CDFMR: A Distributed Statistical Analysis of Stock Market Data using MapReduce with Cumulative Distribution Function

Devendra Dahiphale*, Abhijeet Wadkar†, Karuna Joshi‡
Department of Computer Science and Electrical Engineering *†,
Department of Information Systems ‡,
University of Maryland, Baltimore County, USA
Email:{devendr1, abhi6†, karuna.joshi‡}@umbc.edu*

***Abstract*—The stock market generates a massive amount of data on a daily basis, in addition to a deluge of historical data. As a prime indicator of our global economy, investors and traders look to stock market data analysis for assurance in their investments. This has led to immense popularity in the topic, and consequently, much research has been done on stock market predictions and future trends. However, due to relatively slow electronic trading systems and order processing times, the velocity of data, the variety of data, and social factors, there is a need for gaining speed, control, and continuity in data processing (real-time stream processing) considering the amount of data that is being produced daily. Processing this massive amount of data on a single node is inefficient, time-consuming, and unsuitable for real-time processing. Recently, there have been many advancements in Big Data processing technologies such as Hadoop, Cloud MapReduce, and HBase. This paper proposes a MapReduce algorithm for statistical stock market analysis with a Cumulative Distribution Function (CDF). We also highlight the challenges we faced during this work and their solutions. We further showcase how our algorithm is spanned across multiple functions, which are run using multiple MapReduce jobs in a cascaded fashion.**

***Index Terms*—Big Data, MapReduce, Hadoop, Prediction, Stock Market Analysis, Distributed, Streaming, Probability Distribution, Cascaded Jobs.**

I. Introduction

In today's data-driven world, algorithms and tools are continuously collecting data about everything around us, for example, humans, systems, processes, and organizations. Consequently, it increases the velocity and variety of data. Therefore, it must be processed efficiently and continuously to get meaningful insights. This is where MapReduce [1] framework comes into the picture. MapReduce is a programming model and an associated implementation for processing big data sets in a parallel and distributed fashion. A MapReduce program comprises a map phase, which performs filtering of input data sets and sorting, and a reduce phase, which aggregates the data produced in the Map phase.

The approach of predicting stock market trends has tantalized people for many years; the gist of the problem lies in huge data sizes, processing speed, network delays, various data sources (such as tweets and stock indices), and streaming [2] nature of the data. Hundreds of millions yearly invest in the stock market, ETFs, Bonds, or NFTs worldwide to make some quick bucks. However, a company's stock prices fluctuate regularly; investing is a calculated risk, but it deters many from investing. Since the beginning, professional analysts have tried to predict future trends for companies or a particular sector.

As we enter into a new era of Big Data [1] and Big Data processing frameworks, the ways we process stock market data and make predictions [3] has changed. For example, social media feeds have begun outlining future trends in companies' values, and stock exchanges are beginning to outsource their traditional RDBMS to cloud providers for faster data manipulation, reliability, maintenance, and cost savings. While social media prediction analysis is beyond the current scope of this paper, we did make use of cloud technologies (Hadoop [4], HBase [5], MapReduce [1]) and implemented a MapReduce algorithm for statistical stock market analysis with Cumulative Distribution Function (CDF) to find future trends in the stock market.

We started by organizing the company's historical data from the NYSE and the NASDAQ stock exchanges. To perform data analysis, we implemented cascading [6] MapReduce [1] algorithms. With the final analysis, we display the results in a Web-App, showing price, volume, and prediction graphs. Our result is presented via an interface (see figure 7), allowing anyone to search for a company and determine whether or not they should invest in a particular company (with some probability) a user is interested in. The following is an in-depth look at the technologies and algorithms we have used and the process yielding our results.

II. Background Survey

A. MapReduce

MapReduce [1] is a programming model developed by Google for processing big data sets in a distributed fashion. Mapreduce operates in two phases: first, a Map phase, in which input data is split into multiple chunks/splits, and for processing each chunk/split, a mapper (a thread that applies the Map function on data) is spawned which uses the user-defined function for filtering and shuffling rows/records(a record is a key-value pair), called the Map function; and second, a Reduce phase, which aggregates the data produced in the Map phase (by mappers) to generate the final output. The

Reducers (workers from the Reduce phase) aggregate all the rows/records with the same key.

Hadoop Online Prototype (HOP) [7] is one of the widely used MapReduce implementations. There are many MapReduce implementations, for example, Hadoop [4], Apache Spark [8] etc. Hadoop Online Prototype [7] [9] [10] is a modification to the traditional Hadoop implementation. In traditional Hadoop, Map and Reduce phases run sequentially - first Map phase finishes, and then Reduce phase starts. However, in Hadoop Online Prototype, the Map and the Reduce phases run simultaneously - making the overall system more efficient. As HOP is well-proven, most popular, and efficient in processing Big Data sets, we decided to use it for processing stock market data sets.

B. Hadoop Distributed File System (HDFS)

HDFS [4] is a distributed file system that provides reliable and scalable data storage. It is designed explicitly for spanning large clusters on the commodity hardware. Hadoop uses HDFS for data storage in the backend.

C. Prediction Analytics

Currently, there are many approaches for stock market prediction [11], either with different processing or data sources. The processing stock market data approaches are categorized into two groups. In the first category, the emphasis is given to statistical analysis [12] of a company's stock price movements from the past. On the other hand, in the second category, in addition to statistical analysis, the focus is also given to machine learning [13] techniques.

To improve the prediction accuracy of the stock prices, some currently used algorithms also consider social, political, economic, and environmental factors. For example, Karl [11], R. J. Kuo [14], and Kimoto T. [15] in their independent research use Neural Network approaches. Rohit Verma [16] in his work uses a neural network approach for stock market prediction. Moreover, he uses standard Error Correction Neural Network (ECNN) [16] [14] for daily forecast, whereas an extension of ECNN for weekly predictions. Our predictions only use statistical data with Score Adjust Factors, which vary according to daily stock price movements. For long-term investments, we calculate the rank for each company, where rank decides how a particular company is likely to perform to other companies from the given data. In addition to calculating the positions of companies, statistical analysis was also done to predict the next day's stock price for each company with all probabilities from 0 to 1.

D. Cumulative Distribution Function (CDF)

In probability theory and statistics, the cumulative distribution function (CDF) of a real-valued random variable X, or just the distribution function of X, evaluated at x, is the probability that X will take a value less than or equal to x. We will use the CDF on the percentage change of stock price values for a day, calculated for a few years of data.

$$F_X(x) = P(X \leq x)$$

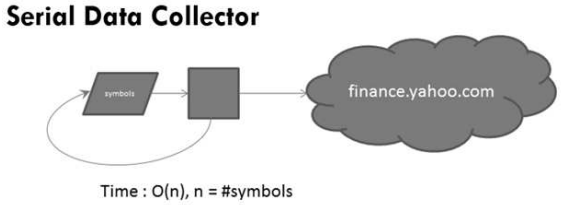


Fig. 1: Serial Data Collector

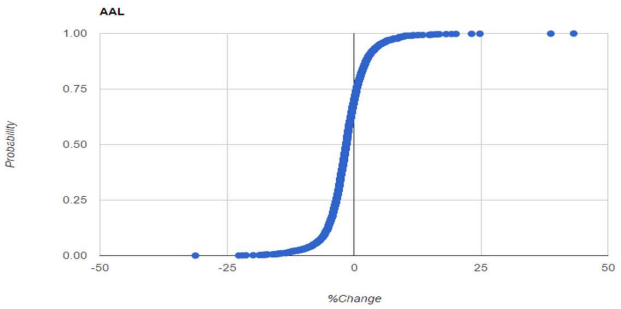


Fig. 2: Probability Distribution Graph: Predicting Stock Price Movement Statistically using CDF

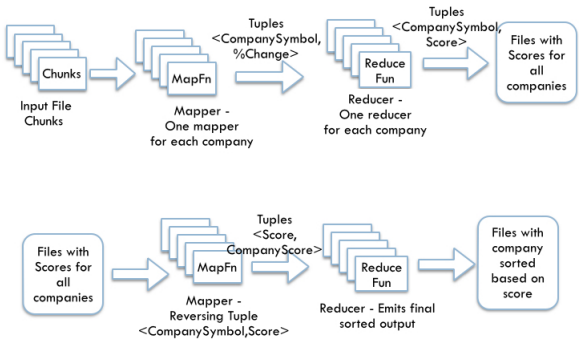


Fig. 3: Data Flow and Algorithm Processing

$$F_X(x) = function\ of\ X$$

$$X = real\ value\ variable$$

$$P = probability\ that\ X\ will\ have\ a\ value\ <=\ x$$

III. Our Contribution: Prepossessing and Algorithm

A. Data Collection and Preparation

As a part of data collection and preprocessing, we identified what data sets are available for the stock market. Most data sources require paid subscriptions, but the historical data sets on finance.yahoo.com are free of cost. As depicted in figure 10, the historical data set contains the date, opening price, highest in a day, lowest in a day, closing price, the volume traded, and adjusted volume for each stock symbol. First, we collected all

current ticker symbols from NYSE and NASDAQ stock exchanges. Then, we developed a multi-threaded application for downloading and collating historic stock market-related data using the Hadoop framework. In this multi-threaded approach, many Hadoop Mappers (mapping threads) use ticker symbol lists as input and simultaneously collect historical data from the Yahoo Finance site. However, when the Mappers made multiple HTTP requests to the server, the server interpreted it as an attack and rejected the HTTP requests. Therefore, we replaced the multi-threaded data collectors with serial versions of the data collector if we only needed sample data for one year. And in other cases, a limited number of threads in an application. Limiting the number of threads is done by allocating a set of Tickers to a thread rather than one thread for one ticker model. We have collected historical data for approximately 3000 NASDAQ and 1450 NYSE ticker symbols. The original data set is in comma Separated Value (CSV) file format.

Furthermore, we developed an application for raw stock market data and stored it in the HBase [5] database. The unique key is the combination of the exchange symbol, the ticker symbol, and the date (even though the ticker itself is supposed to be unique, we added this provision for more safe approach). We also developed a CSV file generator application that combines all raw data files into one single CSV file with a corresponding exchange symbol and ticker symbol prefix to each record. We could generate a unique key from the first three columns: the exchange symbol, the ticker symbol, and the record date. After preparing data into CSV format, the files are upstreamed to the Hadoop Distributed File System (HDFS) for the analysis step. We have collected and preprocessed approximately 15 million records. The preprocessing is done using multiple MapReduce jobs.

Algorithm 1 Predicting Stock Price Movement Statistically for a Company (Ticker).

procedure PREDICTOR($ticker, close\_prices$)
    $number\_of\_days \leftarrow len(close\_prices)$
    $ptg\_changes \leftarrow [\ ]$

    for $<d\ in\ range(1,\ number\_of\_days)>$ **do**
        $today\_price \leftarrow close\_prices[d]$
        $yesterday\_price \leftarrow close\_prices[d-1]$
        $price\_change \leftarrow today\_price - yesterday\_price$
        $ptg\_change \leftarrow ((price\_change \div today\_price) * 100)$
        $ptg\_changes.append(ptg\_change)$
        $sort\ ptg\_changes\ values\ descending\ order$
    end for

B. Serial Statistical Analysis

We first created a serial algorithm (for running on a single node) based on current approaches for predicting stock price movement statistically. Specifically, we used Cumulative Distribution Function (CDF) described in the background survey. The serial algorithm that we wrote for statistical analysis on a single node for the analysis of a company is shown in algorithms 1 and 2.

Algorithms 1 and 2 can be used for one ticker symbol. This must be repeated for all the tickers we used for analysis in our work. Processing this big data for all the companies on a single node is impractical and not very useful to take action in real-time. Therefore, further, we fit this algorithm by adding more algorithm steps (for aggregation and re-processing until we get the final results) to the MapReduce framework using multiple MapReduce working towards achieving one goal. The algorithm 2 queries the data represented using the graph in figure 2.

C. Distributed Statistical Analysis

The data analysis is done using three different MapReduce Jobs (including the data collection and pre-processing, we developed five Map Reduce jobs; however, we are focusing on three primary data analysis MapReduce jobs). The three MR Jobs are 1. Calculates percent change for each company for each day, 2. The second MapReduce job is for sorting the output generated by the first MR job according to calculated scores for each company. and 3. The third MapReduce job calculates the probability distribution [23] of stock price movement. All the jobs are run in cascaded [6] [17] fashion. The cascaded MapReduce [6] jobs enable us to run all the jobs under an umbrella job where the output of one job is used as input to the next job and so on. Eventually, the results are available to the end user via a simple interface. The overall approach and data flow are quickly presented in figure 4.

Algorithm 2 Predicting Stock Price Movement Statistically Query Function.

procedure QUERY($price, ptg\_changes$)
    $days \leftarrow len(ptg\_changes)$
    $split\_index \leftarrow nearest\ location\ in\ ptg\_changes$
    $probability \leftarrow (split\_index \div days) \times 100$

    $return\ probability$ ▷ Probability that price will be greater or equal to price.

1) MapReduce Job 1: The first MapReduce job is to find companies' scores and percentages times the stock price increased. These two scores are important for deciding the company's rank. The company with the highest rank will likely be the best to invest in.

Map Function:

A data prepared and stored on the HDFS [4] [5] is given as input to the first MapReduce job. The directory contains multiple files representing a company's historical stock trading data. The percentage change in stock price each day is used to calculate the score value. For a record R, opening price OP, and closing price CP, the percentage change PC is calculated using the formula as $PC = (CP - OP)/OP) * 100$.

The Map function for all the mapper threads is presented in the code 1; all the threads will apply this function to the input chunk of data allocated to them. Here, the Map function calculates the percentage change in the stock price for each day for a company (represented as a ticker symbol). Output is a daily record in the $\langle Key,\ Value \rangle$ pair where the key is the company symbol and the value is the percentage change in a day for that company.

Listing 1: MR Job1 - Map Function

```
void map(key, value, Context context) {
 for(String line : list) {
  String[] columns = line.split(",");
  double opening_price =
      Double.parseDouble(columns[1]);
  double closing_price =
      Double.parseDouble(columns[6]);
  double percentage_change = ((closing_price -
      opening_price) / opening_price)*100;
  context.write(new Text(filename), new
      DoubleWritable(percentage_change));
 }
}
```

Reduce Function:

The Reduce Function is presented in the code section 2. The records from all the mapper threads with key-value pairs are processed by the reducer threads (which output record from a mapper goes to which reducer is decided based on the hash value of a record key, therefore all the records with the same key will go to the same reducer thread - this is required in the aggregation of records with the same key). The Reduce function calculates a score for each company. Three different values are used to calculate the score viz. 1. Score Adjust Factors is used 1. Up Adjust Factor, 2. Down Adjust Factor, and 3. Percentage Change Value. The score-adjusting factors are tuned according to stock price movements daily. Figure 6 shows that feedback on stock price movement is used to adjust the 'Score Adjust Factors.' When the stock price increases (percentage change is positive), the 'Up Adjust Factor' increases, and the Down Adjust Factor decreases. On the other hand, when the stock price is decreased (percentage change is negative), the 'Down Adjust Factor' increases, and the 'Up Adjust Factor decreases.' So far, the score for a company is adjusted linearly. In addition to Up and Down adjustment factors, the percentage change amount in the stock price is also considered for calculating the score.

Listing 2: MR Job1 - Reduce Function

```
void reduce(Text key,
    Iterable<DoubleWritable> values, Context
    context){
 for (DoubleWritable val : values) {
  total_records++;
  if(val.get() > 0) {
   score += up_adjust_factor + val.get();
   times_increased++;
   // tune Up and Down adjust factors
  } else if (val.get() < 0) {
   score -= down_adjust_factor + val.get();
   // tune Up and Down adjust factors
  }
 }
 // prepare composite value and emit the
     output record
}
```

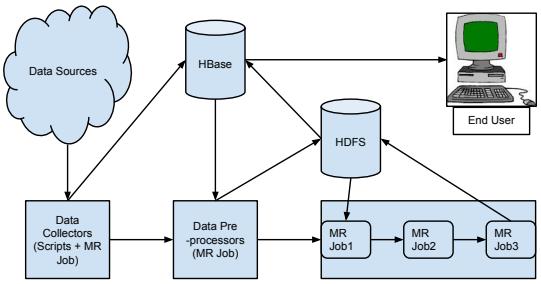


Fig. 4: Data Flow

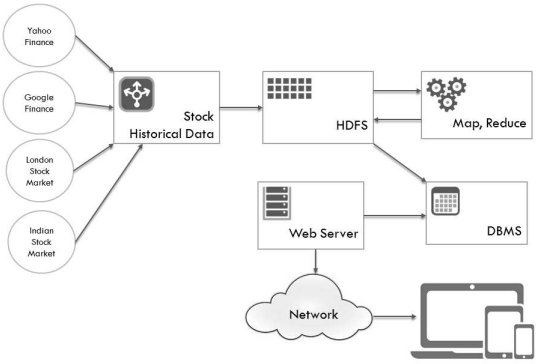


Fig. 5: Service Architecture

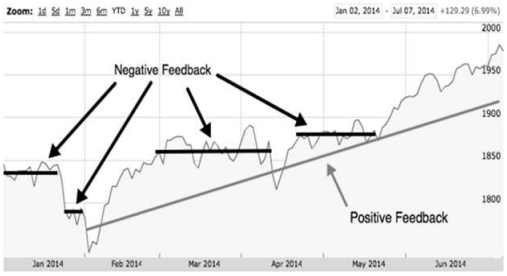


Fig. 6: Adjust Factors. 1) Up Adjust Factor 2) Down Adjust Factor 3) Percentage Change Value

2) MapReduce Job 2: The second MapReduce job is for sorting the output generated by the first MR job according to calculated scores for each company. The final output of this MapReduce job will have all the companies in order of their rank. In the visualization (please see figure. 7) section, we will see that our website/user interface shows the top five companies on the home page; however, the user has given a provision for searching other companies' details through the same portal using a search window.

Map function: The Map function for the second MapReduce job is shown in code listing 3. The map threads take output produced by the MapReduce job1 as input. For each record, the mappers create a composite key (StockKey [11]) using the company's name and score, whereas the value is a percentage number that the stock price increased.

Listing 3: MR Job2 - Map Function

```
public void map(Object key, Text value,
    Context context) {
 String[] result =
    value.toString().split("\n");
 for (String line : result) {
  String[] columns = line.split("\\s+");
  // floor all the values for convenience
  double score =
      Double.parseDouble(columns[1]);
  double percentage_times_up =
      Double.parseDouble(columns[2]);
  context.write(new StockKey(columns[0],
      score), new
      DoubleWritable(percentage_times_up));
 }
}
```

As shown in the code snippet 4, to sort all the records by value (that is, a company's score from records) rather than a ticker, we had to override the default record comparator function. Instead, we used the composite key (Stock Key) comparator - to compare a record by the score (value from the record) rather than the symbol (key from the record). This method is overridden in CompositeKeyComparator and NaturalKeyGroupingComparator classes from the Hadoop framework, which are given as inputs to the MapReduce job in the setup phase.

Listing 4: MapReduce Job2 - Composite Key Comparator

```
class NaturalKeyGroupingComparator extends
    WritableComparator {
      protected
          NaturalKeyGroupingComparator() {
          super(StockKey.class, true);
      }
      @Override
      public int compare(WritableComparable
          w1, WritableComparable w2) {
       StockKey k1 = (StockKey) w1;
       StockKey k2 = (StockKey) w2;

       return (
       k2.getValue().compareTo(k1.getValue()));
   }
}
```

Furthermore, we have provided NaturalKeyPartitioner class to the job; please refer to the code snippet presented in 5. Because we want to use hashing function only on the first part (company symbol) of the Composite Key (company symbol, score). In the NaturalKeyPartitioner class, we have overridden the getPartition method, modified to apply the hashing function on the company symbol instead of the complete stock key. This provision of hashing only on the symbol from the composite key will ensure that all records from a company will go to the same reducer each time.

Listing 5: MR Job2 - Partition Method

```
// Overridden getPartition method
public int getPartition(StockKey key,
    DoubleWritable val, int numPartitions) {
      int hash = key.getSymbol().hashCode();
      int partition = hash % numPartitions;
      return partition;
 }
```

Reduce function: As shown in the code snippet 6, the Reduce function changes the records back to their original format (as they were before fed to the map function). The reducers emit a (key, value) pair where the key is the company symbol and the value is a composite on (score/rank, %times increases). With this new format, we can hold a ticker symbol, its position, and %times increase data in a single record.

Listing 6: MR Job2 - Reduce Function

```
// Core part of Reduce Function
// for each value of a key
for (DoubleWritable value : values) {
 // restructure the key and value to the
     original format  (same as before the
     mapping function)
 context.write(new Text(key.getSymbol()),
 new Text(key.getPercentageChange().toString()
 + " " + Double.toString(value.get())));
}
```

For the MR Job2, we use Hadoop to sort the records according to the value rather than a key.

3) MapReduce Job 3: The third MapReduce job is used for calculating probability distribution [18] of stock price movement. Using probability distribution, we can predict the %change in the stock price for different probabilities. For example, we can predict what will be the %change in the stock price for 0.6 probability. For calculating probability distribution in MR job3, the Map Function is the same as that of MR job1. However, the percentage change values must be sorted for a probability distribution. We use the same technique as in the MR job2, sorting according to values rather than keys. The Reduce function assigns values ranging from 0 to 1 to each percentage change value for a particular company. Usually, the output of each Reduce function is written into one single file. Still, we want each company's output to be written into different files for this task. This will reduce the overhead of searching a company's data into one massive file. Therefore, MapReduce job 3 writes each company's final output into separate files. For naming the output files, we use the company symbol and the partition number (CompanySymbol-r-00000), for example, AAL-r-00000.

IV. Service Architecture

Figures 4. and 5. depict our system's service architecture for the overall approach we used. Moreover, the data flow and

processing are well shown in the figure. 3, As mentioned in section III.A, that is, in the data collection and preprocessing section, we have collected stock market historical data from various sources using Python script and an MR job. The collected data is first kept in HBase and, after preprocessing, pushed onto HDFS (we chose HDFS because it is the default storage for Hadoop). Multiple MR jobs will access the historical data from HDFS, and results will be stored in the HDFS again. Then a javascript transfers the final output into DBMS to be queried by end users efficiently. The web application, developed using JSP and Java Servlets, uses the data stored in the DBMS. We have used HBase as the DBMS one more time in the end. A snapshot of the web application user interface page is shown in figure 7. The various component shown in the figures 4. and 5. are explained briefly as follows.

1) Data Sources: The scripts and the MR job we developed for collecting data are source agnostic. Therefore, we mainly used Yahoo Finance as a data source. However, with the paid subscription, all other sources were available.

2) Historical Data: This is the first staging point for the data we collected. The information here is in raw format and not yet preprocessed. We used HBase for storing initial raw data.

3) HDFS - for Storing Preprocessed Data: The data is then preprocessed to eliminate unnecessary information and presented in a well-organized format to be processed further. The preprocessed data is kept in HDFS - this is our second staging area for the data that is kept preprocessed but not yet analyzed.

4) The Main Processing - Mapper, Reducers: This is done using multiple Map Reduce jobs, which are run in a cascaded manner as the output of one Map Reduce job is fed to the next one and so on. At this stage, the actual processing is done on the data.

5) Results - DBMS: The output of all MapReduce jobs is in a huge file, making it hard for humans to read this data efficiently and quickly. Therefore, we again moved all the results to HBase so that a query engine could pull them down whenever needed.

6) User Interaction: Finally, a website is designed for users to take advantage of all the processing done so far. They can populate the filters and drop-down boxes to search for what they want.

V. Visualization

A. Design/Layout

Our overall layout includes three graphs: 1) Change in company's stock price, 2) Change in volume traded and 3) Probability for predicted percent change. Five companies were chosen from the NASDAQ top 100 list and displayed as selectable buttons with their data, analysis, and prediction values on the screen. In addition, there is an option to search a company's stock symbol (the first column from the figure 8 for viewing. The goal was to create a user-friendly, presentable page with all operations done in one window.

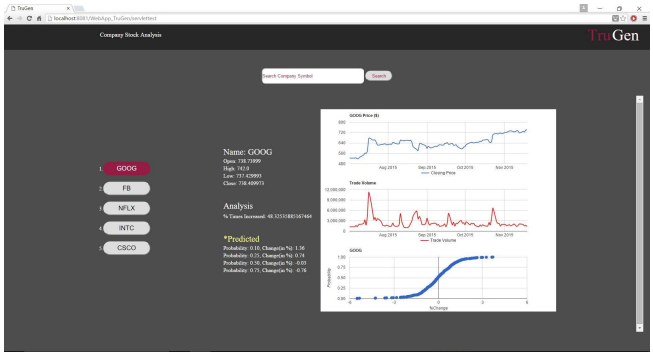


Fig. 7: User Interface Snapshot

B. Searching

We enable users to input desired stock symbols for viewing. For querying the Hbase table, we implemented the web app as a Java servlet. Requests would come to the Java side of the application; we would query the Hbase table and send results back to the HTML. For efficient searching of data required for Probability Distribution Function, each company's data is kept in a different file.

C. Graphing

The graphs were implemented using the Google Charts application programmer interface, designed for the web. This JavaScript-based API allowed us to feed our finalized data into a table and generate a configurable graph. Users can click on graph lines for more specific information, for example, prediction for a particular probability.

VI. Results and Evaluation

Figure 8 shows the output of the first two Cascaded MapReduce [19] jobs. Each row from the output represents a record (key, compositeValue) where Key=Company Symbol and CompositeValue is a combination of 1. score and 2. percentage time stock price increased. The output file contains companies' symbols in order of their rank. The score of that company decides the rank of a company. Moreover, if the rank of two or more companies is the same, the percentage time stock price increased value is considered. When deciding the rank of a company, the score has given more importance than just how many times the stock price went up. This is because the mere value of % times the stock price increased would not infer anything firmly; for example, if the stock price increases on every alternate day, it does not suggest anything - 50% On the other hand, the score is a better metric because it considers multiple factors. First, the score finds whether the stock price has consecutively increased or decreased. The Score Adjust Factors are tuned after processing each day's stock data for each company. It also considers the percentage value by which the stock price has increased or decreased.

The reduce function of the third MR job assigns values ranging from 0 to 1 to each %change value in increasing order for each company; this is used for creating the Probability Distribution of the percentage change. The aggregated output

```
CFBK.csv        7681577.0 85.60338743824983
CRAY.csv        7470877.0 75.87702010248324
UTSI.csv        7347991.0 92.76864728192163
ARCW.csv        7261338.0 50.31416143064283
NEWT.csv        7253488.0 92.91524534242981
MTSL.csv        7153375.0 80.3951041442989
AEZS.csv        7133643.0 86.00307062436029
GNVC.csv        7110437.0 83.448642895157
CYCC.csv        7056027.0 83.08115543328748
AGEN.csv        7046288.0 85.32225579053375
ACPW.csv        6979505.0 88.53354134165366
RDCM.csv        6932266.0 78.5073320201357
KTOS.csv        6912577.0 78.46878097125867
GSIG.csv        6823771.0 85.33619456366237
PICO.csv        6814668.0 64.83834646935821
```

Fig. 8: Final Output Of Cascaded MapReduce Jobs (MR1 and MR2)

Symbol	0.90	0.75	0.50	0.25	Actual %Change
AAL	+3.01	+0.51	-1.49	-0.6	+0.40
AAME	-8.79	-5.41	-5.41	-3.31	+1.00
AAOI	-3.74	-2.11	-0.14	+1.34	+2.01
AAVL	-4.06	-1.89	-0.78	+2.04	+1.01
AAXJ	-10.5	-9.06	-6.50	-3.92	0.00

TABLE I: Comparison of Predicted %Change with Actual %Change in the Stock Price for Some Probabilities

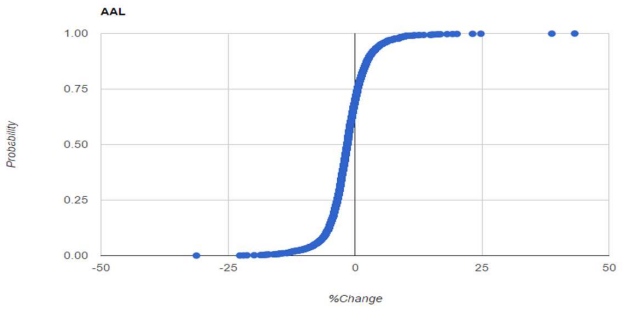


Fig. 9: Probability Distribution Function.
A few Predictions From Above Graph:
1) Probability: 0.10 Prediction: %Change: >+3.01
2) Probability: 0.25 Prediction: %Change: >+0.51
3) Probability: 0.50 Prediction: %Change: >-1.49
4) Probability: 0.75 Prediction: %Change: >-3.45

of all the reducers is written in a massive file per Hadoop implementation; however, here, we want each company output to be written into a different file as we do not want to add overhead for searching for a company in one huge file. So the third MR job is written to output each company's data in a separate file where each output file is named (CompanySymbol)-r-00000. Figure **??** depicts the output of the third MR job for a company. It is used for rendering Probability Distribution Graphs. The Probability Distribution Graph for a company is shown in figure 9. This graph can predict the next day's stock price with any probability. For example, according to figure 9, there is a 0.1 probability that the %change in the stock price of the company will be more than +3.01%, 0.25 probability that the %change in the stock price will be more than 0.51% and so on. We can predict the percentage change in the stock price for each probability using Probability Distribution Graph. Table 1 compares our prediction for different probabilities with actual movement in this stock price for five random companies from the input data. Furthermore, the table shows that the predictions are 100% correct for the probabilities greater than 0.25; however, for possibilities less than 0.25, the prediction is not always correct. We can observe from table 1 that predictions with greater probabilities are always accurate, but the prediction is not precise (the range is more fantastic). These statistics are verified for all the companies present in the input data.

For evaluating the results, initially, we suppressed all records from 2015 and only processed data till 31 December 2014 for many companies. After comparing our prediction with actual stock market movement, on average, our results were correct 100% of the time for greater probabilities than 0.30. We also continued the same strategy for predicting stock price change for other days, suppressing records after a specific date and checking if our prediction was correct. The correctness of our results also varied for different probabilities. The predictions for the same companies for less than 0.25 probability were not promising.

Furthermore, we have evaluated the companies according to their rank. The metric used is how often a company with the higher score is likely to be a better company to invest in. We found out that more than 60 percent of the time, a company with more score or higher rank is likely to be a better firm to invest in.

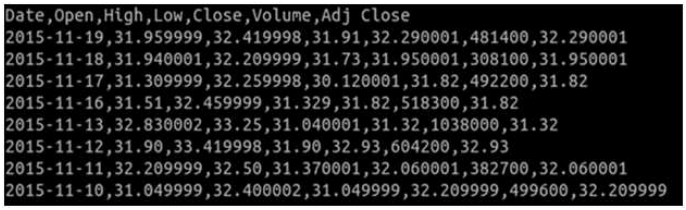
We can observe that the actual movement in the stock price is in the prediction range for the %change 0.5 and 0.75 probability, whereas, for 0.25 probability, prediction is approximately the same as that of actual movement. Moreover, we found the same pattern of predictions and actual stock price movement for other companies.

A. Challenges and Solutions

1) Multiple Connection by Hadoop Collector: Each process takes a ticker symbol list as input in a multi-process data collector and concurrently creates an HTTP connection to http://finance.yahoo.com. The connection requests come from the same IP range, making the server interpret that someone is staging a DDoS [20] attack on it. So, it simply rejects the connection requests. To solve this problem, we implemented a serial data collector and let a single process handles all the jobs. The method sends a new connection request after a previous connection is completed.

2) Input data format: Whether to keep a single large file or a file per company? Again, splitting data exactly on a company boundary would become problematic. This is tackled by keeping a separate file for each company.

3) Split Size for Data Partition: Even though a separate file is kept for each company, it could be split across many mappers depending upon a split size. We iterated through the input folder to decide on an optimal split size so that



```
Date,Open,High,Low,Close,Volume,Adj Close
2015-11-19,31.959999,32.419998,31.91,32.290001,481400,32.290001
2015-11-18,31.940001,32.209999,31.73,31.950001,308100,31.950001
2015-11-17,31.309999,32.259998,30.120001,31.82,492200,31.82
2015-11-16,31.51,32.459999,31.329,31.82,518300,31.82
2015-11-13,32.830002,33.25,31.040001,31.32,1038000,31.32
2015-11-12,31.90,33.419998,31.90,32.93,604200,32.93
2015-11-11,32.209999,32.50,31.370001,32.060001,382700,32.060001
2015-11-10,31.049999,32.400002,31.049999,32.209999,499600,32.209999
```

Fig. 10: Raw Data Format

a mapper will not have more than one company's data. First, iterating through the input folder was performed to determine the biggest file size. Once we get the maximum size, the split size is set to that value. This ensures that a file (a company's data) will go to only one mapper.

4) Sorting by values: In MR job three, which calculates probabilities distribution for stock price movement, we need to sort both keys and values. The details about how we achieved this are explained in section IV.

VII. Conclusion and future work

Although multiple serial algorithms are currently present for predicting stock market trends, we proposed a distributed algorithm. We were successfully able to bring it to realization by processing multi-million records. The challenges presented in this paper are expected to highlight problems before someone designs a distributed algorithm for the same problem or similar problems.

As our distributed algorithm implementation for predicting the stock market only performs statistical analysis [12] on stock data, its results will not be long-term enough if some other factors come into the picture. The non-statistical factors such as politics and social media affect the stock market significantly, and we plan on incorporating them as we move forward.

Our stock data analysis in a distributed fashion using Hadoop is simple and accurate enough to give an idea to the user about which companies they should focus on for future investment or at least monitoring.

We further suggest a few ideas for future work on top of this work. By nature, the stock market data is streaming data (real-time data). It would be great if we could peep into the partial output regularly rather than wait for all the Map Reduce jobs to be completed. The approach from this paper, combined with the early snapshot approaches from [6], would be a great thing to bring to fruition and see the results.

The other future idea, on top of our work, we thought of improvement in adjustments of up/down adjust-factors while calculating companies' rank or position with respect to each other. The code is publicly available online on the authors' GitHub, an open-source source control tool.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.

[2] R. Karve, D. Dahiphale, and A. Chhajer, "Optimizing cloud mapreduce for processing stream data using pipelining," in *2011 UKSim 5th European Symposium on Computer Modeling and Simulation*, 2011, pp. 344–349.

[3] E. Koo and G. Kim, "A hybrid prediction model integrating garch models with a distribution manipulation strategy based on lstm networks for stock market volatility," *IEEE Access*, vol. 10, pp. 34 743–34 754, 2022.

[4] "Hadoop." [Online]. Available: http://hadoop.apache.org/

[5] T. Harter, D. Borthakur, S. Dong, L. T. Amitanand Aiyer, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis of hdfs under hbase: A facebook messages case study." 12th USENIX Conference on File and Storage, 2014.

[6] D. Dahiphale, L. Huan, R. Karve, A. V. Vasilakos, Z. Yu, A. Chhajer, and C. Wang, "An advanced mapreduce: Cloud mapreduce, enhancements and applications," vol. 11. IEEE Transactions on Network and Service Management, 2014, pp. 101 – 115. [Online]. Available: https://doi.org/10.1109/TNSM.2014.031714.130407

[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in *NSDI*, 2010, pp. 313–328.

[8] D. D. Mishra, S. Pathan, and C. Murthy, "Apache spark based analytics of squid proxy logs." IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8710044

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in *NSDI*, 2010, pp. 313–328.

[10] H. Liu and D. Orban, "Cloud mapreduce: A mapreduce implementation on top of a cloud operating system," *IEEE International Symposium on Cluster Computing and the Grid*, vol. 0, pp. 464–474, 2011.

[11] K. Nygren, "Stock prediction - a neural network approach." Royal Institute of Technology, KTH, 2004.

[12] T. Wang and J. Wang, "The statistical analysis of chinese stock market fluctuations by the interacting particle systems," in *2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 3, 2008, pp. 42–46.

[13] L. Mathanprasad and M. Gunasekaran, "Analysing the trend of stock marketand evaluate the performance of market prediction using machine learning approach," in *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2022, pp. 1–9.

[14] K. T., A. K., Y. M., and T. M., "Stock market prediction system with modular neural networks." 12th USENIX Conference on File and Storage, p. 199–212.

[15] R. J. Kuo, C. H. Chen, and Y. C. Hwang, "An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network," 2001.

[16] R. Verma, P. Choure, and U. Singh, "Neural networks through stock market data prediction," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, 2017, pp. 514–519.

[17] H. Liang, J. Hogan, and Y. Xu, "Parallel user profiling based on folksonomy for large scaled recommender systems: An implimentation of cascading mapreduce," in *2010 IEEE International Conference on Data Mining Workshops*, 2010, pp. 154–161.

[18] R. Li, M. S. Kang, and G. Wang, "Evaluation of statistic probability distribution functions: Ii. chi-squared probability function and the inverse functions of z, t, f, and chi-squared distributions," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1253–1260.

[19] D. Dahiphale, *An Algorithm for Finding 2-Edge Connected Components in Undirected Graphs Using MapReduce*. University of Maryland, Baltimore County, 2017.

[20] M. A. Saleh and A. Abdul Manaf, "Optimal specifications for a protective framework against http-based dos and ddos attacks," in *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, 2014, pp. 263–267.