Stock Market Data Analysis and Predictions using Hadoop

Devendra Dahiphale*, Abhijeet Wadkar†, Bernie McNamee‡, Benjamin Latt§, Dr. Karuna Joshi¶
University of Maryland, Baltimore County,
Email:{devdahiphale, abhi6†, bmcnam1‡, blatt1§, kjoshi1¶}@umbc.edu*

***Abstract*—Stock markets are wells of complex data that determine financial decisions made globally. Because so much of our worth is defined by the Market, more people now than ever look for some assurance as to the future value of their investments. Consequently, much research has been done on prediction algorithms within the stock market, however, due to slow electronic trading systems and order processing time, there is a need for gaining speed/control with the number of transactional records being logged. Using industry-standard cloud technologies and concepts for big-data manipulation, such as Hadoop and Hbase, we overcome these challenges and implement a prediction algorithm to forecast a stock's "next-day" price.**

***Index Terms*—Big Data, MapReduce, Hadoop, Stock Market Analysis.**

I. Introduction

THe idea of predicting the stock market has tantalized people for years. Hundreds of millions of people every year invest in stock exchanges all over the world with hopes of making it big. However, because company values fluctuate and change, investing is a calculated risk- deterring many from investing. For as long as the stock markets have been around, professional analysts have tried to predict future trends in the value of a company. In the past, these predictions were often made by correlating foreign economies? transactions as well as looking at current events - these methods have since changed. As we enter into a new era of Big-Data, not only have the ways we make predictions changed, the way we process stock data has also begun changing. Now social media feeds have begun outlining future trends in companies? values, and stock exchanges are beginning to outsource their traditional RDBMs to big-data providers for faster data manipulation. While social media prediction analysis is beyond the current scope of this study, we did make use of cloud technologies (Hadoop, HBase, MapReduce) and implement a prediction algorithm to forecast how a company?s value will change in a day. We began by organizing company transactional data from both the NYSE and the NASDQ. To perform the analysis, we implemented cascading MapReduce algorithms for calculating the percentage changed probability distribution. With the final analysis, we display the results in a web app, showing graphs for price, volume and prediction. Our end result is an interface allowing anyone to search for a company and determine whether or not they should invest. The following is an in-depth look at the technologies and algorithms used in addition to the process yielding our end results. As our contribution, we have developed a simple but an effective algorithm for stock market data analysis and prediction of future stock movement. Further, we have developed a web application which could be used by any user (a person or other firm).

The paper is organized as follows: In section II, we begin with a literature survey of technologies and services related to our work on stock market analysis and prediction. Section III describes prototype and further details about implementation

Section IV concentrates on visualization of results and also talks about the web application which is developed for the end user. Section V elaborates on challenges faced during design and implementation, and solutions to these. Section VI gives an evaluation of experimental results. Section VII presents assumptions and limitation of our work .... Finally, we conclude in the section VIII by presenting future score or enhancements to our work.

II. Literature Survey

A. MapReduce

MapReduce [1] is a programming model developed by Google for processing big data sets in a distributed manner. It operates in two phases: a Map phase, in which input data is split into multiple chunks/splits and for processing each chunk/split a mapper is spawned which applies a user defined function for processing, called the Map function; and a Reduce phase, which aggregates the data produced in the Map phase.

B. Online MapReduce (Hadoop Online Prototype)

There are many MapReduce implementations [3], for example Hadoop [2], Spark, Disco, MapReduce-MPI, MARIANE, MARISSA and Phoenix. The most common and widely used is Hadoop. HOP [2] is a modification to the traditional hadoop. In traditional Hadoop Map and Reduce phases run sequentially - first Map phase finishes and then Reduce phase starts. However in HOP, Map and Reduce phases run simultaneously. As HOP is well proven, most popular and efficient in processing Big Data sets, we decided to use it for processing stock market data.

C. Hadoop Distributed File System

It is a distributed file system which provides reliable and scalable data storage. It is specifically designed for spanning large clusters on the commodity hardware. Hadoop uses HDFS for data storage which is to be processed.

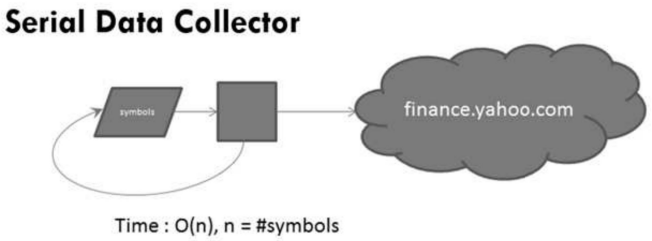

Fig. 1: Serial Data Collector

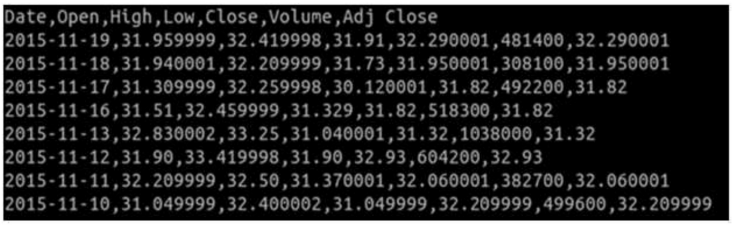

Fig. 2: Raw Data Format

D. Something related to stock market

.....need to edit this section.....

III. Project Implementation and Prototype

A. Data Collection and Preparation

Before developing plan for the project, we identify what data sets are are available for our project. Most data source required subscriptions or fees but the historical data sets on finance.yahoo.com is free and open to the public. The historical data set contains date, opening price, highest in the day, lowest in the day, closing price, volume traded, and adjusted volume for each stock symbol. First we made some research and collect all current ticker symbols from NYSE and NASDAQ stock exchanges. Then we developed a multi-process file collector using hadoop framework. In this multi-process approach many hadoop mapper processes use ticker symbol list as input and collect historical data from http://finance.yahoo.com simultaneously. We have found a major design flaw in this approach. When mappers made the multiple http requests to the server, the server thought it was an attack to the server and rejected the http requests. We replaced multi-process data collector with serial version of data collector. In this serial data collector, all jobs are done in a single process. We have successfully collected historical data of 5000 NASDAQ ticker symbols and 1450 NYSE ticker symbols. The original data set is in Comma Separated Value file format.

In the next step, we developed a file parser which consumes the data collected from the website and stores it to the hbase table. The unique key is the combination of exchange symbol, ticker symbol and the date. We also developed a CSV file generator which combines all raw data files into a one single CSV file with corresponding exchange symbol and ticker symbol prefix to each record. We could generate a unique from the first three columns: exchange symbol, ticker symbol, and the record date. After preparing data into CSV format, the file is push to the hadoop file system (HDFS) for analysis step. We have collected approximately 15 million records for this project.

B. Analysis

The data analysis is done using three different MapReduce Jobs. 1. The first MR job is for finding out companies score and percentage times the stock went up. These two score are important for deciding company rank. The best score indicates the best company to invest in. Map Function: A directory on the HDFS is given as an input to this MR job. The directory contains multiple files - a file for a company. Percentage change on each day in stock price is used for calculating score. For record R, opening price OP and closing price CP, the percentage change (PC) is calculated by using PC = (CP - OP) / OP) * 100. So, Map function calculates percentage change in the stock price for each day for a company. Output is a record for a day in the format of ¡key, Value¿ pair where key is company symbol and value is a double percentage change.

```
public void map(Object key, Text value,
    Context context
                ) throws IOException,
                    InterruptedException {

// Get the company name and use it as a key
     FileSplit fileSplit =
         (FileSplit)context.getInputSplit();
     String filename =
         fileSplit.getPath().getName();
     String[] result =
         value.toString().split("\n");
// Get all rows in a list and reverse the list
     List<String> list = new
         ArrayList<String>();
     for(String line : result) {
       list.add(line);
     }
     Collections.reverse(list);
     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: Records from mappers with a key-value pair are processed by reducers. Reducer function calculates score for each company. For calculating score, three score adjust factor are used, up adjust factor, down adjust factor and percentage change value. The adjust factors are tuned according stock movement.

```
public static class IntScoreReducer extends
```

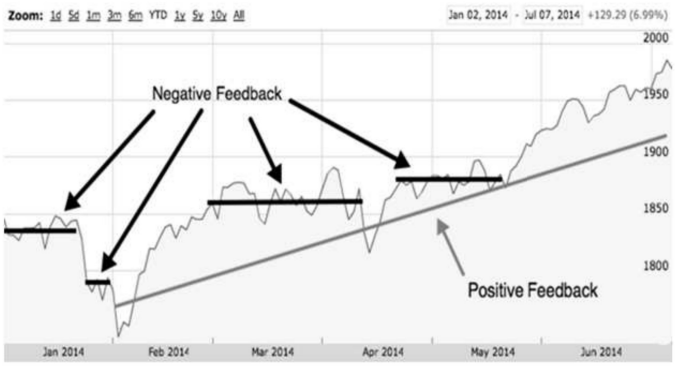

Fig. 3: Adjust Factors. 1) Up Adjust Factor 2) Down Adjust Factor 3) Percentage Change Value

```
 Reducer<Text,DoubleWritable,Text,Text> {
private DoubleWritable result = new
    DoubleWritable();
public void reduce(Text key,
    Iterable<DoubleWritable> values,
    Context context) throws IOException,
    InterruptedException {
  int score = 0;
  int total_records = 0;
  double times_increased = 0;
  int up_adjust_factor = 1;
  int down_adjust_factor = 1;
  // calculating score for each company
  for (DoubleWritable val : values) {
    // calculate total number of records
    total_records++;
    if(val.get() > 0) {
          score += up_adjust_factor +
              val.get();
          times_increased++;
          up_adjust_factor++;
          if(down_adjust_factor > 0)
                down_adjust_factor--;
    } else if (val.get() < 0) {
          score -= down_adjust_factor +
              val.get();
          down_adjust_factor++;
          if(up_adjust_factor > 0)
                up_adjust_factor--;
    }
  }
  double percentage_times_increased =
      (times_increased / total_records) *
      100;
  result.set(score);
  // Here, Key=company symbol and
      Value=<score, %times_increased>
  context.write(new Text(key), new Text(new
      Double(score).toString()+ " " + new
      Double(percentage_times_increased).toString()));
  }
}
```

2. The second MR job is just for sorting the output generated by MR job1 according to scores. Map function: Takes output of the MR job1. For each record it swaps key and value and emit an intermediate record.

Reduce function: Again it changes the records in the original format by swapping key and value. In a nutshell, we are using using Hadoop to sort the records according to the value rather than key. The third MR job is for calculating probability distribution of stock price movement. The map function is same as that of MR job1 calculating percentage change in stock prices. For probability distribution the percentage change values has to be sorted. The obvious solution would be to sort the values when it reaches reduce function. But in hadoop implementation of MR, there is a sorting in intermediate staging area between map and reduce function. We are leveraging these fact to sort the key-value pairs emitted by map function on not only key but value also. In our implementation the composite key is ¡CompanySymbol, PercentageChange¿. Sorting is first done on key that is CompanySymbol, if key is same then PercentageChange is used for sorting. So when these key-value pairs reaches reduce function they are already sorted on CompanySymbol as well as PercentageChange. To achieve this function we had to override Hadoop?s in-build functions - compare function by implementing CompositeKeyComparator, NaturalKeyGroupingComparator and NaturalKeyPartitioner. Reduce function assigns value ranging from 0 to 1, to each percentage change value for perticular company symbol. The output of each reduce function is written into one single file, but here we want each company output to be written into different file as we do not want add overhead for searching values of each company into one huge file. So MR job is written such that it outputs these values into separate file, one for each company named ¡CompanySymbol-r-00000¿.

```
// core part of the Map function
for(String line : result) {
  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;
  // generating stock key for sorting purpose
  context.write(new StockKey(filename,
      percentage_change), new
      DoubleWritable(percentage_change));
}
```

```
// a new composite key class
class StockKey {
      private String symbol;
      private double timestamp;
      public StockKey(String symbol, double
          timestamp) {
            this.symbol = symbol;
            this.timestamp = timestamp;
      }

      public String getSymbol() {

            return symbol;
      }
```

```
      public void setSymbol(String symbol) {
            this.symbol = symbol;
      }

      public double getTimestamp() {
            return timestamp;
      }

      public double setTimestamp(double
          timestamp) {
            this.timestamp = timestamp;
      }
}
```

```
// Composite Key Comparator
class CompositeKeyComparator extends
    WritableComparator {
   protected CompositeKeyComparator() {
      super(StockKey.class, true);
   }
   @SuppressWarnings("rawtypes")
   @Override
   public int compare(WritableComparable w1,
       WritableComparable w2) {
      StockKey k1 = (StockKey)w1;
      StockKey k2 = (StockKey)w2;

      int result =
          k1.getSymbol().compareTo(k2.getSymbol());
      if(0 == result) {
         result = (k1.getTimestamp() ==
             k2.getTimestamp())?1:0;
      }
      return result;
   }
}
```

```
// key grouping comparator
class NaturalKeyGroupingComparator extends
    WritableComparator {
   protected NaturalKeyGroupingComparator() {
      super(StockKey.class, true);
   }
   @SuppressWarnings("rawtypes")
   @Override
   public int compare(WritableComparable w1,
       WritableComparable w2) {
      StockKey k1 = (StockKey)w1;
      StockKey k2 = (StockKey)w2;

            // compare symbols of the
               companies
      return
          k1.getSymbol().compareTo(k2.getSymbol());
   }
}
```

```
// overridden the hashing function as well
class NaturalKeyPartitioner extends
    Partitioner<StockKey, DoubleWritable> {

   @Override

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

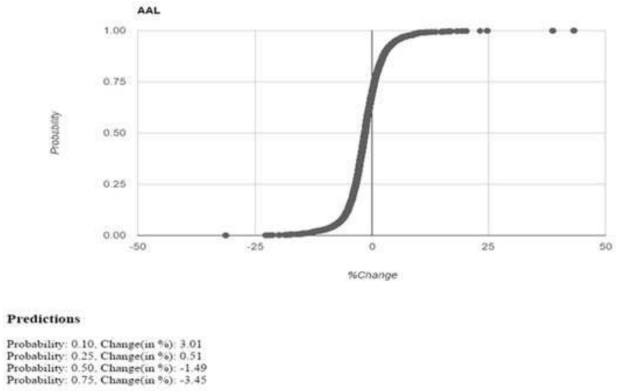

Fig. 4: Probability Distribution Graph

```
// Reducer just separates key back into
    symbol and value
// now key is again symbol for the company
   rather than StockKey
context.write(new Text(key.getSymbol()),
    val.get());
```

C. Architecture

Fig [illegible] depicts the architecture of system build for testing approaches that we have articulated. As mentioned in XX data collection section, we have collected stock market historical data from various sources using python script. The collected data is then pushed on to the HDFS. Multiple MR jobs will access the historical data from HDFS and results will be stored into the HDFS. Then a java script transfers the MR job results into DBMS, which intern will be used by web application to show analytics to user.

```
do {
  // dequeue one reduce key , generate an
      iterator of values
  for (Message msg : masterReduceQueue) {
    String bucket = msg.getBody();
    reduceWorkers.push(new
        ReduceRunnable(jobID, bucket, reduce,
        reduceCollector,
        msg.getReceiptHandle()));
    // only generate work when have capacity,
        could sleep here
    reduceWorkers.waitForEmpty();
  }
} while ( !dbManager.isStageFinished(jobID,
    Global.STAGE.REDUCE));
reduceWorkers.waitForFinish();
```

```
while(!Global.endCurrentJob) {
  ArrayList<S3Item> fileListToBeProcessed =
      new ArrayList<S3Item>();
```

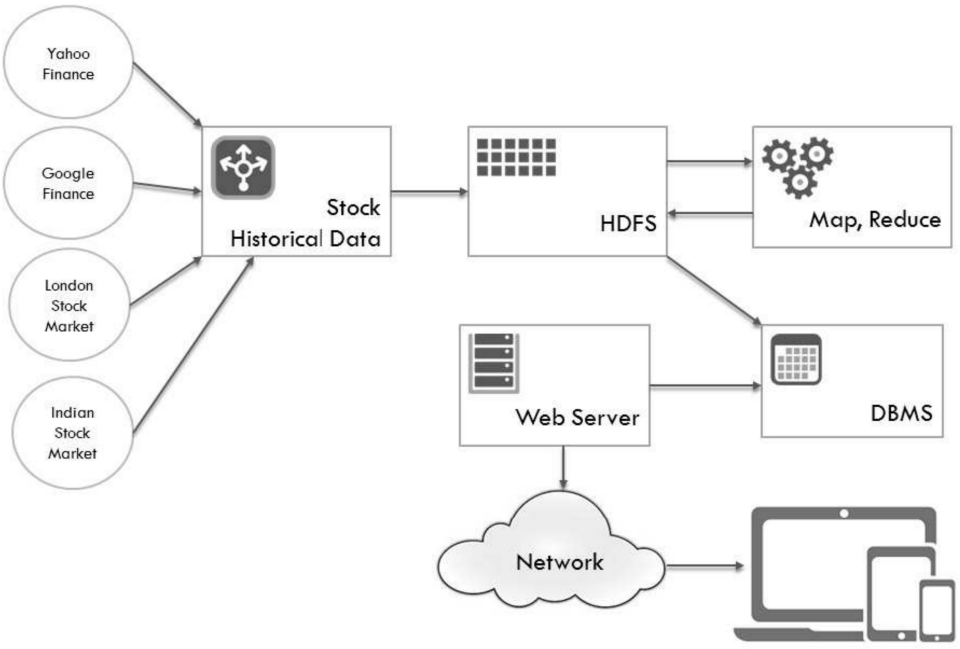

Fig. 5: Architecture

```
  // check for new files to process
  if(addDirToList(s3FileSystem.getItem(s3Path),
      fileListToBeProcessed)){
    // total number of mappers launched so far
    Global.numSplit += numSplit;
    // add MapId, pointer to splits in input
        queue
    CreateAndAddSplitsToInputQ(fileListToBeProcessed);
  }
}
```

IV. Visualization

A. Design/Layout

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

B. Searching

We enable users to input desired stock symbols for viewing. For querying the Hbase table, we implemented the webApp as a Javaservlett. Requests would come to the java side of the application, we would query the Hbase table, and send results back to the html.

C. Graphing

The graphs were implemented using the ?Google Charts? api. Designed for web, this JavaScript-based api allowed us to feed-in our finalized data into a table and generate a configurable graph. Users can click on graph lines for more specific information.

V. Challenges and Solutions

A. Multiple Connection by Hadoop Collector

In multi-process data collector, each process takes ticker symbol list as input and concurrently creates http connection to http://finance.yahoo.com. The connections requests are coming from the same ip range and that makes the server to think someone is staging an DDoS 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 process sends a new connection request after a previous connection is completed.

B. Percentage change in a day or between two days

SQS's message visibility timeout mechanism is used for failure detection and recovery. When a worker reads a message from a queue, the message disappears from the queue for a certain period of time called visibility timeout of that message. The visibility timeout of a message can be reset by any worker to an appropriate value as required.

- Map worker failure: The input queue holds messages which are used for Map task assignment. SQS queues have a very short visibility timeout (default 30 seconds). If a Map worker cannot finish processing the assigned chunk of data within visibility timeout period, it keeps on resetting the visibility timeout of the associated message i.e. a key-value pair. When a worker finishes with the assigned chunk, it removes an associated message from the queue. If the Map worker fails before finishing a

Fig. 6: User Interface Snapshot

complete chunk of assigned data, the visibility timeout of message will not be reset, and that message will re-appear in the queue, which then can be taken by another Map worker for processing. The Map workers commit their status to simpleDB only when they have finished with the assigned portion of work. Partial output by the Mappers is simply discarded. We will see how duplicate messages are identified and discarded in the next point.

- Reduce worker failure: The visibility timeout mechanism of SQS is also used for failure detection and recovery of Reduce worker. However, two Reduce workers processing same Reduce queue can pose a problem, as they will cause to produce duplicate messages in the final output of the job. If it happens, simpleDB can be used for conflict resolution between different Reduce workers processing the same Reduce queue. We will discuss conflict resolution in the next section i.e. duplicate message.

C. Input data format

Whether to keep a single large file or a file per company? Again, it would become hard to exactly split data on a company boundary. This is tackled by keeping a separate file for each company.

D. Split Size

Even though a separate file is kept for each company, it could be splitted across many mappers depending upon a split size. To decide on a optimal split size so that a mappers will not have more than one company?s data, we iterated through the input folder. Iterating through the input folder was performed to find out the size of the biggest file. 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.

E. Sorting by values

In MR job 3 which calculates probabilities distribution for stock price movement, we need to sort not only keys but value also. The details about the how we achieved this explained in section XXXXX.

VI. Results and Evaluation

Each row in the output of the first two cascaded MR jobs is record ¡key and compositeValue¿ where Key = Company Symbol and CompositeValue is combination of two scores - first,

VII. Limitations

In this analytics, we do not consider mergers of companies. When two or more companies merge, we just discard earlier data of those company and process merged company?s data as a new company. The resulted company out of merger, will not have much data initially, so our results for this new company won?t be accurate enough as compared to other companies. Also, other factors such as political, social or economical are not considered for the prediction. However, even though we are considering statistical data only, our results shows enough accuracy in the prediction of stock market.

VIII. Conclusion and future work

This approach of stock market prediction only consider statistical data and do statistical analysis. However, other factors such as political, social could also affect the stock market. We are planning to incorporate those factors to into our prediction as well as a future scope.

Acknowledgment

Our special thanks to University of Maryland Baltimore County for providing High Performance Computing Facility to develop this project. We also grateful Dr. Karuna Joshi (one of the author) for continuous guidance and channeling our thoughts into the right direction throughput the project

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.
[2] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in *NSDI*, 2010, pp. 313–328.
[3] "Hadoop." [Online]. Available: http://hadoop.apache.org/
[4] Alec N.Kercheval, Yuan Zhang, Modeling high-frequency limit order book dynamicswith support vector machines, October 24, 2013.
[5] R. Karve, D. Dahiphale, and A. Chhajer, "Optimizing cloud mapreduce for processing stream data using pipelining," in *EMS*, 2011, pp. 344–349.
[6] Devendra Dahiphale, Rutvik Karve, Athanasios V. Vasilakos, Huan Liu, Zhiwei Yu, Amit Chhajer, Jianmin Wang, and Chaokun Wang, An Advanced MapReduce: Cloud MapReduce, Enhancements and Applications march 2013.
[7] O'Reilly Media, Hadoop: The Definitive Guide Book.
[8] http://www.trade-ideas.com/DOSA/
[9] https://vangjee.wordpress.com/2012/03/20/secondary-sorting-aka-sorting-values-in-hadoops-mapreduce-programming-paradigm/

Devendra D. Dahiphale pursuing Master's of Science degree in Computer Science from University of Maryland, Baltimore County. He has received the B.E. degree in Computer Science and Engineering from the University of Pune, India, in 2012. He also holds a Bachelor of Arts Degree in English from YCMOU and a Diploma in Teacher Education from the University of Pune. He worked as a Software Design Engineer at Imagination Technologies for three years. He is also a member of the Dreamz Group which is a group of software professionals who are passionate about innovation and technology development.