A REPORT ON THE PROJECT ENTITLED

“Analysis of Stock Market Using Hadoop and Map Reduce”

INDEX

1. ABSTRACT

Stock Market has huge high risk and high profit. That is why its analysis is very important. The Stock Market contains huge amounts of data measuring in GBs or TBs. These are very complex and should be analyzed by data mining methods. Large computations are more costly and time consuming. Distributed computing provides cost effective and scalable solutions to such problems.

2. INTRODUCTION

Volatility is a statistical measure of the dispersion of returns for a given security or market index. In most cases, the higher the volatility, the riskier the security. Volatility is often measured as either the standard deviation or variance between returns from that same security or market index.

To compute volatility over large data using normal computing takes a lot of time. In order to provide trade off between speed and cost of computing one can use distributed computing and solve the issue using map reduce and hadoop framework.

In the securities markets, volatility is often associated with big swings in either direction. For example, when the stock market rises and falls more than one percent over a sustained period of time, it is called a "volatile" market. An asset's volatility is a key factor when pricing options contracts. This way one can be cautious about a certain stock using this volatility.

Following are the terms that are used in relations of the dataset of an example stock AAPL.

Date represents the date of the stock AAPL
Open represents the open price in that day of stock AAPL
High represents the highest price in that day of stock AAPL
Low represents the lowest price in that day of stock AAPL
Adj Close represents the close price in that day of stock AAPL
Volume represents the volume in that day of stock AAPL

3. EXISTING SYSTEM

RDBMS stores data in a structural way. Scaling up (upgrading) is very expensive. Basic data unit is relational tables. Sequential or traditional computing takes a lot of time. It adds more cost and time.

Solution of scaling out and trade off cannot be made easily using traditional computing. One cheap processing batch unit costs around Rs. 30,000 whereas a standard machine with good power costs more than 5 times of cheap processing unit. These cheap processing units can be repaired and even if lost it won't add much expense to management.

One can even easily set up a hadoop batch with a set of 3-4 laptops and perform distributed computing which was very tough with traditional approaches.

3. PROPOSED SYSTEM

This project uses Mapreduce on a Hadoop environment to compute the monthly volatility of stocks. The database is of 2970 stocks on NASDAQ market for 3 years from 01/01/2012 to 12/31/2014 (except holidays, otherwise called trading days). Our job is to analyze stock price data, and find out which stocks in a certain period have higher earnings potential, etc. One characteristic that is widely used by traders is the volatility index. Our data is 2970 CSV format files (Comma Separated). Each file contains the data for one stock using its symbol as the file name.

4. IMPLEMENTATION DETAILS

This project uses Mapreduce paradigm of Hadoop to serialize the calculation of volatility for each month and computes the top 10 most volatile and bottom 10 least volatile stock values.

- Number of Mapper Implementation : 3
- Number of Reducer Implementation : 3

Roles of each Mapper and Reducer:

Mapper1

• splits the input data and options the date and close adjacent value.

• key - stock_name + month + year

• value - date + adjacent close value

Reducer1

• Since after the map step the values which have the same key are grouped together and passed to the reducer as iterable, values that correspond to specific month and year of the particular stock are grouped together.

• Beginning adjacent close value and end adjacent close value are obtained by integrating through the iterable and the value of xi for the corresponding month is computed.

• Key - Company Name

• Value - Computed X_i

Mapper2

• Now we have to consolidate all the values obtained from the reducer with respect to the company name.

• Key - Company Name

• Value - X_i

Reducer2

• All the xi corresponding to the the respective companies are grouped together.

• Volatility for the particular company is obtained from these values.

• Key - Company Name

• Value - Volatility

Mapper3:

• All the companies are grouped together with a common key.

• Key - Common

• Value = Company Name + Volatility

<u>Reducer3</u>:

• Obtained iterable contains all the company name with values and they are sorted by a custom comparator.

• Top 10 and bottom 10 values are obtained from the List

5. ADVANTAGES

a) Scaling Out: In Traditional RDBMS it is quite difficult to add more hardware, software resources i.e. scale up. In Hadoop this can be easily done i.e. scale down.

b) Transfer code to data In RDBMS generally data is moved to code and results are stored back. As data is moving there is always a security threat. In Hadoop small code is moved to data and it is executed there itself. Thus data is local. Thus Hadoop correlates preprocessors and storage.

c) Fault Tolerance: Hadoop is designed to cope up with node failures. As a large number of machines are there, a node failure is a very common problem.

d) Abstraction of complexities Hadoop provides proper interfaces between components for proper working. e) Data protection and consistency Hadoop handles system level challenges as it supports data consistency.

a) Low cost As Hadoop is an open-source framework, it is free. It uses commodity hardware to store and process huge data. Hence not much costly.

b) High Computing power: Hadoop uses a distributed computing model. Due to this, tasks can be distributed amongst different nodes and can be processed quickly. Cluster has thousands of nodes which gives high computing capability to Hadoop.

6. LIMITATIONS

This approach may lead to the issue of HDFS error due to clustering. This project formulated the overall problem as an interaction problem between namenode and data node and solved it by creating a temp directory for data node.

Apache Hadoop is for batch processing, which means it takes a huge amount of data in input, processes it and produces the result. Although batch processing is very efficient for processing a high volume of data, depending on the size of the data that processes and the computational power of the system, an output can delay significantly. Hadoop is not suitable for Real-time data processing.

This can be solved by Apache Spark and Apache Flink.

Hadoop is not so efficient for iterative processing, as Hadoop does not support cyclic data flow.

Hadoop is challenging in managing complex applications. If the user doesn't know how to enable a platform who is managing the platform, your data can be a huge risk. At storage and network levels, Hadoop is missing encryption, which is a major point of concern.

This can be solved by, third-party vendors have enabled an organization to leverage Active Directory Kerberos and LDAP for authentication.

In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work

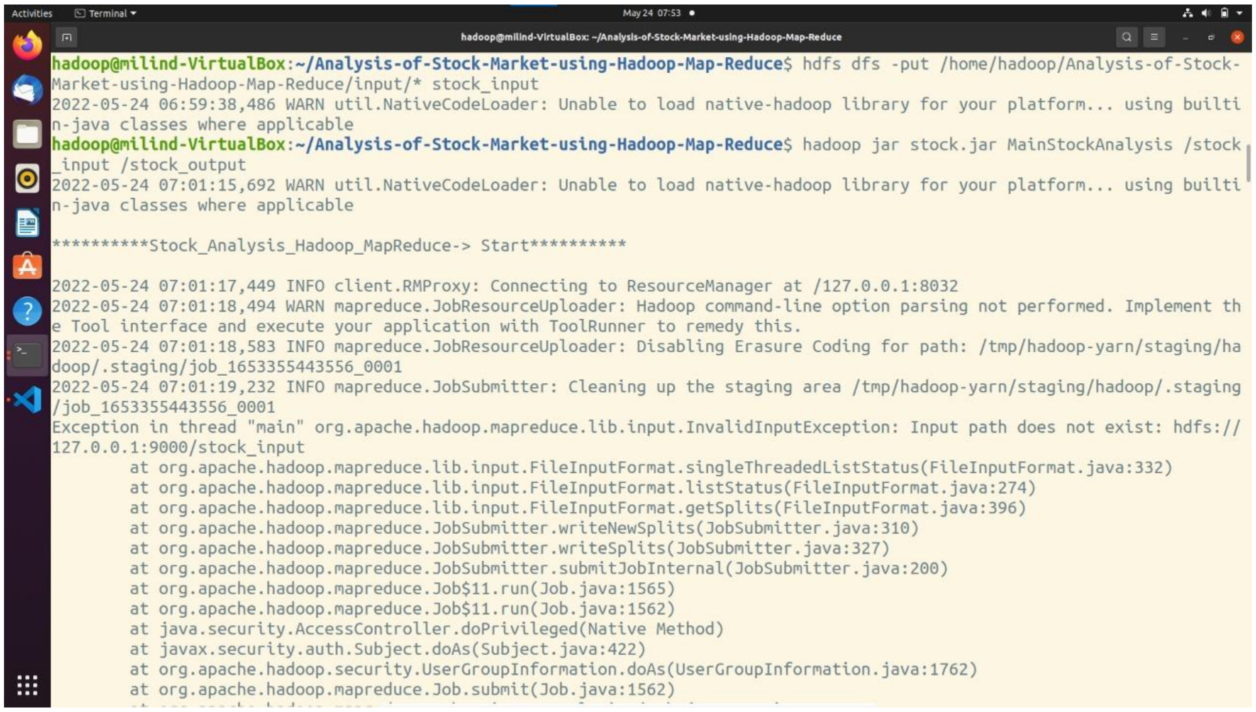
7. FUTURE WORK

This project can be scaled to work in conjunction with Machine Learning Algorithms that can use these kinds of Map Reduce applications to handle large data sets with multiple Master nodes.

8. CONCLUSION

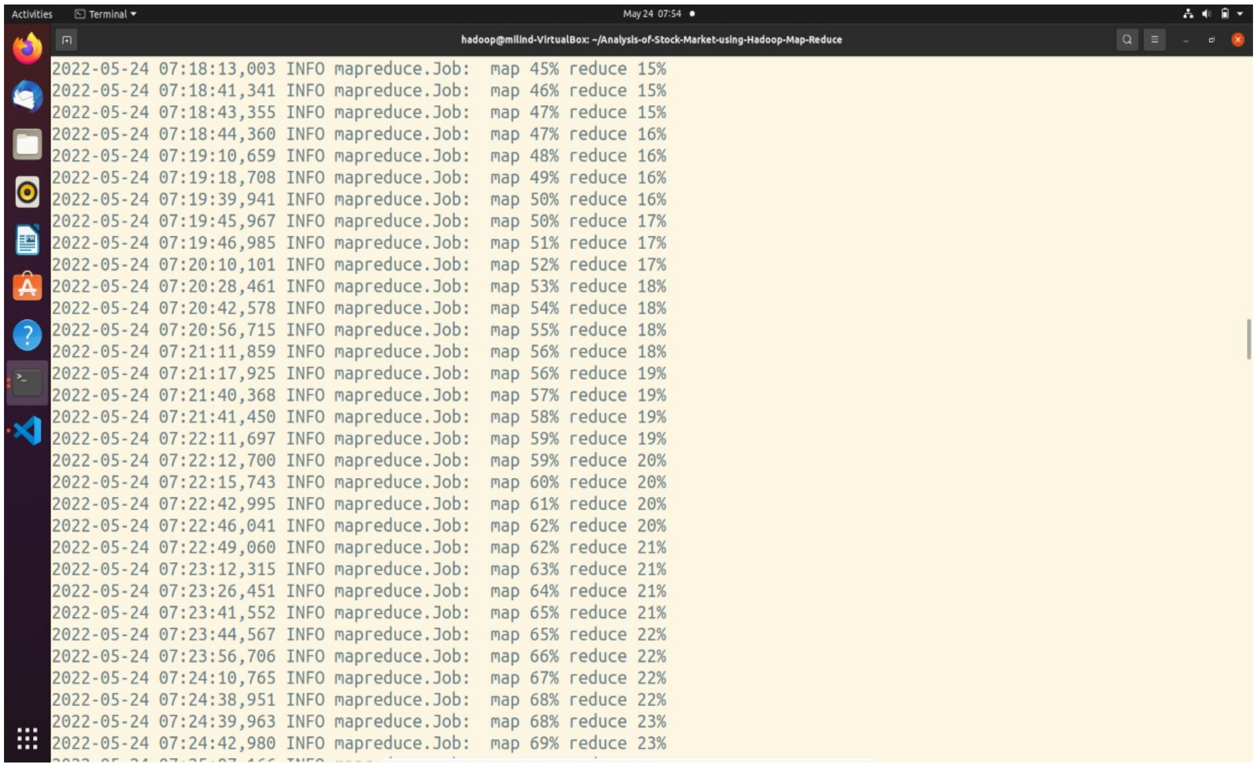
This project proposes a Triple Map Reduce Model that selects data from different sources with the help of Hadoop Distributed File System (HDFS) such that the stock price of volatility can be used for either minimum 10 or maximum 10.

Results:


Activities
Terminal
May 24 07:53
hadoop@milind-VirtualBox: ~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce

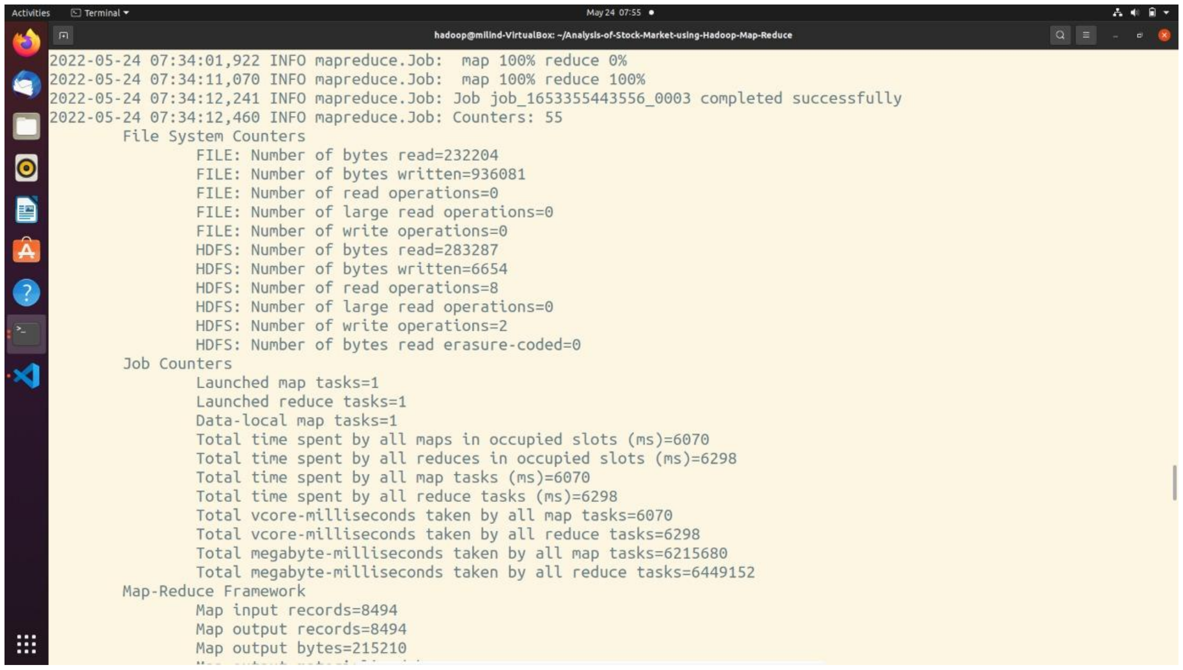
```
hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce$ hdfs dfs -put /home/hadoop/Analysis-of-Stock-
Market-using-Hadoop-Map-Reduce/input/* stock_input
2022-05-24 06:59:38,486 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce$ hadoop jar stock.jar MainStockAnalysis /stock
_input /stock_output
2022-05-24 07:01:15,692 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable

**********Stock_Analysis_Hadoop_MapReduce-> Start**********

2022-05-24 07:01:17,449 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2022-05-24 07:01:18,494 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement th
e Tool interface and execute your application with ToolRunner to remedy this.
2022-05-24 07:01:18,583 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/ha
doop/.staging/job_1653355443556_0001
2022-05-24 07:01:19,232 INFO mapreduce.JobSubmitter: Cleaning up the staging area /tmp/hadoop-yarn/staging/hadoop/.staging
/job_1653355443556_0001
Exception in thread "main" org.apache.hadoop.mapreduce.lib.input.InvalidInputException: Input path does not exist: hdfs://
127.0.0.1:9000/stock_input
        at org.apache.hadoop.mapreduce.lib.input.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:332)
        at org.apache.hadoop.mapreduce.lib.input.FileInputFormat.listStatus(FileInputFormat.java:274)
        at org.apache.hadoop.mapreduce.lib.input.FileInputFormat.getSplits(FileInputFormat.java:396)
        at org.apache.hadoop.mapreduce.JobSubmitter.writeNewSplits(JobSubmitter.java:310)
        at org.apache.hadoop.mapreduce.JobSubmitter.writeSplits(JobSubmitter.java:327)
        at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:200)
        at org.apache.hadoop.mapreduce.Job$11.run(Job.java:1565)
        at org.apache.hadoop.mapreduce.Job$11.run(Job.java:1562)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapreduce.Job.submit(Job.java:1562)
```

fig: commands to put our input on hadoop distributed file system and run the program


Activities
Terminal
May 24 07:54
hadoop@milind-VirtualBox: ~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce

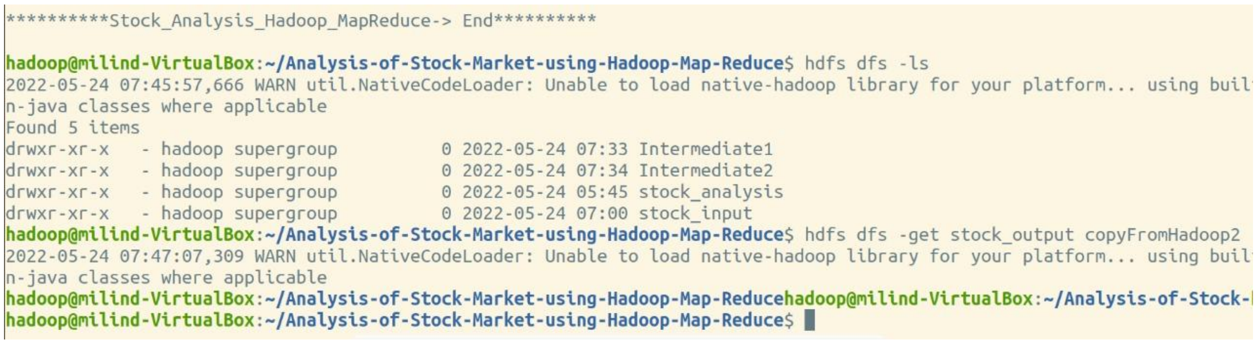
```
2022-05-24 07:18:13,003 INFO mapreduce.Job:  map 45% reduce 15%
2022-05-24 07:18:41,341 INFO mapreduce.Job:  map 46% reduce 15%
2022-05-24 07:18:43,355 INFO mapreduce.Job:  map 47% reduce 15%
2022-05-24 07:18:44,360 INFO mapreduce.Job:  map 47% reduce 16%
2022-05-24 07:19:10,659 INFO mapreduce.Job:  map 48% reduce 16%
2022-05-24 07:19:18,708 INFO mapreduce.Job:  map 49% reduce 16%
2022-05-24 07:19:39,941 INFO mapreduce.Job:  map 50% reduce 16%
2022-05-24 07:19:45,967 INFO mapreduce.Job:  map 50% reduce 17%
2022-05-24 07:19:46,985 INFO mapreduce.Job:  map 51% reduce 17%
2022-05-24 07:20:10,101 INFO mapreduce.Job:  map 52% reduce 17%
2022-05-24 07:20:28,461 INFO mapreduce.Job:  map 53% reduce 18%
2022-05-24 07:20:42,578 INFO mapreduce.Job:  map 54% reduce 18%
2022-05-24 07:20:56,715 INFO mapreduce.Job:  map 55% reduce 18%
2022-05-24 07:21:11,859 INFO mapreduce.Job:  map 56% reduce 18%
2022-05-24 07:21:17,925 INFO mapreduce.Job:  map 56% reduce 19%
2022-05-24 07:21:40,368 INFO mapreduce.Job:  map 57% reduce 19%
2022-05-24 07:21:41,450 INFO mapreduce.Job:  map 58% reduce 19%
2022-05-24 07:22:11,697 INFO mapreduce.Job:  map 59% reduce 19%
2022-05-24 07:22:12,700 INFO mapreduce.Job:  map 59% reduce 20%
2022-05-24 07:22:15,743 INFO mapreduce.Job:  map 60% reduce 20%
2022-05-24 07:22:42,995 INFO mapreduce.Job:  map 61% reduce 20%
2022-05-24 07:22:46,041 INFO mapreduce.Job:  map 62% reduce 20%
2022-05-24 07:22:49,060 INFO mapreduce.Job:  map 62% reduce 21%
2022-05-24 07:23:12,315 INFO mapreduce.Job:  map 63% reduce 21%
2022-05-24 07:23:26,451 INFO mapreduce.Job:  map 64% reduce 21%
2022-05-24 07:23:41,552 INFO mapreduce.Job:  map 65% reduce 21%
2022-05-24 07:23:44,567 INFO mapreduce.Job:  map 65% reduce 22%
2022-05-24 07:23:56,706 INFO mapreduce.Job:  map 66% reduce 22%
2022-05-24 07:24:10,765 INFO mapreduce.Job:  map 67% reduce 22%
2022-05-24 07:24:38,951 INFO mapreduce.Job:  map 68% reduce 22%
2022-05-24 07:24:39,963 INFO mapreduce.Job:  map 68% reduce 23%
2022-05-24 07:24:42,980 INFO mapreduce.Job:  map 69% reduce 23%
```

fig: Map Reduce jobs running in parallel manner facilitated by the distributed file system


Activities
Terminal
May 24 07:55
hadoop@milind-VirtualBox: ~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce

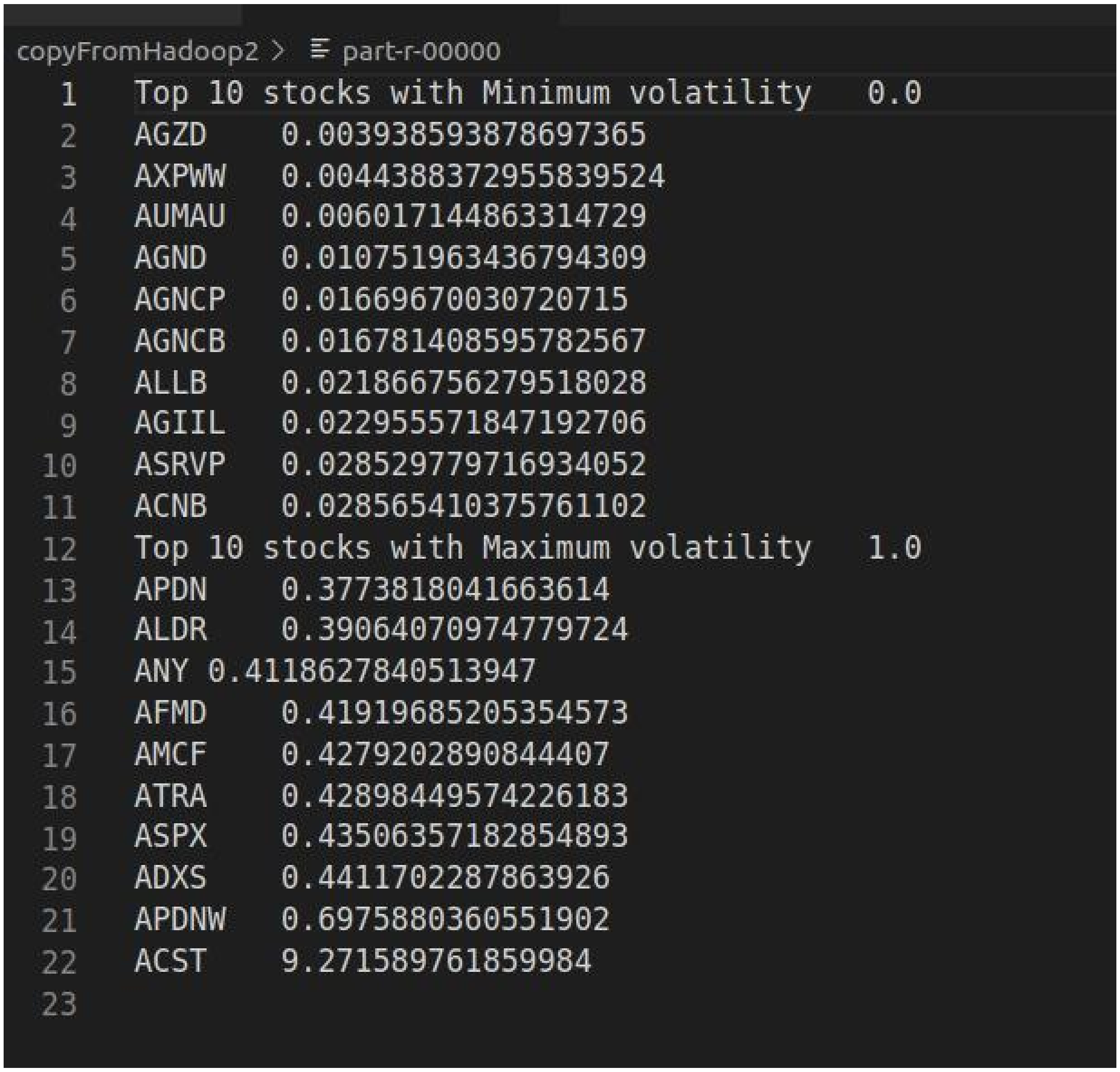
```
2022-05-24 07:34:01,922 INFO mapreduce.Job:  map 100% reduce 0%
2022-05-24 07:34:11,070 INFO mapreduce.Job:  map 100% reduce 100%
2022-05-24 07:34:12,241 INFO mapreduce.Job: Job job_1653355443556_0003 completed successfully
2022-05-24 07:34:12,460 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=232204
                FILE: Number of bytes written=936081
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=283287
                HDFS: Number of bytes written=6654
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=6070
                Total time spent by all reduces in occupied slots (ms)=6298
                Total time spent by all map tasks (ms)=6070
                Total time spent by all reduce tasks (ms)=6298
                Total vcore-milliseconds taken by all map tasks=6070
                Total vcore-milliseconds taken by all reduce tasks=6298
                Total megabyte-milliseconds taken by all map tasks=6215680
                Total megabyte-milliseconds taken by all reduce tasks=6449152
        Map-Reduce Framework
                Map input records=8494
                Map output records=8494
                Map output bytes=215210
```

fig: Completion of All tasks



```
**********Stock_Analysis_Hadoop_MapReduce-> End**********

hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce$ hdfs dfs -ls
2022-05-24 07:45:57,666 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
n-java classes where applicable
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2022-05-24 07:33 Intermediate1
drwxr-xr-x   - hadoop supergroup          0 2022-05-24 07:34 Intermediate2
drwxr-xr-x   - hadoop supergroup          0 2022-05-24 05:45 stock_analysis
drwxr-xr-x   - hadoop supergroup          0 2022-05-24 07:00 stock_input
hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce$ hdfs dfs -get stock_output copyFromHadoop2
2022-05-24 07:47:07,309 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
n-java classes where applicable
hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reducehadoop@milind-VirtualBox:~/Analysis-of-Stock-
hadoop@milind-VirtualBox:~/Analysis-of-Stock-Market-using-Hadoop-Map-Reduce$
```

fig: Commands to get the output back from the distributed file system onto our own file system



copyFromHadoop2 > part-r-00000

```
Top 10 stocks with Minimum volatility   0.0
AGZD    0.003938593878697365
AXPWW   0.004438837295583952 4
AUMAU   0.006017144863314729
AGND    0.010751963436794309
AGNCP   0.01669670030720715
AGNCB   0.016781408595782567
ALLB    0.021866756279518028
AGIIL   0.022955571847192706
ASRVP   0.028529779716934052
ACNB    0.028565410375761102
Top 10 stocks with Maximum volatility   1.0
APDN    0.3773818041663614
ALDR    0.39064070974779724
ANY 0.4118627840513947
AFMD    0.41919685205354573
AMCF    0.4279202890844407
ATRA    0.42898449574226183
ASPX    0.43506357182854893
ADXS    0.4411702287863926
APDNW   0.6975880360551902
ACST    9.271589761859984
```

fig: final result