CSE 587
DATA INTENSIVE COMPUTING
SPRING 2015
PROGRAMMING ASSIGNMENT #1 – REPORT

PRATIK PRAMOD CHAVAN
UBIT NAME: PCHAVAN2
UB PERSON #:- 5013-4114
CONTACT NO:- (716)-400-4194
EMAIL ID: pchavan2@buffalo.edu

1. IMPLEMENTATION:

The objective of this programming assignment is to find the volatility of stocks in NASDAQ. We are given with daily (trading days) data of 2970 stocks for 3 years from 01/01/2012 to 12/32/2014. The task is to compute volatility of each stock and then return top 10 stocks with highest volatility and lowest volatility.

We are given the formula to calculate monthly rate of return for return for each stock. Using that, we can calculate the volatility for that particular stock. We are given the formulae for calculating volatility and monthly rate of return.

Basically, to calculate the monthly rate of return we just need the Adjusted Close Price at the beginning and at the end of the day. This monthly rate of return value for each month is then used along with the average monthly rate of return in the formula to calculate the volatility.

So, in each .csv file for each stock the only entries of our interest are the ones at the start of the month and end of the month. In those entries also, the only fields of interest are the Date and the Adjusted Close Price. I have used date to create key and adjusted close price as value.

I am using Text as output format of mapper for both key and value and input and output format of reducer for both key and value. I am using company_name + month + year as a key. For each key there will be 2 values. One value will be a letter 'b' (to denote the beginning adjusted close price of the month) + the adjusted close price for that day. The other value will be a letter 'e' (to denote the end adjusted close price of the month) + the adjusted close price for that day. For the last <key,value> input pair to the map job which will definitely a beginning day for some month (possible an end day too if for example 10/31 is the day.), I am using getLength() method to calculate the size of the input split and then using key (LongWritable which denotes the byte offset of the input split till that point) and getLength() method on value to check if the record is the last record in the input split. If key + value's length = length of input split – 1, then it is the last input record to the mapper. So, that entry is processed accordingly.

Using this at a reducer, monthly rate of return will be calculated. (At reducer we have only the beginning and ending adjusted close prices of every month for every stock.) These monthly rates of return will be stored in an array which will be used to compute volatility once the next company data starts coming. (Next company data can be verified from the key which contains the company name). Then on cleanup function at the reducer (called once at the end of the reduce job) the volatility for the last month is calculated. (for the last record there won't be a next record to verify if the company is same). Then, the top ten and Bottom ten values of volatilities are calculated and returned along with the keys as company names.

2. EXPERIMENTS:

We are required to find the computing times for the 3 types of data sets (small, medium and large) each on 1,2 and 4 nodes (each with 12 cores) at CCR at University at Buffalo.

The objective here is to check the performance of the program and check its data scalability and node scalability.

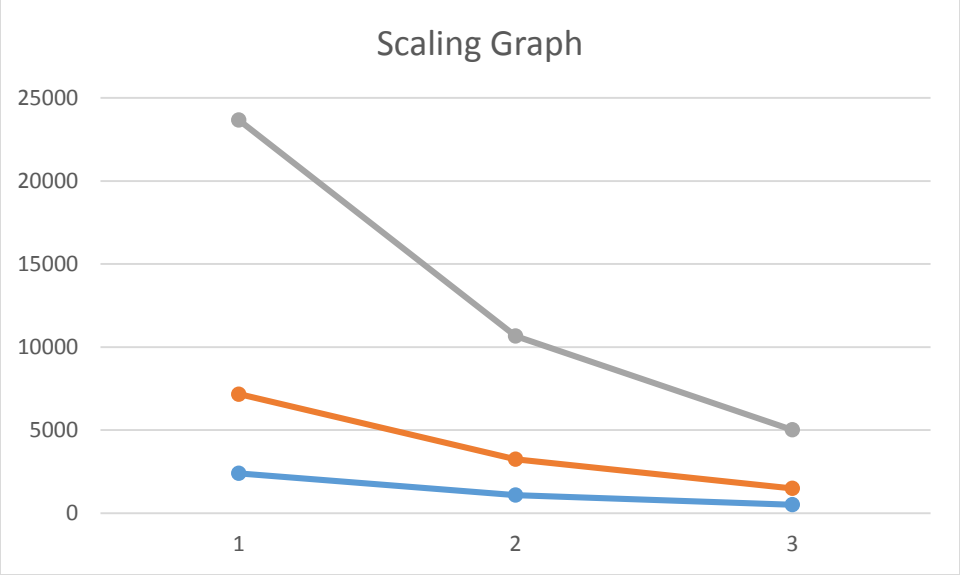
The following table shows the execution times which I got for all the cases. I have used the code in main function of driver class to compute the time required to execute the main function in seconds. I have referred the driver class code of Matrix Multiplication practice assignment (by Chao Feng) posted on UBLearns.

Problem Size	**Execution Time: 1 node (12 cores)**	**Execution Time: 2 nodes (24 cores)**	**Execution Time: 4 nodes (48 cores)**
Small	**2400 (40 min)**	**1085 (18.08 min)**	**508 (8.47 min)**
Medium	**7168 (119.47 min)**	**3245 (54.08 min)**	**1492 (24.87 min)**
Large	**23667 (394.45 min)**	**10661 (177.68 min)**	**5017 (83.61 min)**

3. DISCUSSION:

The graph below is the scaling graph which tells us how the scaling against number of nodes and the scaling against size of dataset is taking place.

X- axis has 3 values which represent number of nodes which are 1, 2 and 4. Blue line represents the small dataset, orange line represents medium dataset and purple line represents large dataset. Y-axis represents time in seconds taken to execute the job.



From the graph we can see that proper scaling in terms of both data scaling and node scaling is taking place.

Using more number of nodes for the same size of dataset is doing the job faster.

Also, the bigger the size of the job is, longer it takes to execute if we keep number of nodes same.

Executing small dataset with 4 nodes takes the least execution time where as large dataset with 1 node takes the maximum amount of time to execute.

In conclusion, I would like to say that, node scaling and data scaling are getting done properly. So, if the dataset size is bigger, more number of cores should be used to reduce the execution time. And, if the dataset is smaller, less number of cores should be used if we have to attain approximately similar execution time in all the cases.