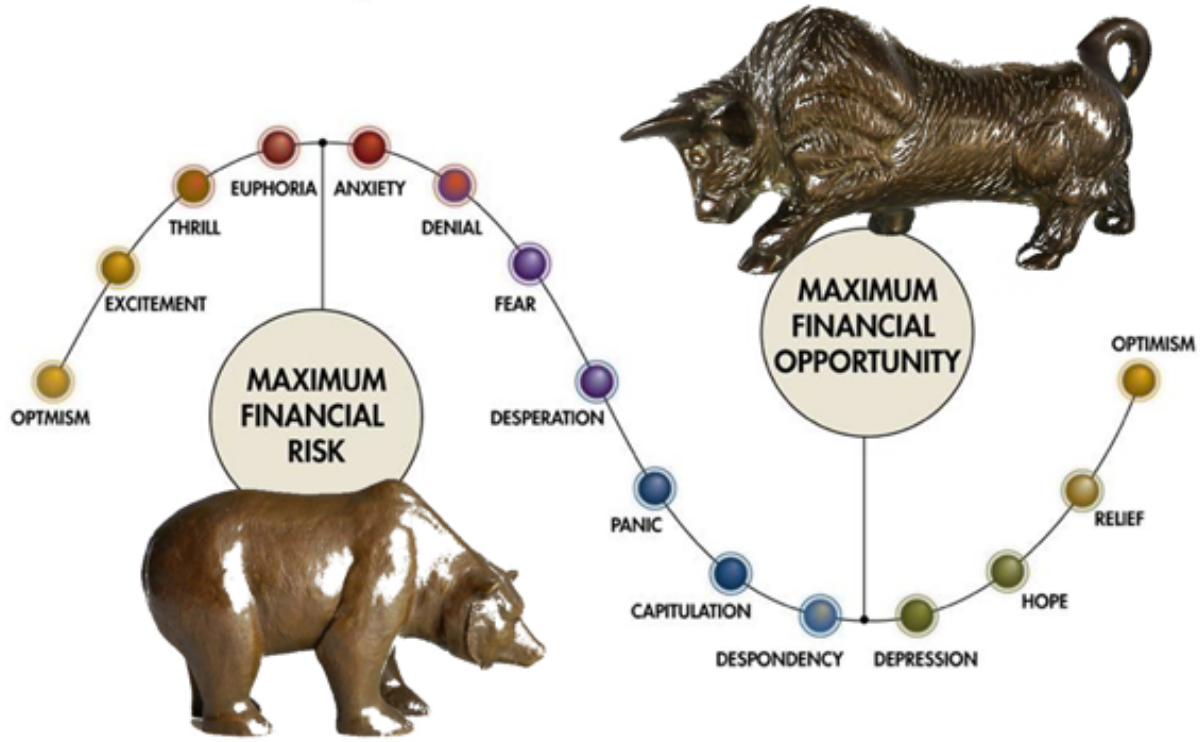
EB5001

Stock Analysis using Big Data Engineering for Analytics

Our Journey and Learnings



TEAM 1

ANURAG CHATTERJEE (A0178373U)
BHUJBAL VAIBHAV SHIVAJI (A0178321H)
GOH CHUNG TAT KENRICK(A0080891Y)
LIM PIER (A0178254X)
LIU THEODORUS DAVID LEONARDI (A0178263X)
TEO WEI KIN DARREN (A0178197L)
TSAN YEE SOON (A0178316Y)

Contents

1 Executive Summary

One of the most challenging, yet important tasks in the financial sector is financial stock analytics. Traditional methods focused on quantitative analytics. However, data such as twits and other unstructured big data sources is important, which requires the organisation to build scalable well-architected and engineered big data system for analytics.

The project highlights the objectives of our big data system for publicly traded US equity analytics. The process is challenging for the team, but under the guidance of our lecturers, we managed to build the big data analytics system successfully. This report highlights our learning journey in building the financial big data analytics system.

2 Introduction

2.1 Background

Financial assets worldwide are estimated to be worth over US$300 trillion, with worldwide publicly traded companies market capitalisation at nearly US$100 trillion [1]. With so much at stake, it is not surprising that one of the most important research in the financial sector is to obtain accurate updated or real-time forecast of financial instrument prices.

Traditionally, the forecast methodology for financial instruments, such as publicly traded equities, are quantitative and statistical methods such Time Series forecast and Stochastic techniques [2] [3] [4], analysing on just pure quantitative and structured data inputs such as historical prices data and financial report figures. In recent years, researchers and traders have started to incorporate real-time unstructured information such as twits, real-time news [5] [6] into their forecast systems and use other more advance machine learning techniques for the prices forecast. This meant that the forecasting system not only must be capable to be scalable pipelines with real time processing and it must also have a scalable well designed Data Lake for all the data. In addition, the entire big data architecture must also integrate analytics processing capabilities.

2.2 Objective

The main objective of this project is to:

1. Design and implement a scalable big data system with well-designed data lake that is able to ingest and store both real-time traditional structure price data and real-time unstructured data
2. The big data system must be capable of real-time stream processing to extract insights from real-time financial data
3. Integrate machine learning techniques to forecast financial instrument prices

The expected output of the system must be able to give users a variety of insights on the financial instrument.

- Real Time Streaming insights/analytics
- Dash board for near real-time or batch processing analytics and price forecast of the financial instruments

2.3 Scope of the project

From our initial research and investigation, archiving accurate price forecast with big data analytics systems is still very much a research-in-progress work and even if there are successful systems, the developers of the successful systems will most likely not divulge their “secret source”.

For the purpose of this project, the first version of our deliverable will be focus on building the scalable pipelines and data lake. Analytics will be implemented with machine learning techniques that we have researched, and real-time data stream processing will also be implemented with the US publicly traded equity prices data [7] and StockTwits data [8] as its initial use case. With this well-designed and engineered big data system, it will be easy to expand on the analytics in future work.

3 Implemented Big Data Architecture

3.1 Implemented Architecture

The overall landscape looks like the below in terms of Big data technologies and the proposed functionalities.

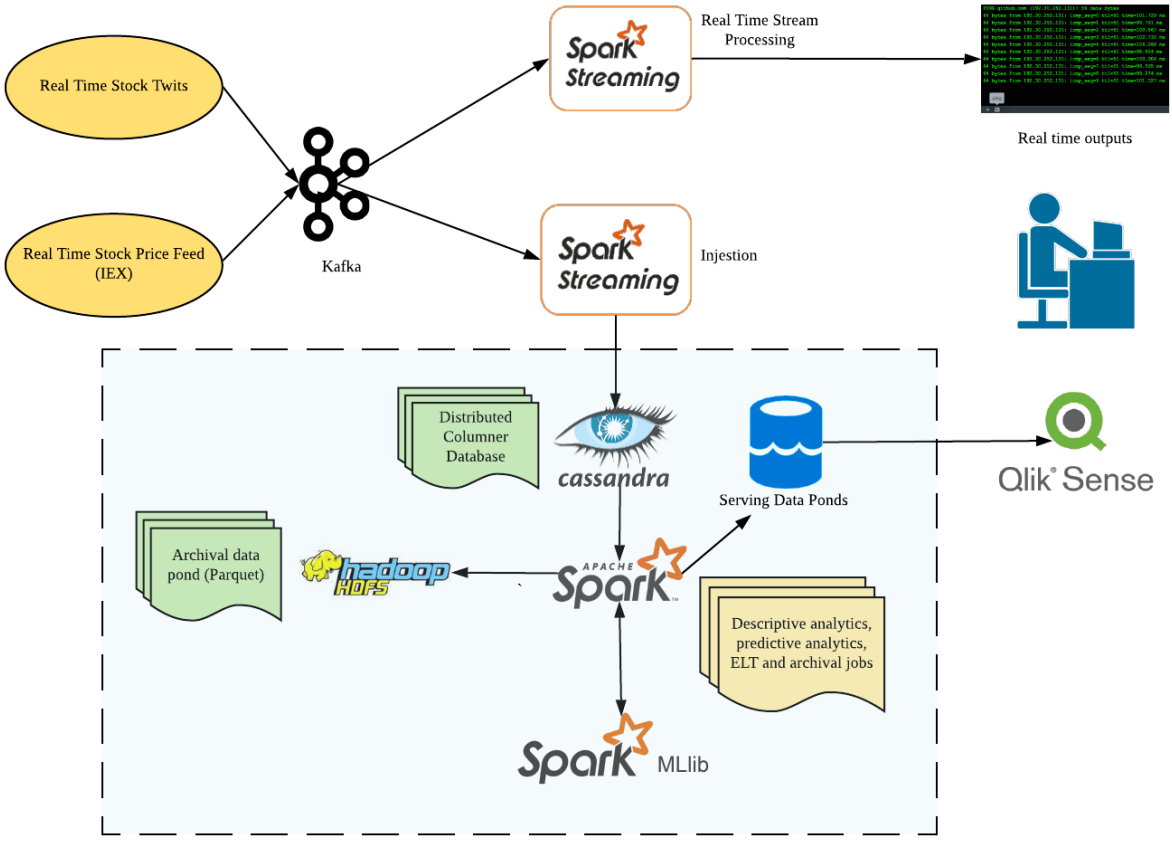


Figure 1 Big data technologies architecture

There are two real-time data producers which fetch data from 2 different REST APIs. The stock quote API provides updates on the real-time price of the stocks from IEX and the stock twits API provides tweets related to the stock. The producers continuously fetch responses from these APIs and push the retrieved

JSON to Kafka topics. The data from the Kafka topics are then processed by Spark streaming jobs in real-time. There are two category of jobs, one that performs real-time aggregations and visualize in a console and other that pushes to Cassandra. The data at rest in Cassandra are then processed by 2 categories of Spark batch jobs. The first category performs aggregations on the static data and the other category performs batch machine learning. The results from these batch jobs are saved in separate tables in Cassandra. A separate batch job performs archival by routinely converting the data stored in Cassandra to Parquet files. Qlik Sense is used to visualize the results of the batch processing into a dashboard using the Cassandra connector. The below sections focus on the learnings that were had when the above architecture was implemented.

4 Data persistence strategy

After researching all the available options, we used Cassandra as our main database, for the following reasons:

- It is a columnar NoSQL database, which lends itself well to analytics of big data.
- We are able to use CQL to do quick querying, even if the data is spread across the clusters.
- It is data-center aware, and we are able to have Cassandra clusters.
- Many reputable companies use Cassandra, giving us confidence in its capabilities.
- We created the schema for the Cassandra tables such that combination of the partition key and the clustering key ensured that the data could be sharded and queried effectively. For example, for the results of the descriptive analytics jobs the partition key was the JOB ID and the partition key a unique ID which ensured that while visualization the results from a specific partition could be retrieved which were ordered by the unique ID.

5 Setting up the Big Data Environment and Tools

5.1 Cloudera Distributed Hadoop

Initially, we are going to use Cloudera Distributed Hadoop (CDH) version 5.13. We found this image very resource heavy, and it was a challenge getting it to run properly on any of our machines. We were initially able to run the image on a desktop with 16GB of ram, but when some spark jobs were running the processes slowed down considerably. We then decided to look at other Hadoop packages and distribution formats.

5.2 Hortonworks Data Platform

After using Cloudera Distributed Hadoop, we decided to use the Hortonworks Data Platform Docker distribution. We deployed this docker image on EC2 instance with 32GB of RAM alongside Cassandra and Redis dockers. Later we found out that we need to upgrade the EC2 instance type to r5a.2xlarge which has 8 vCPU and 64GB of RAM to accommodate all of the running docker instances.

Another issue that we face is difficulty to access Cassandra server from another docker container which has Spark in it. Thus, we finally decided to install Hadoop, Spark, Kafka, and Zookeeper locally on our machine.

6 Coding and Debugging Big Data in Scala

6.1 Writing Spark jobs using Scala

Initially, it was difficult to make the Scala Spark set up work in the local systems for development. We found that it was faster to set up the Scala and Spark development environment using IntelliJ since it handles the build of the Scala applications itself. However, we still faced several issues during Scala development using IntelliJ:

a. Scala SBT takes a lot of time to cache all the dependencies during project load.
b. A lot of files are generated by the IDE that makes it necessary to be careful while committing the code to source control.
c. We needed to be very careful about the versions mentioned in the build file to ensure that the Spark jobs execute properly later when the application is built into a JAR and deployed as a Spark job. For example, if you use an old version of Spark and it is dependent on classes only available in the new version, it will cause a java "class not found" error when trying to run the jar files via spark-submit in the target machine.

6.2 Debugging Scala Spark Jobs

There were instances where running the spark job itself took some time, and errors were not immediately evident due to the large amount of messages being printed in the console. One way we felt was useful to do effective debugging was to step through the code line-by-line in the IDE (IntelliJ). In this manner, we were able to look at the individual variables in a step-by-step manner allowing us to understand what the Scala code was doing. This was invaluable to us considering that most of us had only just started with Scala and Functional Programming.

6.3 Scala Spark Streaming Debugging

As an additional complexity, it was not always easy to debug streaming jobs. We found that even stepping through the code did not suffice in this case. We had to do logging in order to see what was happening in the code in order to debug it effectively.

6.4 Functional Programming learnings

Coming from a Python background it took some effort to understand the Functional constructs in the Scala language and the Scala Spark applications.

One of the ways where we saw considerable value in using functional programming is the way we could chain up multiple commands and transformations in a line like this:

```
predictions = predictions.withColumn( colName = "symbol", lit( literal = "AAPL"))
  .withColumn( colName = "id", monotonically_increasing_id())
  .withColumn( colName = "job_at", lit(System.currentTimeMillis / 1000))
  .withColumn( colName = "date", lit(from_unixtime( ut = $"epoch", "yyyy-MM-dd")))
  .withColumn( colName = "hour", lit(from_unixtime( ut = $"epoch", "HH")))
  .select( col = "symbol",  cols = "id", "job_at", "epoch", "date", "hour", "marketaverage", "prediction")
```

This gave us the mindset of thinking about programs like a mathematical function, where there are no intermediate variables holding the values.

7 Ingestion

7.1 Limitations of the source APIs

The IEX stock data API sends valid response only during the US stock market opening hours: 10:30 PM – 5:00 AM (9:30 PM – 4:00 AM during daylight saving time) Singapore time. Since we could not rely on the API to get streaming data during weekends which was during our presentation, we simulated the streaming input by getting a response for a specific date which was an array of JSON objects and then writing the producer such that it returned one object at a time from the received response. We set the interval in which records were sent from the endpoint as 1 second.

With this setup, we are able to create a streaming API in a controlled environment. The initial interval that we had set is 10 seconds and subsequently, we have reduced the interval to 1 second per record. This allows us to stress our Kafka clusters and in turn show the robustness of the server that we have set up. It is noted that with the short interval of data being sent by our producers, there are not visible breakdown in the service.

The StockTwits API only return 30 tweets on one API call. To avoid duplication of records pulled from the API we cache the previous query parameter to the API. We needed to introduce Redis as the caching technology.

7.2 Unix DateTime

The two data source that we are using have different formats as well as schema, the time format. Thus in order to standardize the formats for easy data digestion, we have used the Unix DateTime which is favoured by our chosen database Cassandra.

The following example shows the different time format for IEX data source and StockTwits are shown below:

IEX Data for Apple Stock Market, 15th April 2019 , 0930


```
{"date":"20190415","minute":"09:30","label":"09:30 AM",
"high":199.43,"low":198.51,"average":199.054,"volume":5051,
"notional":1005421.01,"numberOfTrades":58,"marketHigh":199.43,
"marketLow":198.51,"marketAverage":198.765,"marketVolume":500842,
"marketNotional":99549678.06,"marketNumberOfTrades":985,"open":198.53,
"close":199.31,"marketOpen":198.58,"marketClose":199.31,"changeOverTime":0,
"marketChangeOverTime":0}
```


An Example of one of the tweets from StockTwits


```
 "messages": [
  {
    "id": 64,
    "body": "an idea about a $JOY",
    "created_at": "2012-10-08 21:42:33 UTC",
```

```
"user": {
 "id": 2,
 "username": "ppearlman",
 "name": "Phil Pearlman",
 "avatar_url": "http://avatars.stocktwits.com/images/default_avatar_thumb.jpg",
 "avatar_url_ssl": "https://s3.amazonaws.com/st-avatars/images/default_avatar_thumb.jpg",
 "identity": "Official",
 "classification": [
  "ir",
  "suggested"
 ], ....
```

We will convert both of them into the same time format namely the UNIX time format for standardization.

7.3 Parsing Json using Scala

There are many JSON parsers available for Scala, and they are not consistent with the libraries we found on other programming languages like Python. Hence, parsing the JSON using Scala is challenging, and we had to resort to a method similar to XPath but for JSON to get to the relevant fields especially in heavily nested JSON responses which was the case for the stock quotes API response.

We tried spray-json from Spray project, play-json from Play Framework, Google GSON and json4s. We decided to use Json4s since it allows us to easily access JSON key and value like this:

```
val JInt(x) = json \ "foo" \ "bar"
```

In order to convert parsed JSON values back to JSON string, we have to use Java HashMap and Google GSON Builder.

```
// This is how we utilize HashMap to generate key-value pair dictionary
val jsonMap: HashMap[String, Any] = new HashMap[String, Any]()
val jsonId: Int = getInteger(message \ "id")
val jsonBody: String = getString(message \ "body")

// This is how we convert the HashMap to JSON string
val gsonMapBuilder = new GsonBuilder
val gsonObject = gsonMapBuilder.create
val jsonObject = gsonObject.toJson(jsonMap)
```

Scala is a functional programming as well. As such variables are immutable. In the case of the IEX data, we are required to create multiple variables to store the keys of the JSON document and create the UNIX Time format from there. Only after which can we create a new JSON document to be pushed into the Cassandra database. This method of transforming the data into something useable is crude and cumbersome.

8 Real Time Stream Processing

8.1 Determine the Optimal Solution for computing Moving Averages for financial data

The Spark streaming API is not consistent, documentation not well enough and since we wished to use the latest abstractions like dataframe, relevant examples were not readily available in the internet.

9 Machine Learning Analytics

9.1 Feature Selection

In our code, we only used one feature, marketAverage for our predictive model. On hindsight, this could be coupled together with features obtained from StockTwits' ingested data in Cassandra to give better predictions.

We initially wanted to use the spark-ts library to do time-series prediction. After spending some time with it and getting some errors, we decided to consult the author about the library.

Sandy Ryza <sandyryza@gmail.com>
to me, Time

Hey Pier,

That's correct - this library is no longer under active development and has lots of gaping holes.

It looks like Flint (https://github.com/twosigma/flint/commits/master) has had some more recent development.

It also depends what you mean by "do time series". If you're looking for time series statistical models, and have

-Sandy

After looking at the Flint library, it seemed that it was under active development. Due to time considerations, we decided to do a simple regression on the data using the in-built SparkML libraries.

In Spark ML, predictive analytics features into an algorithm like RandomForestRegressor need to be packed into a vector using VectorAssembler. This is unlike the Scikit-Learn format, where this process (if required) has been abstracted away for you.

10 Descriptive Analytics using batch processing

We used the Dataframe abstraction for writing the aggregations for the raw data from the Cassandra tables as part of our descriptive analytics using Batch processing. This abstraction was easier to process compared to the RDD abstraction. Also, initially we spent some effort to obtain the correct Scala-Spark-Cassandra connector and finally we resorted to the DataStax Spark Cassandra connector.

11 Visualisation

11.1 Connectors

There is no pre-built connector from Cassandra to Qlik Sense. The initial plan was to create a separate MySQL database for which the pre-built connector is available. However, there were difficulties to connect Cassandra to MySQL. Ultimately, we managed to find an ODBC connector that can connect direct from Cassandra to Qlik Sense, which is a better solution.

11.2 Schema

Cassandra stores multiple data transformation jobs. However in the visualization tool, only the output of the latest job is required. Therefore, using the query engine within Qlik Sense, it only extracts the data for the latest job. To extract the latest batch job, the max value of the "job_at" was first added to the table within Qlik Sense before the "WHERE" clause is added to filter out the latest job.

12 Final thoughts

- We can explore managed ETL cloud services like AWS Glue to see how it might save some work in building big data pipelines.
- Wherever applicable, we will try to leverage on managed cloud services:
  - Infrastructure will be managed by the cloud provider wherever possible
  - To allow us to concentrate on the business functionality and coding the business functions
  - Scalability and availability are typically taken care of when managed cloud services are used
- Trend seems to be moving towards a container-centric world:
  - We can explore Kubernetes with Spark jobs to see whether it is easier to deploy Spark jobs with such an architecture

13 Bibliography

[1] "Global market cap is about to hit $100 trillion," Business Insider, 2017. [Online]. Available: https://www.businessinsider.sg/global-market-cap-is-about-to-hit-100-trillion-2017-12/?r=US&IR=T.

[2] J. . Zhang, R. . Shan and W. . Su, "Applying Time Series Analysis Builds Stock Price Forecast Model," *Mathematical Models and Methods in Applied Sciences,* vol. 3, no. 5, p. 152, 2009.

[3] R. . Weron and A. . Misiorek, "FORECASTING SPOT ELECTRICITY PRICES WITH TIME SERIES MODELS," *Econometrics,* vol. , no. , p. , 2005.

[4] C. N. Babu and B. E. Reddy, "A moving-average filter based hybrid ARIMA-ANN model for forecasting time series data," *Applied Soft Computing,* vol. 23, no. , pp. 27-38, 2014.

[5] M. . Arias, A. . Arratia and R. . Xuriguera, "Forecasting with twitter data," *ACM Transactions on Intelligent Systems and Technology,* vol. 5, no. 1, p. 8, 2013.

[6] R. . Feldman, B. . Rosenfeld, R. . Bar-Haim and M. . Fresko, "The Stock Sonar — Sentiment Analysis of Stocks Based on a Hybrid Approach," , 2011. [Online]. Available: https://aaai.org/ocs/index.php/iaai/iaai-11/paper/viewpaper/3506. [Accessed 22 4 2019].

[7] “IEX Trading Executive Team,” , . [Online]. Available: http://www.iextrading.com/about/. [Accessed 22 4 2019].

[8] . . Staff, “StockTwits App Integrations,” , . [Online]. Available: http://stocktwits.com/developers/docs/integrations. [Accessed 22 4 2019].