Data Intensive Computing CSE 587

Volatility Analysis Report

Using Mapreduce to compute the volatility of stocks in NASDAQ

Prakash Natarajan | pn33@buffalo.edu

Spring 2015

Volatility Analysis Report

Programming Assignment 1

Problem Statement

1. Use Mapreduce to compute the volatility of stocks in NASDAQ
2. Evaluate the scalability of your implementation on different data sizes and number of nodes.

Description

In this assignment, you will use Mapreduce on a Hadoop environment at CCR to compute the monthly volatility of stocks. You are given daily data of 2970 stocks on NASDAQ market for 3 years from 01/01/2012 to 12/31/2014 (except holidays, otherwise called trading days). Imagine that you are a data analyst working for an investment company, your daily job is to analyse 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. You can get more details at http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:standard_deviation _volatility
Your data is 2970 CSV format files (Comma Separated). Each file contains the data for one stock using its symbol as the file name. A stock list file is also provided.

Data Format

For example, in the file **AAPL.csv**(Apple inc. stock), there are 7 columns, each row represents one day. The 5th column Close can be neglected(no use).

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;

Calculation

The calculation steps of Volatility is as follows:

- Calculate the average (mean) price for the number of periods or observations.
- Determine each period's deviation (close less average price).
- Square each period's deviation.
- Sum the squared deviations.
- Divide this sum by the number of observations.
- The standard deviation is then equal to the square root of that number.

Objective

- Find the top 10 stocks with Lowest (min) volatility
- Find the top 10 stocks with the Highest (max) volatility

Algorithm

I used mapreduce paradigm of Hadoop to serialise the calculation of volatility for each month and computed the top 10 and bottom values.

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

Roles of each Mapper and Reducer:

<u>Mapper1</u>

- splits the input data and options the date and close adjacent value.
- key - stock_name + month + year
- value - date + adjacent close value

<u>Reducer1</u>

- Since after the map step the values which have 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 Xi.

<u>Mapper2</u>

- Now we have to consolidate all the values obtained from the reducer with respect to company name.
- Key - Company Name
- Value - Xi

<u>Reducer2</u>

- 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

<u>Mapper3</u>:

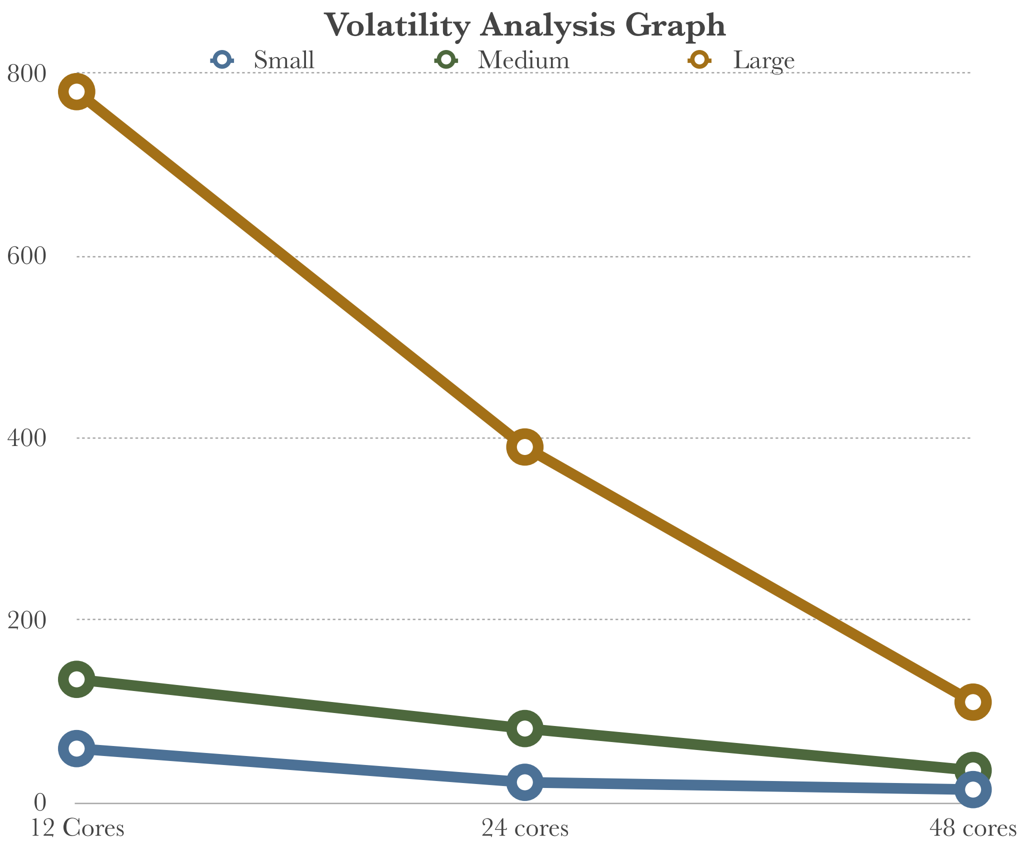
- All the companies are grouped together with a common key.
- Key - Common
- Value = Company Name + Volatility

<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

Results and Graph

Problem size	Time in 12 cores	Time in 24 cores	Time in 48 cores
Small	59:10	22:39	14:35
Medium	2:14:23	1:21:58	35:10
Large	13:19:27	6:30:40	1:49:51



Performance and Scalability

From the above table and graph we can see that the ***time required for Computation decreases as the number of cores increases.***

For small data, the time taken by 4 nodes decreases about one-fourth of the time taken by one node. Similarly for medium data too time decreases by one fourth of the time taken by the 1 node. For large data, one node took around 13 hours whereas 4 node took around 2 hours. This is one sixth of the time taken in 4 nodes.

Conclusion

We can see from the above graph, that map reduce programs perform better if we increase the cores of the computation. So, **scale out** is better than scale up.