Cassandra Application: Stock Market Analysis – Let's make some money!

Team: 3

Chandni Pakalapati
Dept. of Computer Science
Rochester Institute of Technology
Rochester, New York
cp6023@cs.rit.edu

Priyanka Samanta
Dept. of Computer Science
Rochester Institute of Technology
Rochester, New York
ps7723@cs.rit.edu

Shashank Narkhede
Dept. of Computer Science
Rochester Institute of Technology
Rochester, New York
san8651@cs.rit.edu

Shraddha Atrawalkar
Dept. of Computer Science
Rochester Institute of Technology
Rochester, New York
ssa2923@cs.rit.edu

ABSTRACT

In this project report we would be providing a detailed analysis of how we exploited the distributed feature of No Sql database, Cassandra and developed our application on that.

1. INTRODUCTION

Cassandra is a No Sql database. Its primary and unique feature that distinguishes itself from the relational databases and other No Sql databases are its distributed nature and automatic replication feature.

We decided to build an application for Stock Analysis. The application was made to work in a distributed cluster of Cassandra.

2. DESIGN PHASE

2.1 Setting up the distributed cluster: For setting up the distributed cluster, we needed minimum of 3 nodes to be configured. Due to lack of resources, we had to stick in making use of our personal laptops in making the cluster. We created 3 virtual nodes using ***Oracle Virtual Box.*** Following we installed Ubuntu 14. 0 on each of the nodes. Once our nodes were up and running the following steps were performed to make Cassandra up and running on these nodes:

- **Installation of Java 8 on every node.**
- Installation of Cassandra 2.1.11 on these nodes.
- Changes in the configuration file ***'cassandra.yaml'***:
    - cluster_name= 'team3';
    - seeds: "192.168.56.101"
    - listen_address: "192.168.56.101"
    - rpc_address: "192.168.56.101"

Note: Seed contains the ip address of the single of the multiple nodes that's should be up before we turn on the other nodes of the cluster. Listen_address and the rpc_address the the host ip_address. For our cluster we had 3 nodes with the following ip addresses: 192.168.56.101, 192.168.56.102 and 192.168.56.103. We made the seed as 192.168.156.101. We named these machines as FrontNode, Node2 and Node3 respectively. From the virtual box we had to configure the Network Adaptor as ***"Host-only Adaptor".*** This is to ensure the nodes in the Cassandra cluster can perform peer-to-peer communication.

2.2 Starting the Server: After configuration the nodes in the cluster we had to ensure the sever was starting up and letting us work in the cluster. For bringing up the cluster, we had to first turn on our seed machine i.e the Front Node. And then Node2 and Node3 were brought up. The servers could be started using the following command:

sudo /usr/local/cassandra/bin/cassandra –f

Once all the nodes were up we could check the status of the cluster using the command:

sudo /usr/local/cassandra/bin/nodetool status

```
prisams@prisams-Node2:/usr/local/cassandra/bin$ ./nodetool status team3
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens  Owns (effective)  Host ID                               Rack
UN  192.168.56.101  315.04 KB  256     33.3%             5178c4a2-4116-4b20-a980-94a82e58bbe3  rack1
UN  192.168.56.102  348.32 KB  256     34.4%             e98f7ee5-f612-49ba-ac2b-c5eb5e722527  rack1
UN  192.168.56.103  453.33 KB  256     32.3%             1a835533-a54b-4f0a-a3ca-f44a1fc463ab  rack1
```

Fig 1: Status of the Cassandra Cluster

2.3 Creation of the database and ER diagram: For using Cassandra to create the database we had to create our own keyspace unlike any relational database and then create the tables in that. We created our keyspace with the following command:

CREATE KEYSPCE team3 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;

We kept our replication strategy as 'simple strategy' with replication factor as '1'. This would ensure that every node would

have equal load balancing with the data and would contain exactly once copy or replica of every data.
To build our Stock Analysis application we designed the below ER diagram.

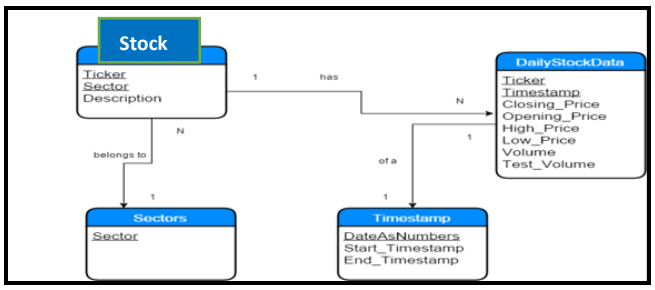

Fig 2: Entity Relationship Diagram

Our goal was to store all the tickers or symbols available under S&P index in the *Ticker* table. The tickers belonged to the different sectors like Infrastructure, Health, Finance, Information Technology and many more. We stored all the available sectors in the *Sector* table (we had total 13 sectors in our dataset). *DailyStockData* table was used to store all the stock data for every ticker. This was the main table which would be used in doing the various stock analysis. Further the *Timestamp* table was used to store the starting and ending timestamp for every date corresponding to the daily stock data.

We collect our stock data from yahoo finance website at runtime using HTTP get request which we implemented using Java.

2.4 Design of the Application: With our application we would be implementing the following functionalities:

1. Basic CRUD operations (Insertion of daily stock data, updation of the symbol description, retrieval of user query and deletion of stock data for a particular symbol.
2. Retrieval of the symbols based on a selected sector (this would give the user an idea of all the symbols available under his selected sector).
3. Retrieval of the selected symbols high and low price on a selected date (this would help the user get an idea of how his selected symbol has performed in the market on the selected date).
4. Retrieval of the maximum shares/quantity that was sold for the selected ticker by the user on a particular date (this would give the user an idea about the performance of the selected ticker on the date he selected).
5. Based on a selected sector, retrieval of the symbol that has performed the best on the selected date (this provides a notion to the user about the symbol to invest on from a particular sector based on its best performance).

All these were implemented using Java and we developed our UI using javafx which provides the user intractability with our Stock application.

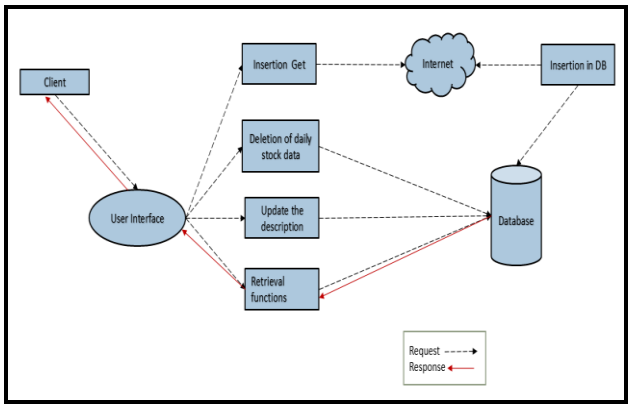

Fig 3: User Case Diagram of the Application

3. IMPLEMENTATION PHASE

3.1 Database Implementation: The following tables were created using the cqlsh interface in the team3 keyspace.

```
CREATE TABLE team3.dailystockdata (
    ticker text,
    timestamp int,
    closing_price double,
    high_price double,
    low_price double,
    opening_price double,
    test_volume double,
    volume double,
    PRIMARY KEY (ticker, timestamp)
)

CREATE TABLE team3.timestamp (
    dateasnumbers int PRIMARY KEY,
    end_timestamp int,
    start_timestamp int
)

CREATE TABLE team3.sectors (
    sector text PRIMARY KEY
)

CREATE TABLE team3.stock (
    ticker text,
    sector text,
    description text,
    PRIMARY KEY (ticker, sector)
) WITH CLUSTERING ORDER BY (sector ASC)
```

Stock and Sector table are static tables whose data pre prepopulated using the code PopulateS_PIndex.java and file Ticker.txt. List of tickers are present in the attached file Ticker.txt

3.2 Code Implementation: Following java classes were written to implement the functionalities mentioned in the design phase:

1. LaunchCassy.java (to Launch the UI application and handle the client's request)
2. TickerTableMethods.java (To get all the tickers and sectors present in the database and populate the UI)

3. GetStockData.java (Get the selected ticker and no. of days from the UI class and fetch the data from yahoo finance website and dump in a text file) and trigger PopulateDailyStockData.java and PopulateTimeStamp.java
4. PopulateDailyStock.java (Get the stock data dumped in the text file and insert the data in the DailyStockData table)
5. PopulateTimeStamp.java (Get the timestamp data dumped in the text file and insert the data in the Timestamp table)
6. TickerPerformanceforADay.java (Get the selected ticker and date from the UI class and retrieve the lowest and highest price of this ticker along with the timestamp on that date)
7. TickerMaxVolumeForADay.java (Get the selected ticker and date from the UI class and retrieve the maximum shares traded for this ticker along with the timestamp on that date)
8. BestTickerPerformanceInASector.java (Get the sector from the UI class and retrieve the best ticker from that sector)
9. UpdateTicker.java (Get the new description and ticker from the UI class and update the information in the database)
10. DeleteData.java (To get the ticker from the UI and delete the stock details from the database)

4. RESULT (Screenshots)

Below the screenshots of our application:

4.1 Our application User Interface:

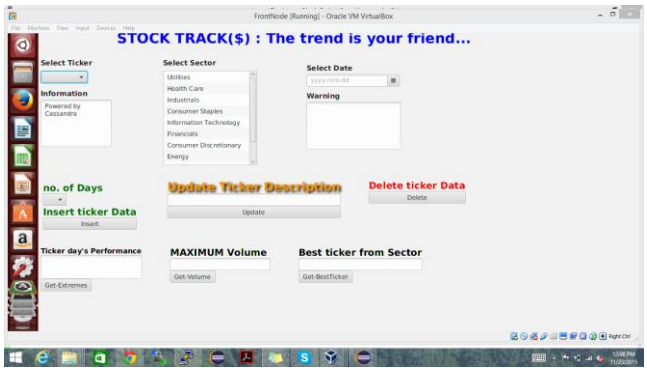

Fig 4: Application UI

4.2 Insertion Operation: Select a ticker from the dropdown list and select the no. of days from now you want the data to be inserted in the database. Click on **Insert.**

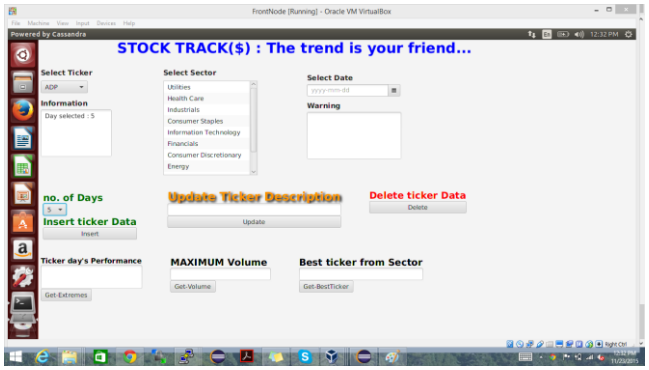

Fig 5: Insert Operation

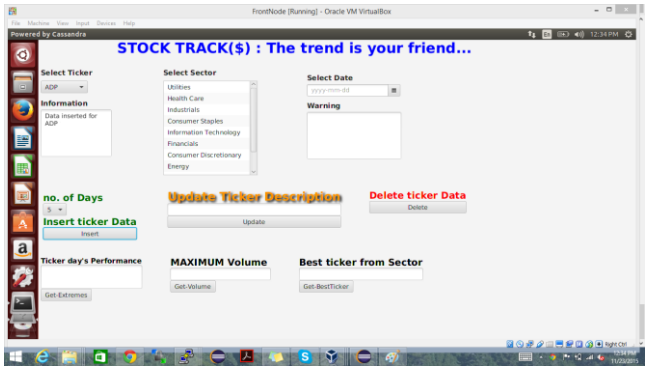

Fig 6: Data Inserted

4.3 Update Operation: Select a ticker from the dropdown list and insert the new description of the selected ticker in the textbox "Update Ticker Description". Click on **Update**.

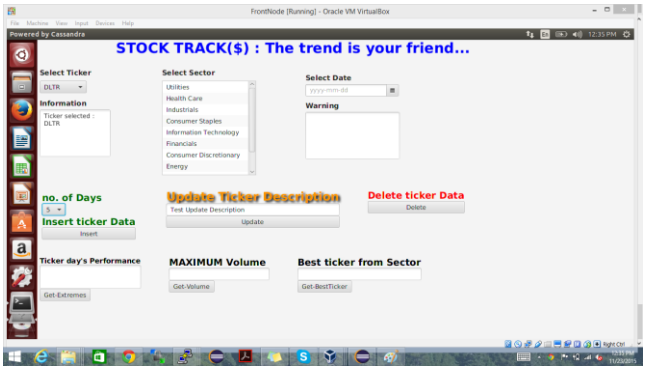

Fig 7: Data to be updated

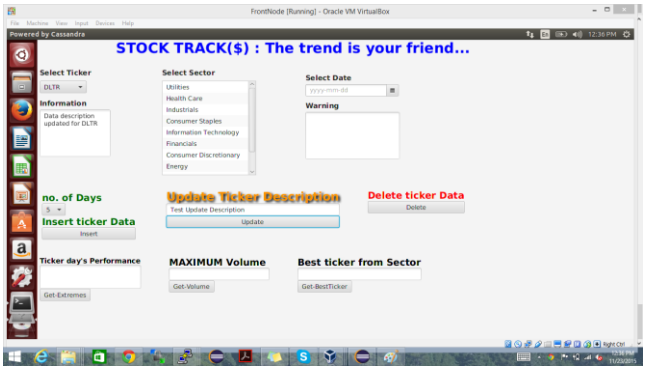

Fig 8: After Update

4.4 List of Sector based on selected Sector: On double clicking on a particular sector, the list of tickers under that sector would be enlisted.

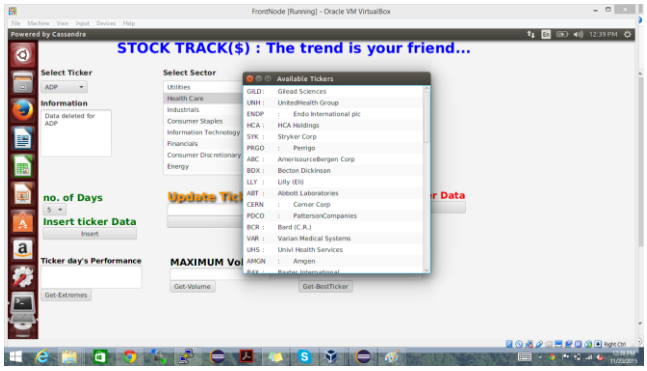

Fig 9: Tickers per selected Sector

4.5 Ticker Performance: Select a Ticker and a particular Date. Click on the **Get-Extremes.** This would display the high and low price as witnessed by the symbol on the selected date.

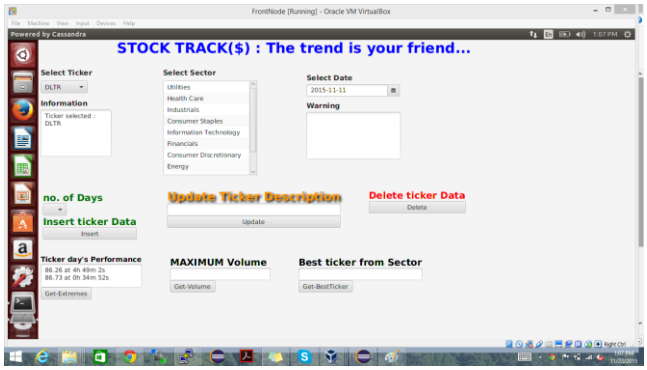

Fig 10: Ticker's High and Low Price with the timestamp on the selected date

4.5 Maximum Volume: Select a Ticker and a particular Date. Click on the **Get-Volume.** This would display the total shares or quantity traded for the selected ticker on the selected date.

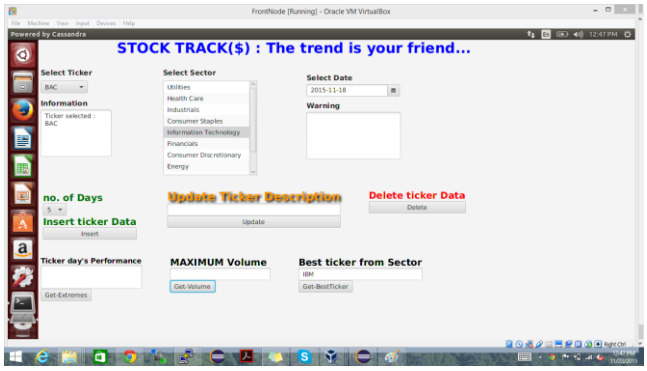

Fig 11: Maximum shares traded for ticker BAC on 11/18/2015

4.5 Best Ticker from Sector: Select any sector and choose a date. Click on the **Get-BestTicker.** This would retrieve the ticker that out performed on the selected date from the selected sector.

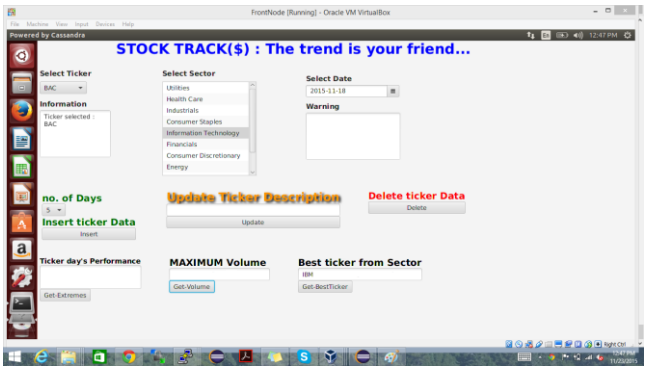

Fig 12: IBM performed the best on 11/18/2015 from IT sector

4.5 Delete: Select a particular ticker and click on the **Delete.** This would delete all the data for the selected ticker from the database.

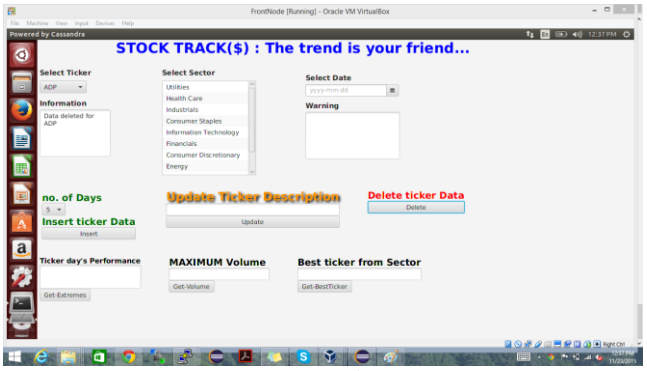

Fig 13: Data deletion for ticker ADP

5. ACCOMPLISHMENTS

We successfully exploited the distributed feature of No Sql database, Cassandra and built our application on the cluster database. This helped us to gain an in depth knowledge about Cassandra database and how to interact and work with the same.

6. CHALLENGES FACED

We wanted to go a step ahead and implement the distributed database cluster as it was interesting and something new for us.

1. In the initial stages, we faced issues in setting up the cluster, as it was something very new and proper resources were not available. But with a thorough research on the Datastax community we were able to achieve our goal.
2. Further using our personal laptop to set up the cluster, made our efficiency poor as we had to work with limited RAM which made our interaction with the database and implementation phase extremely slow.

7. REFERENCES

1. http://arturmkrtchyan.com/how-to-setup-multi-node-cassandra-2-cluster
2. https://academy.datastax.com/courses/ds201-cassandra-core-concepts