DISCIPLINED INNOVATION

BIG DATA ANALYSIS
CASE STUDY – ANALYZING STOCK DATA

Version: 1.0

Date: 25 August 2014

Author: Himanshu Agrawal

Introduction

This case study will evaluate various Big Data Analysis technologies to analyze the stocks data from the sample ‘New York Stock Exchange’ dataset and calculate the covariance for a stock and will target to solve both storage and processing problem related to huge volume of data.

Covariance: This finance term represents the degree or amount that two stocks or financial instruments move together or apart from each other. With covariance, investors have the opportunity to seek out different investment options based upon their respective risk profile. It is a statistical measure of how one investment moves in relation to another.

- A positive covariance means that asset returns moved together. If investment instruments or stocks tend to be up or down during the same time periods, they have positive covariance.

- A negative covariance means returns move inversely. If one investment instrument tends to be up while the other is down, they have negative covariance.

This will help a stock broker in recommending the stocks to his customers.

Problem Statement

1. Evaluate the existing Data Analysis Technologies and recommend the best possible solution for calculating the covariance for the huge stocks data analysis.
2. The application could be console based where user will be prompted to input the valid year in ‘YYYY’ format for which he/she wants to calculate the Covariance of Stocks Data.

Dataset: The sample dataset provided is a comma separated file (CSV) named ‘NYSE_daily_prices_Q.csv’ that contains the stock information such as daily quotes, Stock opening price, Stock highest price etc. at New York Stock Exchange.

Assumption: The dataset provided is just sample small dataset having around 3500 records, but in the real production environment there could be huge stock data into GBs or TBs.

So your solution must be supported in real production environment.

Data Analysis Technologies

Let's have a look on the existing open source Hadoop data analysis technologies to analyze the huge stock data being generated very frequently.

- MapReduce
  - Powerful model for parallelism.
  - Based on a rigid procedural structure.

- Pig
  - Procedural data-flow language
  - Used by Programmers and Researchers

- Hive
  - Declarative SQLish language
  - Used by Analysts generating daily reports

Features Comparison of Big Data Analysis Technologies

Feature	MapReduce	Pig	Hive
Language	Algorithm of Map and Reduce Functions (Can be implanted in C, Python, Java)	PigLatin(Scripting Language)	SQL-like
Schemas/Types	No	Yes(implicit)	Yes(Explicit)
Partitions	No	No	Yes
Server	No	No	Optional(Thrift)
Lines of code	More lines of code	Fewer (Around 10 lines of PIG = 200 lines of Java)	Fewer than MapReduce and Pig due to Sql Like nature
Development Time	More Development Effort	Rapid Development	Rapid Development

Abstraction	Lower level of abstraction (Rigid Procedural Structure)	Higher level of abstraction (Scripts)	Higher level of abstraction (Sql like)
Joins	Hard to achieve join functionality	Joins can be easily written	Easy for joins
Structured vs Semi-Structured Vs Unstructured data	Can deal on all these kind of data types	Works on all these kind of data types	Deal mostly with structured and semi-structured data
Complex business logic	More control for writing Complex Business logic	Less control for writing complex business logic	Less control for writing complex business logic
Performance	Fully tuned Mapreduce program would be faster than Pig/Hive	Slower than fully tuned Mapreduce program, but faster than bad written Mapreduce code	Slower than fully tuned Mapreduce program, but faster than bad written Mapreduce code

Which Data Analysis Technologies Should be used?

Based on the provided sample dataset, it is having following properties:

1. Data is having structured format
2. It would require joins to calculate Stock Covariance
3. It could be organized into schema
4. In real environment, data size would be too much

Based on these criteria's and comparing with the above analysis of features of these technologies, we can conclude:

1. If we are going to use **MapReduce**, then complex business logic needs to be written to handle the joins and much development effort will have to provide. We would not be able to map the data into schema format and all efforts need to be handled programmatically.
2. If we are going to use **Pig**, then we would not be able to partition the data, which can be used for sample processing from a subset of data by a particular stock symbol or particular date or month. In addition to that Pig is more like scripting language which is more suitable for prototyping and rapidly developing MapReduce based jobs. It also doesn't provide the facility to map our data into explicit schema format that would be required for this case study.
3. If we apply **Hive** to analyze the stock data, then we would be able to leverage the Sql capabilities of Hive-QL as well as data can be managed in a particular schema. It will also reduce the development time as well and can manage joins between stock data also using Hive-Ql which is

off course pretty difficult in Mapreduce. Hive also have its thrift servers, by which we can send our hive queries from anywhere to the hive server which in turn executes them.

So based on the above discussion, **Hive** seems the perfect choice for the mentioned case study.

Hive Setup on Cloudera

Since you are now done with zero-in to the data analysis technology part, now it’s the time to get your feet wet with Hive Setup, so that we can run hive commands and queries.

STEP 1:

Cloudera CDH3 Setup file:

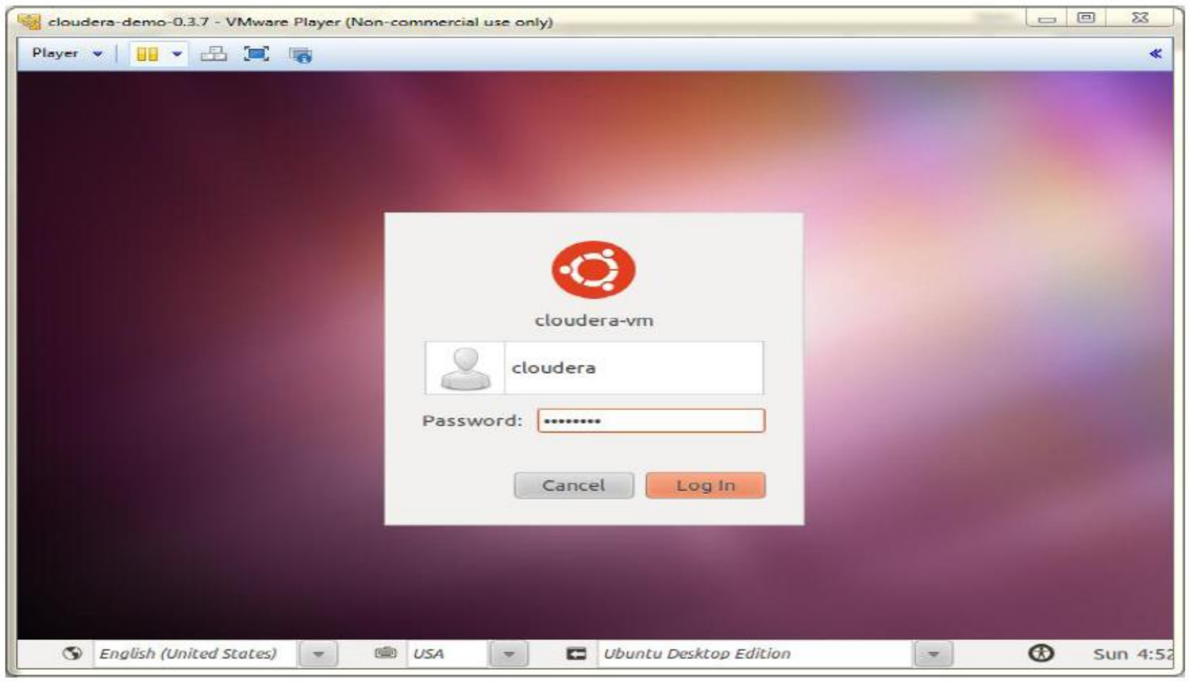
Download the Cloudera VM file from the below link.
https://docs.google.com/file/d/0B-_P02gj6T2mTGZKS3JzUTM3bjA/edit?usp=sharing

Extract this zip file and associate it with VmWare player.

STEP2:

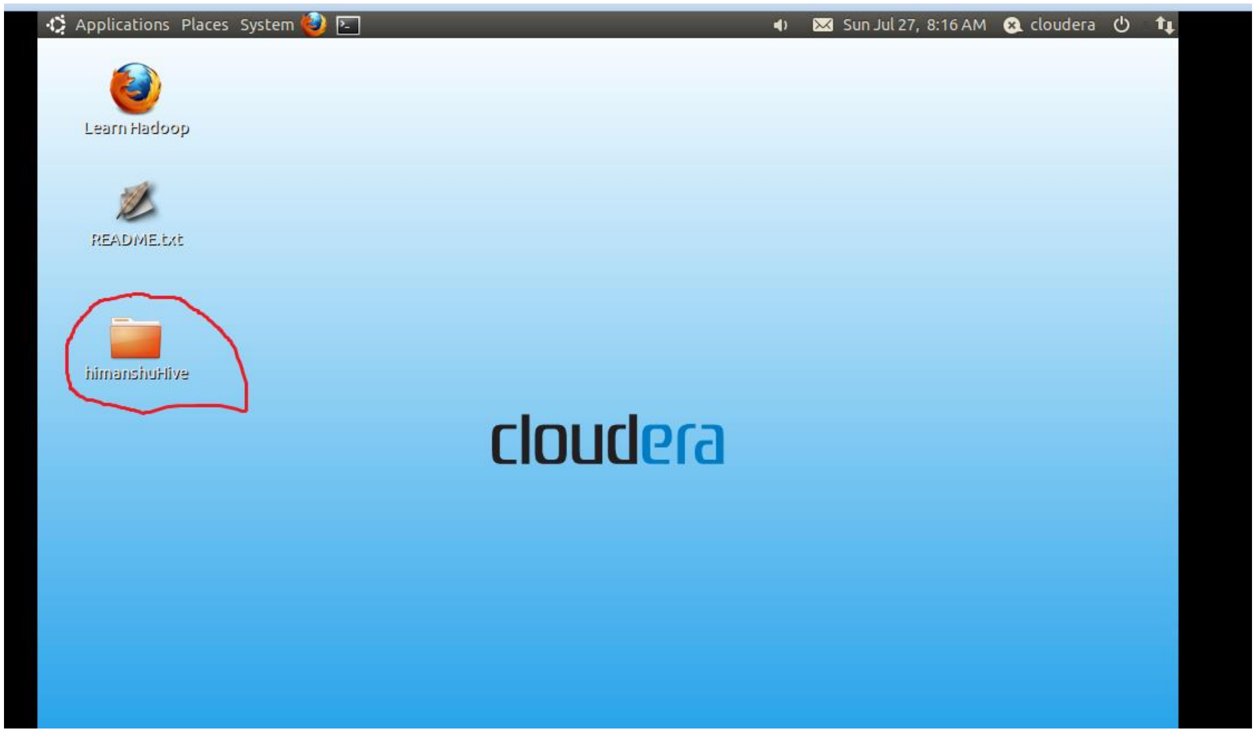
Click on play virtual machine and login on ClouderaVm as below:

Login as:
a. Username - cloudera
b. Password - cloudera



STEP 3:

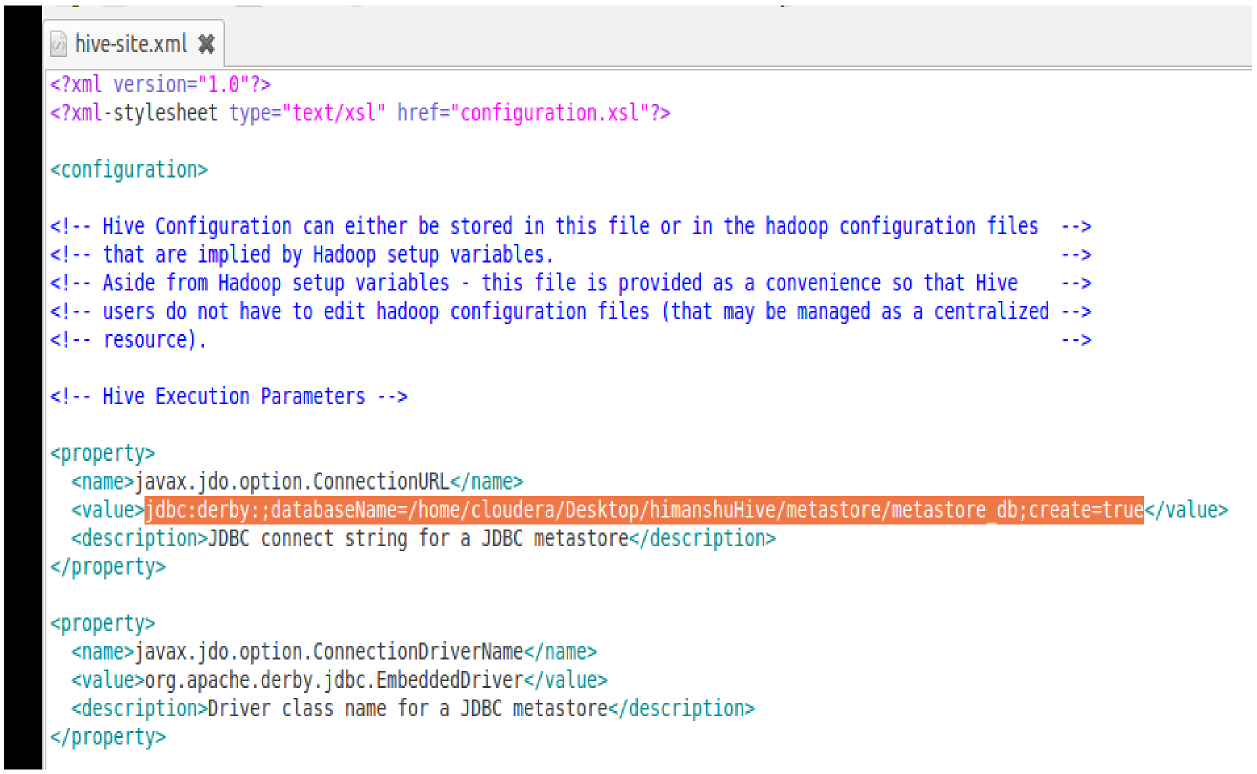
Create a folder with any name on the Cloudera Vm desktop. I provided it himanshuHive.



STEP 4:

- Open Terminal
- Type: **sudo su**
- If it asks for password type: **cloudera**
- root@cloudera-vm:/home/cloudera# > **cd /usr/lib/hive/conf/**
- root@cloudera-vm:/home/cloudera# > **sudo gedit hive-site.xml**
- Now you have to copy the path of the folder that you have created. (ie – himanshuHive)
- Change the property given below:

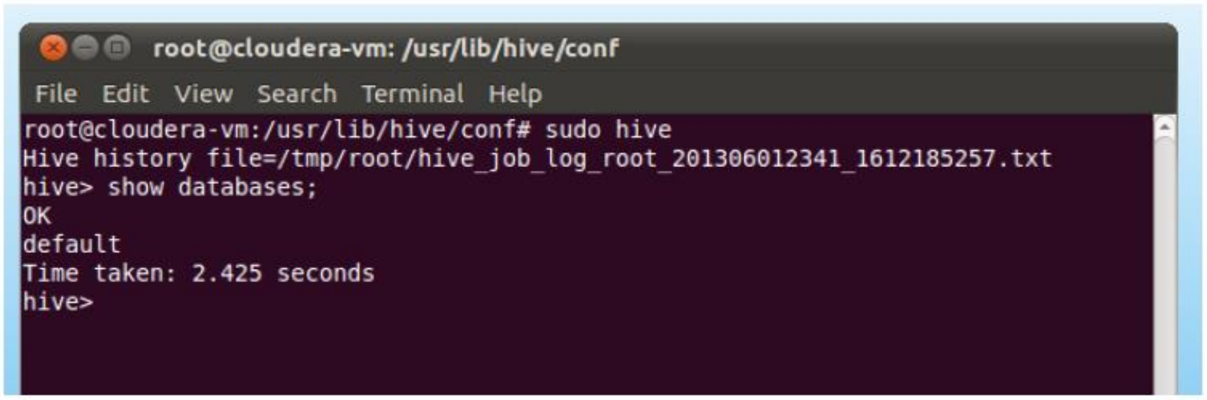
```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby:;databaseName=/home/cloudera/Desktop/himanshuHive/metastore/me
tastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>
```



```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- Hive Configuration can either be stored in this file or in the hadoop configuration files  -->
<!-- that are implied by Hadoop setup variables.                                                -->
<!-- Aside from Hadoop setup variables - this file is provided as a convenience so that Hive    -->
<!-- users do not have to edit hadoop configuration files (that may be managed as a centralized -->
<!-- resource).                                                                                 -->

<!-- Hive Execution Parameters -->

<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby:;databaseName=/home/cloudera/Desktop/himanshuHive/metastore/metastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>org.apache.derby.jdbc.EmbeddedDriver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>
```

STEP 5:

Always type this command to enter into hive shell: **sudo hive**


root@cloudera-vm: /usr/lib/hive/conf
File Edit View Search Terminal Help

```
root@cloudera-vm:/usr/lib/hive/conf# sudo hive
Hive history file=/tmp/root/hive_job_log_root_201306012341_1612185257.txt
hive> show databases;
OK
default
Time taken: 2.425 seconds
hive>
```

Now you are all set to execute your hive command and run hive queries into hive shell.

Problem Solution

To solve the provided problem, I have attached the POC which will serve the following purpose on high level:

1. Create Hive Table

Use 'create table' hive command to create the Hive table for our dataset:

```
hive> create table NYSE (exchange String,stock_symbol String,stock_date
String,stock_price_open double, stock_price_high double, stock_price_low double,
stock_price_close double, stock_volume double, stock_price_adj_close double)
row format delimited fields terminated by ',';
```

This will create a hive table named 'NYSE' in which rows would be delimited and row fields will be terminated by comma(,). This schema will be created into the embedded derby database as configured into Hive Setup.

2. Load Data into Hive Table

Use the following Hive command to load data into Hive table:

```
hive> load data local inpath '/home/cloudera/NYSE_daily_prices_Q.csv' into table NYSE;
```

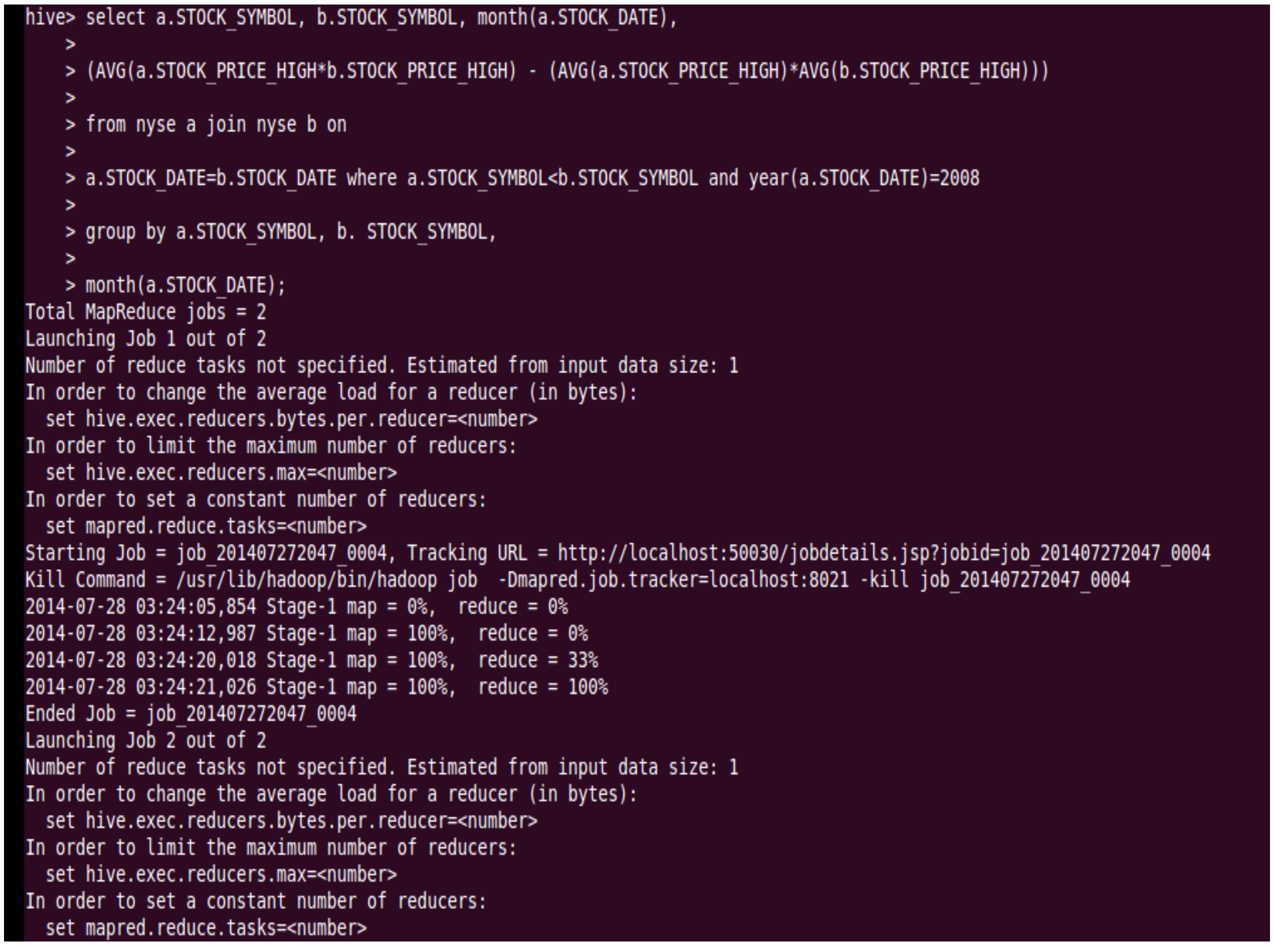
This will load the dataset from the mentioned location to the hive table 'NYSE' as created above but all this dataset will be stored into the Hive-controlled file system namespace on HDFS, so that it could be batch processed further by Mapreduce jobs or hive queries.

3. Calculate the Covariance

We can calculate the Covariance for the provided stock dataset for the inputted year as below using Hive select query:

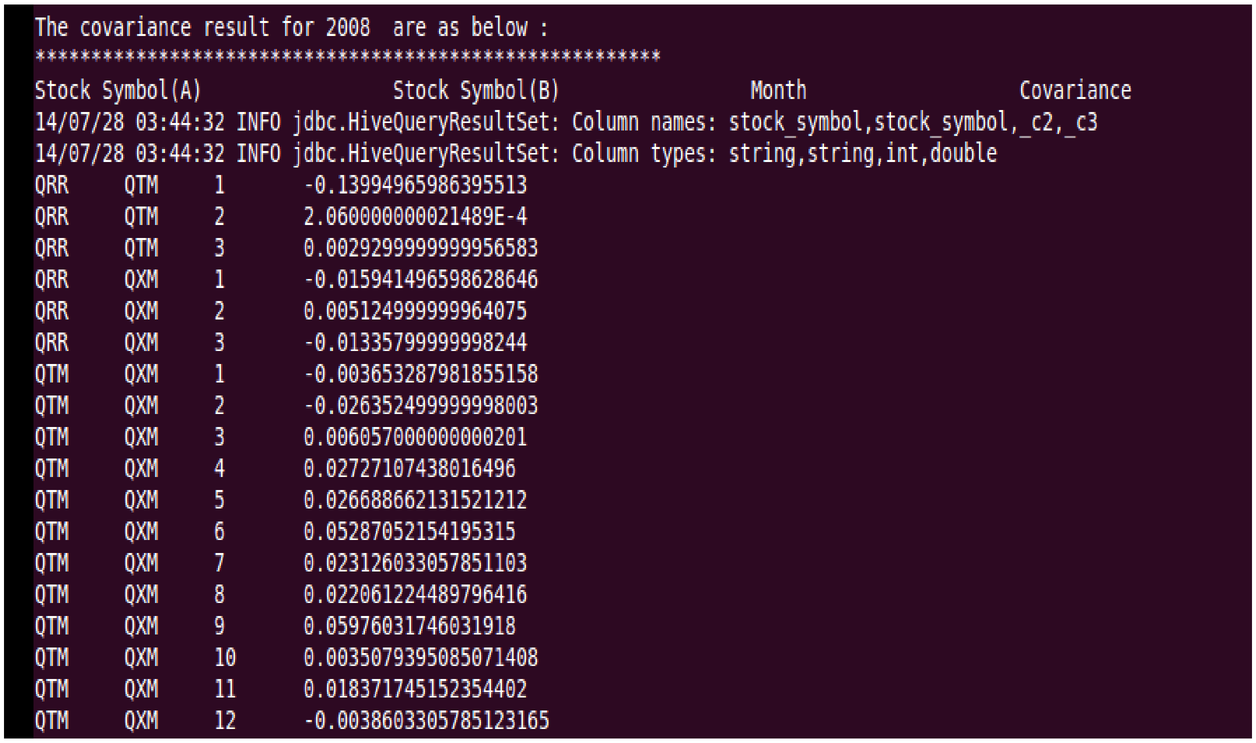
```
select a.STOCK_SYMBOL, b.STOCK_SYMBOL, month(a.STOCK_DATE),
        (AVG(a.STOCK_PRICE_HIGH*b.STOCK_PRICE_HIGH) -
        (AVG(a.STOCK_PRICE_HIGH)*AVG(b.STOCK_PRICE_HIGH)))
from NYSE a join NYSE b on
        a.STOCK_DATE=b.STOCK_DATE where a.STOCK_SYMBOL<b.STOCK_SYMBOL and
        year(a.STOCK_DATE)=2008
Group by a.STOCK_SYMBOL,  b. STOCK_SYMBOL, month(a.STOCK_DATE);
```

This hive select query will trigger the Mapreduce job as below:



```
hive> select a.STOCK_SYMBOL, b.STOCK_SYMBOL, month(a.STOCK_DATE),
    >
    > (AVG(a.STOCK_PRICE_HIGH*b.STOCK_PRICE_HIGH) - (AVG(a.STOCK_PRICE_HIGH)*AVG(b.STOCK_PRICE_HIGH)))
    >
    > from nyse a join nyse b on
    >
    > a.STOCK_DATE=b.STOCK_DATE where a.STOCK_SYMBOL<b.STOCK_SYMBOL and year(a.STOCK_DATE)=2008
    >
    > group by a.STOCK_SYMBOL, b. STOCK_SYMBOL,
    >
    > month(a.STOCK_DATE);
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201407272047_0004, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201407272047_0004
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201407272047_0004
2014-07-28 03:24:05,854 Stage-1 map = 0%,  reduce = 0%
2014-07-28 03:24:12,987 Stage-1 map = 100%,  reduce = 0%
2014-07-28 03:24:20,018 Stage-1 map = 100%,  reduce = 33%
2014-07-28 03:24:21,026 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201407272047_0004
Launching Job 2 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
```

The covariance results after the above stock data analysis are as below:



```
The covariance result for 2008  are as below :
***********************************************************
Stock Symbol(A)                Stock Symbol(B)                Month                Covariance
14/07/28 03:44:32 INFO jdbc.HiveQueryResultSet: Column names: stock_symbol,stock_symbol,_c2,_c3
14/07/28 03:44:32 INFO jdbc.HiveQueryResultSet: Column types: string,string,int,double
QRR     QTM     1       -0.13994965986395513
QRR     QTM     2       2.060000000021489E-4
QRR     QTM     3       0.0029299999999956583
QRR     QXM     1       -0.015941496598628646
QRR     QXM     2       0.005124999999964075
QRR     QXM     3       -0.01335799999998244
QTM     QXM     1       -0.003653287981855158
QTM     QXM     2       -0.026352499999998003
QTM     QXM     3       0.006057000000000201
QTM     QXM     4       0.02727107438016496
QTM     QXM     5       0.026688662131521212
QTM     QXM     6       0.05287052154195315
QTM     QXM     7       0.023126033057851103
QTM     QXM     8       0.022061224489796416
QTM     QXM     9       0.05976031746031918
QTM     QXM     10      0.0035079395085071408
QTM     QXM     11      0.018371745152354402
QTM     QXM     12      -0.0038603305785123165
```

The covariance has been calculated between two different stocks for each month on a particular date for the provided year.
From the covariance results, Stock brokers or fund managers can provide below recommendations:

- For Stocks QRR and QTM, these are having more positive covariance than negative covariance, so having high probability that stocks will move together in same direction.
- For Stocks QRR and QXM, these are mostly having negative covariance, so there exist greater probability of stock prices move in inverse direction.
- For Stocks QTM and QXM, these are mostly having positive covariance for most of all months, so tend to move in same direction mostly.

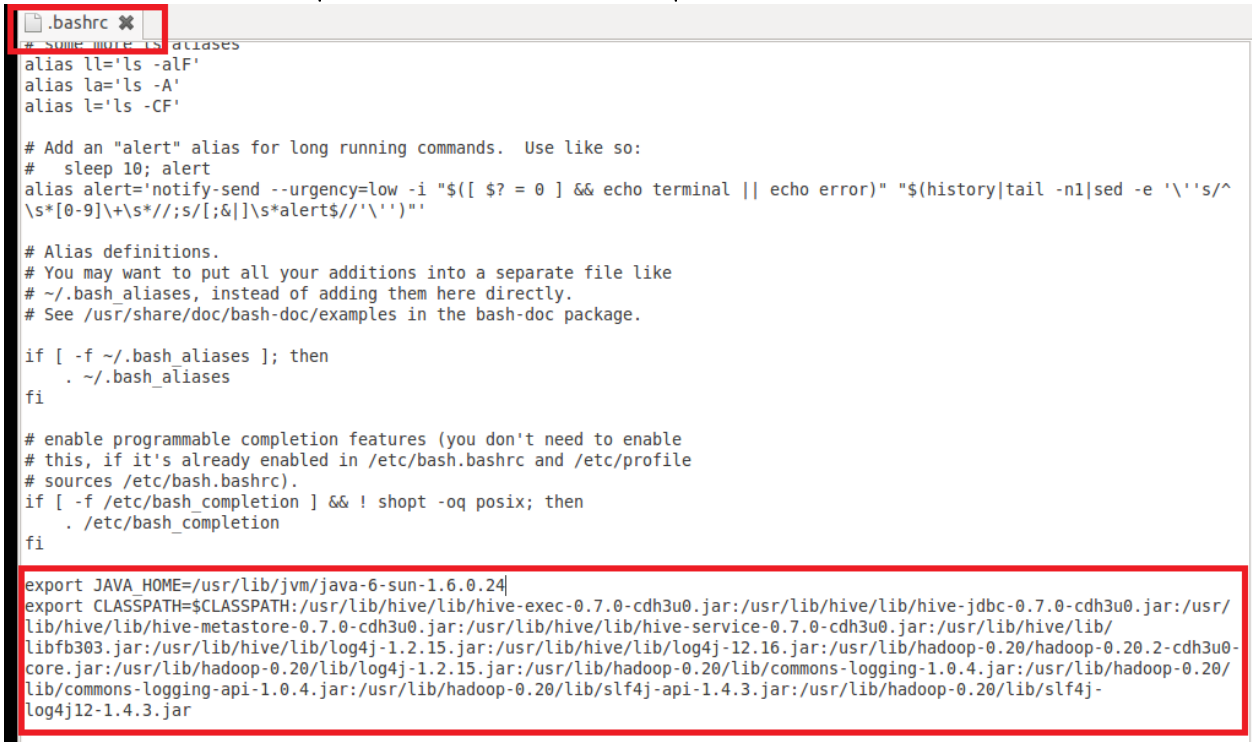
Running the Stock Covariance Application

To execute the attached Stock Covariance application, follow the below steps:

- Extract the attached Poc into some destination folder for eg. into Cloudera home directory

Setting Hive Classpath:

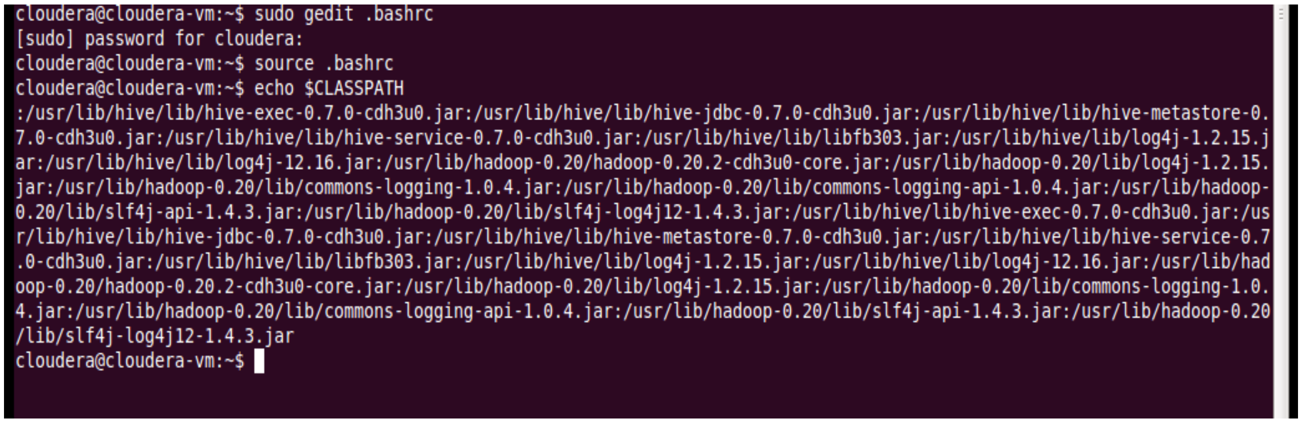
- Open a new terminal and type sudo gedit .bashrc
- Provide Java Home Environment variable into .bashrc file at the end as below:
  export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.24
- Provide hive driver and dependencies libraries into classpath as below:



```
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^
\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.24
export CLASSPATH=$CLASSPATH:/usr/lib/hive/lib/hive-exec-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-jdbc-0.7.0-cdh3u0.jar:/usr/
lib/hive/lib/hive-metastore-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-service-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/
libfb303.jar:/usr/lib/hive/lib/log4j-1.2.15.jar:/usr/lib/hive/lib/log4j-12.16.jar:/usr/lib/hadoop-0.20/hadoop-0.20.2-cdh3u0-
core.jar:/usr/lib/hadoop-0.20/lib/log4j-1.2.15.jar:/usr/lib/hadoop-0.20/lib/commons-logging-1.0.4.jar:/usr/lib/hadoop-0.20/
lib/commons-logging-api-1.0.4.jar:/usr/lib/hadoop-0.20/lib/slf4j-api-1.4.3.jar:/usr/lib/hadoop-0.20/lib/slf4j-
log4j12-1.4.3.jar
```

- Now close this .bashrc file and type the below command:
  cloudera@cloudera-vm:~$ source .bashrc
  This will actually set the Classpath into bash shell.

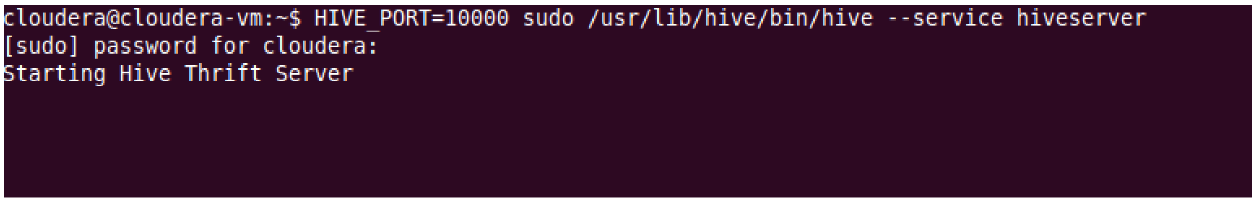
- Verify if Classpath is properly set:
  cloudera@cloudera-vm:~$ echo $CLASSPATH



```
cloudera@cloudera-vm:~$ sudo gedit .bashrc
[sudo] password for cloudera:
cloudera@cloudera-vm:~$ source .bashrc
cloudera@cloudera-vm:~$ echo $CLASSPATH
:/usr/lib/hive/lib/hive-exec-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-jdbc-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-metastore-0.
7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-service-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/libfb303.jar:/usr/lib/hive/lib/log4j-1.2.15.j
ar:/usr/lib/hive/lib/log4j-12.16.jar:/usr/lib/hadoop-0.20/hadoop-0.20.2-cdh3u0-core.jar:/usr/lib/hadoop-0.20/lib/log4j-1.2.15.
jar:/usr/lib/hadoop-0.20/lib/commons-logging-1.0.4.jar:/usr/lib/hadoop-0.20/lib/commons-logging-api-1.0.4.jar:/usr/lib/hadoop-
0.20/lib/slf4j-api-1.4.3.jar:/usr/lib/hadoop-0.20/lib/slf4j-log4j12-1.4.3.jar:/usr/lib/hive/lib/hive-exec-0.7.0-cdh3u0.jar:/us
r/lib/hive/lib/hive-jdbc-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-metastore-0.7.0-cdh3u0.jar:/usr/lib/hive/lib/hive-service-0.7
.0-cdh3u0.jar:/usr/lib/hive/lib/libfb303.jar:/usr/lib/hive/lib/log4j-1.2.15.jar:/usr/lib/hive/lib/log4j-12.16.jar:/usr/lib/had
oop-0.20/hadoop-0.20.2-cdh3u0-core.jar:/usr/lib/hadoop-0.20/lib/log4j-1.2.15.jar:/usr/lib/hadoop-0.20/lib/commons-logging-1.0.
4.jar:/usr/lib/hadoop-0.20/lib/commons-logging-api-1.0.4.jar:/usr/lib/hadoop-0.20/lib/slf4j-api-1.4.3.jar:/usr/lib/hadoop-0.20
/lib/slf4j-log4j12-1.4.3.jar
cloudera@cloudera-vm:~$
```

Start Hive Server:

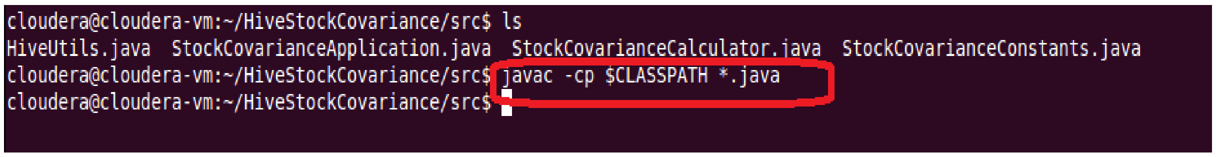
Open a new terminal and type below command to Start the thrift server :
HIVE_PORT=10000 sudo /usr/lib/hive/bin/hive --service hiveserver



```
cloudera@cloudera-vm:~$ HIVE_PORT=10000 sudo /usr/lib/hive/bin/hive --service hiveserver
[sudo] password for cloudera:
Starting Hive Thrift Server
```

Compile and Run the Application:

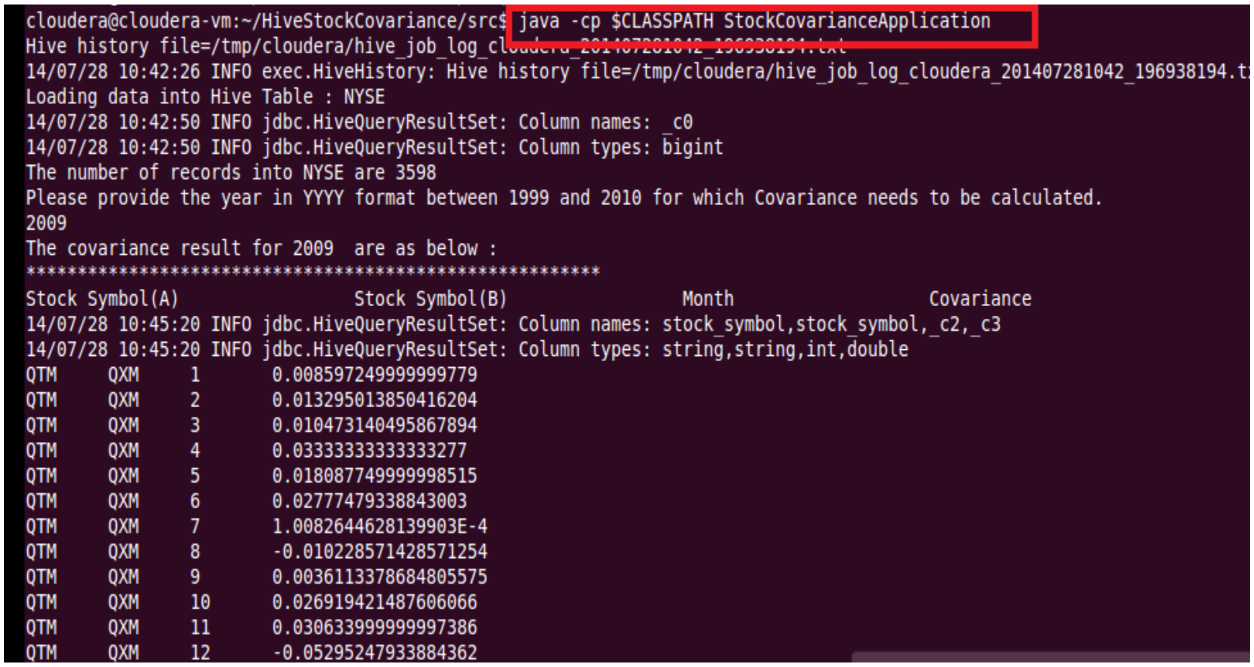
- Open the first terminal and navigate to directory where source java classes are present
- Compile the java classes as below:



```
cloudera@cloudera-vm:~/HiveStockCovariance/src$ ls
HiveUtils.java  StockCovarianceApplication.java  StockCovarianceCalculator.java  StockCovarianceConstants.java
cloudera@cloudera-vm:~/HiveStockCovariance/src$ javac -cp $CLASSPATH *.java
cloudera@cloudera-vm:~/HiveStockCovariance/src$
```

It will compile all the source classes present in the folder.

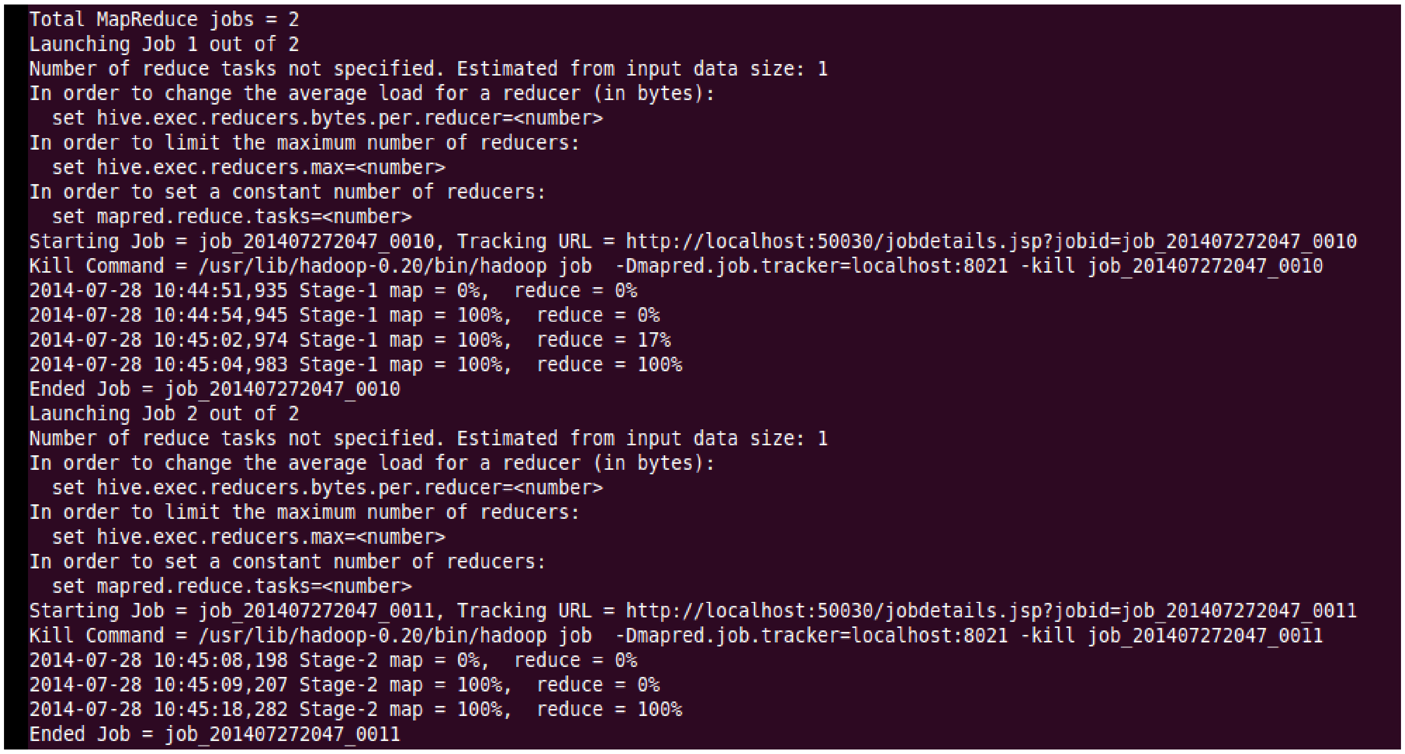
- Now finally execute the application as below:



```
cloudera@cloudera-vm:~/HiveStockCovariance/src$ java -cp $CLASSPATH StockCovarianceApplication
Hive history file=/tmp/cloudera/hive_job_log_cloudera_201407281042_196938194.txt
14/07/28 10:42:26 INFO exec.HiveHistory: Hive history file=/tmp/cloudera/hive_job_log_cloudera_201407281042_196938194.t
Loading data into Hive Table : NYSE
14/07/28 10:42:50 INFO jdbc.HiveQueryResultSet: Column names: _c0
14/07/28 10:42:50 INFO jdbc.HiveQueryResultSet: Column types: bigint
The number of records into NYSE are 3598
Please provide the year in YYYY format between 1999 and 2010 for which Covariance needs to be calculated.
2009
The covariance result for 2009  are as below :
*********************************************************
Stock Symbol(A)                 Stock Symbol(B)                 Month                   Covariance
14/07/28 10:45:20 INFO jdbc.HiveQueryResultSet: Column names: stock_symbol,stock_symbol,_c2,_c3
14/07/28 10:45:20 INFO jdbc.HiveQueryResultSet: Column types: string,string,int,double
QTM     QXM     1       0.008597249999999779
QTM     QXM     2       0.013295013850416204
QTM     QXM     3       0.010473140495867894
QTM     QXM     4       0.03333333333333277
QTM     QXM     5       0.018087749999998515
QTM     QXM     6       0.02777479338843003
QTM     QXM     7       1.0082644628139903E-4
QTM     QXM     8       -0.010228571428571254
QTM     QXM     9       0.0036113378684805575
QTM     QXM     10      0.026919421487606066
QTM     QXM     11      0.030633999999997386
QTM     QXM     12      -0.05295247933884362
```

This will execute the application and will prompt for the year for which covariance need to be calculated.

- In the second terminal, we can analyze the Mapreduce jobs or server logs that are being created by the hive server itself.



```
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201407272047_0010, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201407272047_0010
Kill Command = /usr/lib/hadoop-0.20/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201407272047_0010
2014-07-28 10:44:51,935 Stage-1 map = 0%,  reduce = 0%
2014-07-28 10:44:54,945 Stage-1 map = 100%,  reduce = 0%
2014-07-28 10:45:02,974 Stage-1 map = 100%,  reduce = 17%
2014-07-28 10:45:04,983 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201407272047_0010
Launching Job 2 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201407272047_0011, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201407272047_0011
Kill Command = /usr/lib/hadoop-0.20/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201407272047_0011
2014-07-28 10:45:08,198 Stage-2 map = 0%,  reduce = 0%
2014-07-28 10:45:09,207 Stage-2 map = 100%,  reduce = 0%
2014-07-28 10:45:18,282 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_201407272047_0011
```

So similarly we can analyze more use cases of big data and can explore all possible solution to solve that use case and then by the comparison chart, the final best solution can be narrowed down.

Conclusion/Benefits

So this case study solves the ultimate both goals of Big Data technologies in the best possible way.

1. Storage :

By storing the huge stock data into HDFS, the solution provided is much robust, reliable, economical and scalable. Whenever data size is increasing, we can just add some more nodes, configure into Hadoop and that's all. If sometime any node is down, then even other nodes are ready to handle the responsibility due to data replication.
By managing the hive schema into embedded database or any other standard sql database, we are able to utilize the power of Sql as well.

2. Processing:

Since Hive schema is created on standard sql database, we get the advantage of running Sql queries on the huge dataset also and able to process GBs or TBs of data with simple Sql queries. Since actual data resides into HDFS, so these hive sql queries being converted into Mapreduce job and these parallelized map reduce job processes these huge volume of data and achieve scalability and fault tolerance.