In proceedings of IEEE CLOUD Summit Conference 2023, July 2023

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

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

***Abstract*—The stock market generates massive data daily on top of a deluge of historical data. Investors and traders look to stock market data analysis for assurance in their investments, a prime indicator of our global economy. 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 the 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. Unfortunately, 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 constantly collecting data about everything around us, including humans, systems, processes, and organizations. This results in an ever-increasing volume, velocity, and variety of data. It must be processed efficiently and continuously to gain meaningful insights from this data. This is where the MapReduce [1] framework comes in. MapReduce is a programming model and an associated implementation for processing big data sets in a parallel and distributed fashion. We have briefly explained MapReduce in Section II-A.

Predicting stock market trends has long been a challenge. 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. Despite this, millions invest in the stock market each year, hoping to make a profit. Professional analysts use various methods to predict future trends, but success is not guaranteed.

Machine Learning [3] can also be used for finding patterns in large datasets for making predictions or recommendations [4]. Big Data [1] [5] offers the promise of building more intelligent decision-making systems. However, traditional Machine Learning algorithms are not well-suited for all big data problems due to their high computational complexity. Distributed processing algorithms such as MapReduce can be used to overcome this challenge.

As we enter into a new era of Big Data and Big Data processing frameworks, the ways we process stock market data and make predictions [6] 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 [7] 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 [8], HBase [9], MapReduce).

In this paper, we collected historical data on companies from the NYSE and NASDAQ stock exchanges. We used cascading [10] MapReduce strategy with a distributed statistical analysis of stock market data using MapReduce based on cumulative distribution function (CDF) [11] to analyze the data. Furthermore, we display the results in a web application. The web application shows price, volume, and prediction graphs. Anyone can use the web application to search for a company and determine whether or not to invest in the company with some probability.

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) [12], [13], [14], [15] is a widely used MapReduce implementation. It is a modification of the traditional Hadoop implementation, which runs the Map and Reduce phases sequentially. In HOP, the Map and Reduce phases run simultaneously, which makes the overall system more efficient. In addition, HOP is well-proven, popular, and efficient in processing large datasets, such as stock market datasets. Therefore, we decided to use HOP for processing stock market datasets in our study.

Hadoop uses HDFS for data storage in the backend. HDFS [8] is a distributed file system that provides reliable and scalable data storage. It is designed explicitly for spanning large clusters on the commodity hardware.

B. Prediction Analytics

Currently, there are many approaches to stock market prediction [16], either using different processing or data sources. Approaches to processing stock market data can be categorized into two groups. The first group emphasizes statistical analysis [17] of a company's stock price movements from the past. The second group focuses on machine learning techniques.

To improve the accuracy of stock price predictions, some algorithms currently in use also consider social, political, economic, and environmental factors. For example, Karl et al. [16], R. J. Kuo et al. [18], and Kimoto T. et al. [19] independently used neural network approaches in their research. Rohit Verma's [20] work also uses a neural network approach for stock market prediction. He uses a standard Error Correction Neural Network (ECNN) [20] [18] for daily forecasting, and an extension of ECNN for weekly predictions.

Our predictions use statistical data with Score Adjust Factors (as shown in Figure 5, which vary according to daily stock price movements. For long-term investments, we calculate the rank of each company. The rank determines how likely a particular company is to perform compared to other companies in the given data. In addition to calculating the positions of companies, statistical analysis was also used to predict the next day's stock price for each company, with probabilities ranging from 0 to 1.

C. 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.

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

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

$$X = real\ value\ variable$$

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

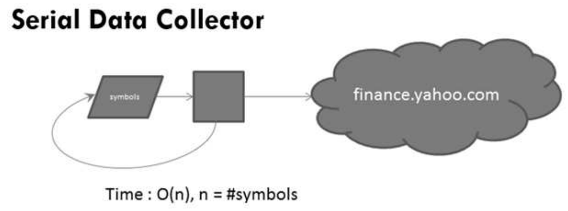


Fig. 1: Serial Data Collector

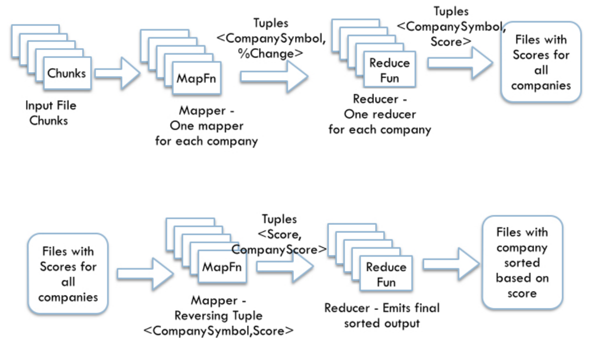

Chunks
Input File Chunks
MapFn
Mapper - One mapper for each company
Tuples <CompanySymbol, %Change>
Reduce Fun
Reducer - One reducer for each company
Tuples <CompanySymbol, Score>
Files with Scores for all companies
Files with Scores for all companies
MapFn
Mapper - Reversing Tuple <CompanySymbol,Score>
Tuples <Score, CompanyScore>
Reduce Fun
Reducer - Emits final sorted output
Files with company sorted based on score

Fig. 2: Data Flow and Algorithm Processing

III. Our Contribution: Prepossessing and Algorithm

A. Data Collection and Preparation

As 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 price in a day, lowest price in a day, closing price, the volume traded, and adjusted volume for each stock symbol.

First, we collected all current ticker symbols from the 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 (as shown in Figure 1) versions of the data collector if we only needed sample data for one year. In other cases, we limited the 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 symbol.

We have collected historical data for approximately 3,000 NASDAQ and 1,450 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 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 a safer approach). We also developed a CSV file generator application that combines all raw data files into one 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 the data into CSV format, the files are uploaded 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).

```
1:  procedure PREDICTOR($ticker, close\_prices$)
2:      $number\_of\_days \leftarrow len(close\_prices)$
3:      $ptg\_changes \leftarrow [\ ]$
4:
5:      for <$d$ in $range(1,\ number\_of\_days)$> do
6:          $today\_price \leftarrow close\_prices[d]$
7:          $yesterday\_price \leftarrow close\_prices[d-1]$
8:          $price\_change \leftarrow today\_price - yesterday\_price$
9:          $ptg\_change \leftarrow ((price\_change \div today\_price) * 100)$
10:         $ptg\_changes.append(ptg\_change)$
11:         $sort\ ptg\_changes\ values\ descending\ order$
12:     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 the Cumulative Distribution Function (CDF) described in the background survey. The serial algorithm 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 useless for taking real-time action. Therefore, we further 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 jobs working towards achieving one goal. The algorithm 2 queries the data represented using the graph in Figure 9.

C. Distributed Statistical Analysis

The data analysis is done using three different MapReduce jobs (including data collection and preprocessing, we developed five MapReduce jobs; however, we are focusing on three primary data analysis MapReduce jobs). The three MapReduce jobs are:

1) Calculate the percent change for each company for each day.
2) Sort the output generated by the first MapReduce job according to calculated scores for each company.
3) Calculate the probability distribution of stock price movement.

All the jobs are run in a cascaded fashion. Cascaded MapReduce 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. Figure 3 presents the overall approach and data flow.

Algorithm 2 Predicting Stock Price Movement Statistically Query Function.

```
1:  procedure QUERY($price, ptg\_changes$)
2:      $days \leftarrow len(ptg\_changes)$
3:      $split\_index \leftarrow nearest\ location\ in\ ptg\_changes$
4:      $probability \leftarrow (split\_index \div days) \times 100$
5:
6:      $return\ probability$    ▷ Probability that price will be greater or equal to price.
```

1) MapReduce Job 1: The first MapReduce job calculates the scores and percentages of stock price increases for each company. These two indicators are used to rank the companies. The score is an integer value calculated for each company using the price fluctuations and $adjust\ factor$ (which is dynamically adjusted) for that company. The company with the highest rank is likely to be the best investment because it has shown the greatest growth potential.

Map Function:

The first MapReduce job takes as input data that has been prepared and stored on the Hadoop Distributed File System (HDFS). The data is stored in a directory containing multiple files, each representing a company's historical stock trading data for each day. The percentage change in stock price for each day is used to calculate the score value for each company. 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 Listing 1. Each thread in the MapReduce job will apply the Map function to its assigned chunk of data. The Map function calculates the %change in the stock price for each day for a company, represented by a ticker symbol. The output of the MapReduce job is a daily record in the key-value pair format, where the key is the company ticker and the value is the %change in price for that company on that day.

Listing 1: MapReduce 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 Listing 2. It aggregates the records from all the mapper threads and calculates a score for each company. The score is calculated using three values:

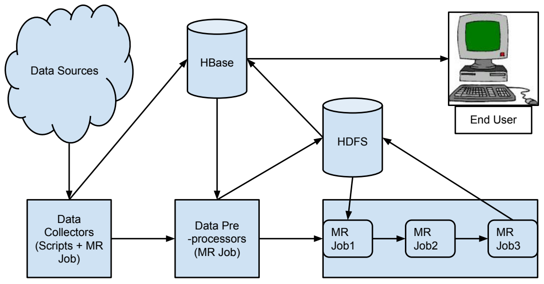
1) Up Adjust Factor: It is used to increase the score for companies with a positive %change in stock price.
2) Down Adjust Factor: It is used to decrease the score for companies with a negative %change in stock price.
3) %Change Value: It is the %change in stock price for the company for each day.

The score-adjusting factors are tuned according to stock price movements, daily. Figure 5 shows how the score-adjusting factors are changed based on feedback on stock price movement. When the stock price increases, the Up Adjust Factor increases, and the Down Adjust Factor decreases. When the stock price decreases, the Down Adjust Factor increases, and the Up Adjust Factor decreases. The score for a company is adjusted linearly, taking into account the Up and Down Adjust Factors, and the %Change in the stock price.

Listing 2: MapReduce 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
}
```

2) *MapReduce Job 2:* The second MapReduce job sorts the output of the first MapReduce job by company score. The output of this job is a list of companies ranked from highest to lowest score. The website/user interface (please see Figure 6) displays the top five companies on the home page. Users can also search for other companies by entering a company name or ticker symbol in the search bar.



Data Sources
HBase
End User
HDFS
Data Collectors (Scripts + MR Job)
Data Pre -processors (MR Job)
MR Job1
MR Job2
MR Job3

Fig. 3: Data Flow

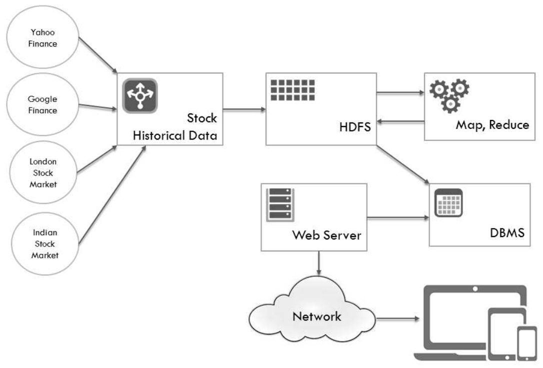


Fig. 4: Service Architecture

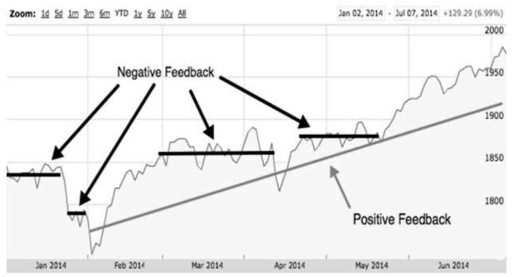


Fig. 5: Adjust Factors. 1) Up 2) Down 3) %Change

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 [16]) using the company's name and score, whereas the value is a percentage number that the stock price increased.

Listing 3: MapReduce 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 Listing 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. Therefore, 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 is given as input 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 Listing in 5 for reference. 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 entire stock key. This provision of hashing only on the symbol from the composite key will ensure that all records for a company will go to the same reducer each time for aggregation.

Listing 5: MapReduce 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 Listing 6, the Reduce function changes the records back to their original format (as they were before fed to the map function). The reducers emit key-value pairs where the key is the company symbol and the value is a composite on (score/rank, %times increases). With this new format, we can incorporate a ticker symbol, its position, and %times increase data in a single record.

Listing 6: MapReduce Job2 - Reduce Function

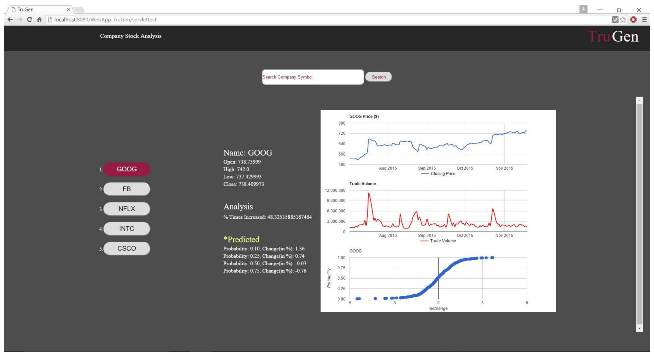
```
// Core part of Reduce Function
// for each value of a key
for (DoubleWritable value : values) {
 // restructure the key and value to the
     original format  (same as before the
     mapping function)
 context.write(new Text(key.getSymbol()),
 new Text(key.getPercentageChange().toString()
 + "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

3) MapReduce Job 3: The third MapReduce job is used for calculating probability distribution [21] of stock price movement. Using probability distribution, we can predict the %change in the stock price with 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 MapReduce job3, the Map Function is the same as that of MapReduce job1. However, the percentage change values must be sorted for a probability distribution. We use the same technique as in the MapReduce 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 3 and 4 depict our system's service architecture for the overall approach we used. The data flow and processing are also shown in Figure 2. As mentioned in Section III-A, we collected stock market historical data from various sources using a Python script and a MapReduce job. The collected data was first stored in HBase, and after preprocessing, it was pushed onto Hadoop Distributed File System (HDFS). We chose HDFS because it is the default storage for Hadoop. Multiple MapReduce jobs accessed the historical data from HDFS, and the results were stored in HDFS again. Then, a JavaScript program transferred the final output into a database management system (DBMS) to be queried efficiently by end users. The web application, developed using JavaServer Pages (JSP) and Java servlets, used the data stored in the DBMS. We used HBase as the DBMS one more time in the end. A snapshot of the web application user interface page is shown in Figure 6. The various components shown in Figures 3 and 4 are explained briefly as follows:

1) Data Sources: The scripts and the MapReduce job we developed for collecting data are source agnostic. Therefore,



Name: GOOG
Open: 738.73999
High: 742.0
Low: 737.429993
Close: 738.409973

Analysis
% Times Increased: 48.32535885167464

*Predicted
Probability: 0.10, Change(in %): 1.36
Probability: 0.25, Change(in %): 0.74
Probability: 0.50, Change(in %): -0.03
Probability: 0.75, Change(in %): -0.76

Fig. 6: User Interface Snapshot

we mainly used Yahoo Finance as a data source. However, all other sources were available with a paid subscription.

2) Historical Data: This is the first staging point for the data we collected. The information here is in raw format and has not yet been preprocessed. We used HBase to store the 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 for further processing. The preprocessed data is stored in HDFS, which is our second staging area for data that is preprocessed but not yet analyzed.

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

5) Results - DBMS: The output of all MapReduce jobs is stored in a huge file, which makes it difficult for humans to read the data efficiently and quickly. Therefore, we 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.

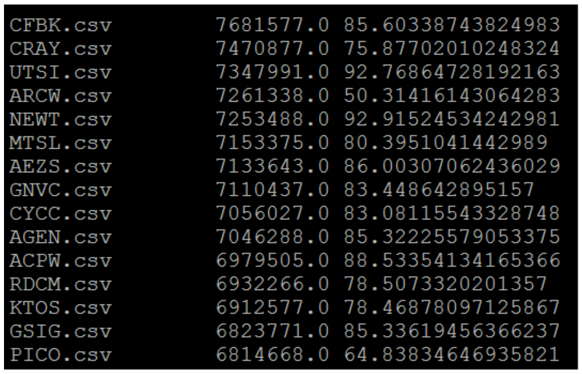
V. Visualization

A. Design/Layout

Our overall layout includes three graphs: 1. Change in the company's stock price, 2. Change in volume traded, 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. Additionally, there is an option to search for a company's stock symbol (the first column from Figure 8) for viewing. The goal was to create a user-friendly, presentable page with all operations done in one window.

B. Searching

We enable users to input desired stock symbols for viewing. We implemented the web app as a Java servlet to query the HBase table. Requests would come to the Java side of the application, and we would query the HBase table and send results back to the HTML. Each company's data is kept in a different file for efficient searching of data required for the probability distribution function.



```
CFBK.csv        7681577.0 85.60338743824983
CRAY.csv        7470877.0 75.87702010248324
UTSI.csv        7347991.0 92.76864728192163
ARCW.csv        7261338.0 50.3141614306428З
NEWT.csv        7253488.0 92.91524534242981
MTSL.csv        7153375.0 80.3951041442989
AEZS.csv        7133643.0 86.00307062436029
GNVC.csv        7110437.0 83.44864289515 7
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. 7: Output Of Cascaded MapReduce Jobs (MapReduce1&2)

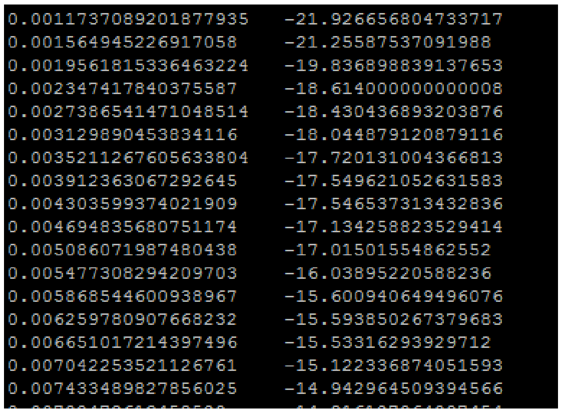
C. Graphing

The graphs were implemented using the Google charts application programming interface (API), designed for the web. This JavaScript-based API allowed us to feed our finalized data into a table and generate a configurable graph. In addition, users can click on graph lines for more specific information, such as predicting a particular probability.

VI. Results and Evaluation

Figure 7 shows the output of the first two Cascaded MapReduce [22] jobs. Each row from the output represents a record (key, compositeValue) where Key=Company Symbol and CompositeValue is a combination of score and percentage time the stock price increased. The output file contains companies' symbols sorted by their rank/score. Moreover, if the rank of two or more companies is the same, the percentage time stock price increased value is a tie-breaker. 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% Additionally, 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 changed.

The reduce function of the third MapReduce 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 of all the reducers is written in a massive file per Hadoop implementation; however, here, we want each company's output to be written into a different file to make



```
0.0011737089201877935    -21.926656804733717
0.001564945226917058     -21.25587537091988
0.0019561815336463224    -19.836898839137653
0.002347417840375587     -18.614000000000008
0.0027386541471048514    -18.430436893203876
0.003129890453834116     -18.044879120879116
0.0035211267605633804    -17.720131004366813
0.003912363067292645     -17.549621052631583
0.004303599374021909     -17.546537313432836
0.004694835680751174     -17.134258823529414
0.005086071987480438     -17.01501554862552
0.005477308294209703     -16.03895220588236
0.005868544600938967     -15.600940649496076
0.006259780907668232     -15.593850267379683
0.006651017214397496     -15.53316293929712
0.007042253521126761     -15.122336874051593
0.007433489827856025     -14.942964509394566
```

Fig. 8: Raw Data for CDF - Output of MapReduce job 3

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

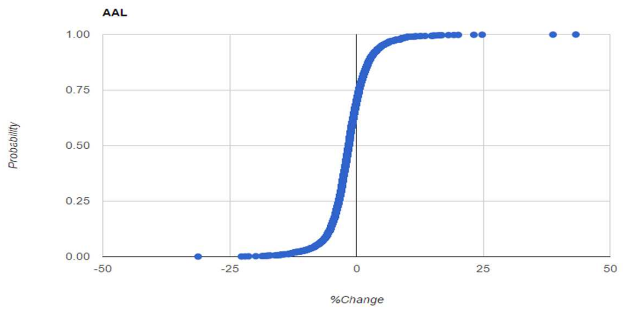
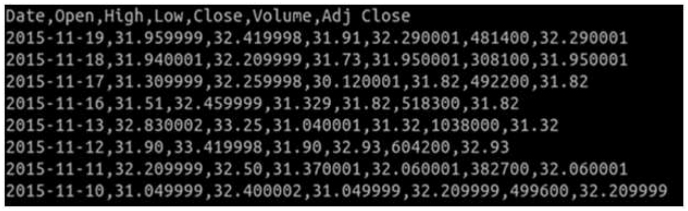


Fig. 9: Probability Distribution Function.
A few Predictions From Above Graph:
1) Probability: 0.10 Prediction: %Change: >+3.01
2) Probability: 0.25 Prediction: %Change: >+0.51
3) Probability: 0.50 Prediction: %Change: >-1.49
4) Probability: 0.75 Prediction: %Change: >-3.45

searching fast and efficient. So the third MapReduce job is written to output each company's data in a separate file where each output file is named (CompanySymbol)-r-00000. Figure 8 depicts the output of the third MapReduce 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 I



```
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

compares our prediction for different probabilities with actual movement in this stock price for five random companies from the input data. Furthermore, Table I shows that the predictions are 100% correct for the probabilities greater than 0.25 (the column 0.75); however, for possibilities less than 0.25, the prediction is not always correct (in the selected test data, the accuracy we found is 80%, however, with change in the test data we expect the accuracy to go down). We can observe from Table I 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% 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: In a multi-process data collector, each process takes a ticker symbol list as input and concurrently creates an HTTP connection to http://finance.yahoo.com. However, the connection requests come from the same IP range, which makes the server think that someone is trying to perform a denial-of-service (DDoS) [23] attack. As a result, the server rejects the connection requests. The solution to this issue is explained in Section III-A.

2) Input data format: Whether to keep a single large file or a file per company. Splitting data exactly on a company boundary would become problematic. This issue 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 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 MapReduce job 3, which calculates the probability distribution of stock price movement, we need to sort the keys and values. The details about how we achieved this are explained in Section III-C3.

VII. Conclusion and future work

We propose a distributed algorithm for stock market prediction. It can process multi-million records and highlights challenges for future designs. Our algorithm only performs statistical analysis on stock data, so its results may not be long-term enough. We plan to incorporate non-statistical factors like politics and social media (sentiment analysis [24]). The system is simple and accurate enough to suggest companies for future investments or monitoring.

There are many ways this work can be extended. First, the stock market data is continuous; we could peek into partial output regularly instead of waiting for all MapReduce jobs to complete. Combining this approach with the early snapshot approaches from [10] would significantly improve results.

Second, the final ranks of companies can be reshuffled by incorporating the sentiment analysis of tickers.

Third, it would be an exciting thing to design and develop a neural network [25], [26] based stock prediction system using the same dataset and then compare the results. The authors are working on this idea as a follow-up work.

Additionally, improvements in up and down score-adjust-factors can be made while calculating companies' rank. The implemented code is publicly available online on the authors' GitHub repository.

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] T. M. Mitchell *et al.*, *Machine learning*. McGraw-hill New York, 2007, vol. 1.

[4] D. Dahiphale, P. Shinde, K. Patil, and V. Dahiphale, "Smart farming: Crop recommendation using machine learning with challenges and future ideas," 6 2023. [Online]. Available: https://doi.org/10.36227/techrxiv.23504496.v1

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[6] 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.

[7] M. Sharma, V. D. Sharma, and M. M. Bundele, "Performance analysis of rdbms and no sql databases: Postgresql, mongodb and neo4j," in *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2018, pp. 1–5.

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

[9] 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.

[10] 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

[11] I. W. Burr, "Cumulative frequency functions," *The Annals of Mathematical Statistics*, vol. 13, no. 2, pp. 215–232, 1942. [Online]. Available: http://www.jstor.org/stable/2235756

[12] e. a. Tyson Condie, "MapReduce online," in *NSDI*, 2010, pp. 313–328.

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

[14] 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.

[15] D. Dahiphale, "Mapreduce for graphs processing: New big data algorithm for 2-edge connected components and future ideas," *IEEE Access*, vol. 11, pp. 54 986–55 001, 2023.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] D. Dahiphale, "Mapreduce for graphs processing: New big data algorithm for 2-edge connected components and future ideas," *IEEE Access*, pp. 1–1, 2023.

[23] 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.

[24] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447914000550

[25] K. Gurney, *An introduction to neural networks*. CRC press, 1997.

[26] A. Moghar and M. Hamiche, "Stock market prediction using lstm recurrent neural network," *Procedia Computer Science*, vol. 170, pp. 1168–1173, 2020, the 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920304865