double call(Vector arg0) throws Exception{
            return logDensityForInstance(arg0);
        }
    }).reduce(
    new Function2<Double,Double,Double>(){
        private static final long serialVersionUID = 1L;
        @Override
        public Double call(Double arg0, Double arg1) throws Exception {
            return arg0+arg1;
        }
    });
```

JavaDStream Object to process Kafka Stream

```
JavaDStream<Trade> processTradesDStream =
FMFraudDetection.jsc.kafkaStream(Trade.class, TradeSerializer.class, kafkaParams,topicMap, new StorageLevel());

// For every new “Trade” object that arrives on the Kafka message topic
processTradesDStream.foreach(
    new Function<JavaRDD<Trade>, Void>(){
        private static final long serialVersionUID = 1L;
        @Override
        public Void call(JavaRDD<Trade> arg0) throws Exception{
            if (arg0.count() > 0){
                for(Trade trade: arg0.collect()){
                    if(accountCluster.containsKey(trade.getAccount())){
                        double[] newTrade = new double[numberOfAttributes];
                        int k=0;
                        for(int i=0; i< numericFields.length; i++){
                            methodName = "get" + WordUtils.capitalize(numericFields[i]);
                            method = trade.getClass().getMethod(methodName);
                            newTrade[k++] = Double.parseDouble(Integer.toString((int)method.invoke(trade)));
                        }
                        for(int i = 0 ;i < categoryFields.length; i++){
                            methodName = "get" + WordUtils.capitalize(categoryFields[i]);
                            method = trade.getClass().getMethod(methodName);
                            categoryValues = tradeProps.getProperty(categoryFields[i]).split("",");
                            for(int j=0 ;j<categoryValues.length; j++){
                                newTrade[k++]=
                                (method.invoke(trade).equals(categoryValues[j])
                                ? 1d : 0d );
                            }
                        }
                        Vector newTradeVector = new Vector(newTrade);
                    }
                }
            }
        }
    });
```
// Get the best cluster based on the Euclidean Distance calculator, also performs another distance check based on a separate distance measure for numeric fields and doing an equality check on category fields.
int clusterIndex = findBestCluster(newTradeVector, trade.getAccount());

// Giving verbose output base on the best cluster found from above. At this point there can be more condition checks/raise alerts work done! But deciding to stop at this point!!!! - I am Happy with my Work!!!
verboseTradeInfo(trade.getAccount(), newTradeVector, clusterIndex);
else {
    System.out.println("No Clustering Info Available for : "+ trade.getAccount());
    
    return null;
}

[9] Distance Function in Fraud Detection
for(int i=0; i< clusters.size(); i++) {
    Cluster cl = clusters.get(i);
    double[] centroid = cl.getCentroid();
    boolean cdiff = true; // Category differences
    boolean adiff = false; // At least one category match
    for(int j=0; j< nFields.length; j++) {
        nFields[j] = instance.apply(j);
        nCent[j] = centroid[j];
    }
    double ldist = Double.MAX_VALUE;
    for(int j=nFields.length; j< numberOfAttributes; j++) {
        if(instance.apply(j) == centroid[j] && instance.apply(j) == 1) {
            adiff = true;
            ldist = Math.sqrt(getDistance(new Vector(nFields), new Vector(nCent), account, i));
            if(cdiff) {
                cdiff = true;
            }
            else if (instance.apply(j) != centroid[j]) {
                cdiff = false;
            }
        }
        if(cdiff){
            bestCluster = i;
            break;
        }
        else if(adiff){
            if(ldist < minDist) {
                minDist = ldist;
                bestCluster = i;
            }
        }
    }
}

return bestCluster;