CSE 487/587 Programming Assignment #1

Due Date: Feb 28, 2015

Name: Daniel Nazareth

UBIT Name: dnazaret

Email: dnazaret@buffalo.edu

UB Person #: 50134215

Problem Statement: (a) Use MapReduce to compute the volatility of stocks in NASDAQ

(b) Evaluate the scalability of your implementation on different data sizes and number of nodes.

Solution (a): **Map Reduce Implementation**

A brief walkthrough of the Map Reduce implementation/code is below:

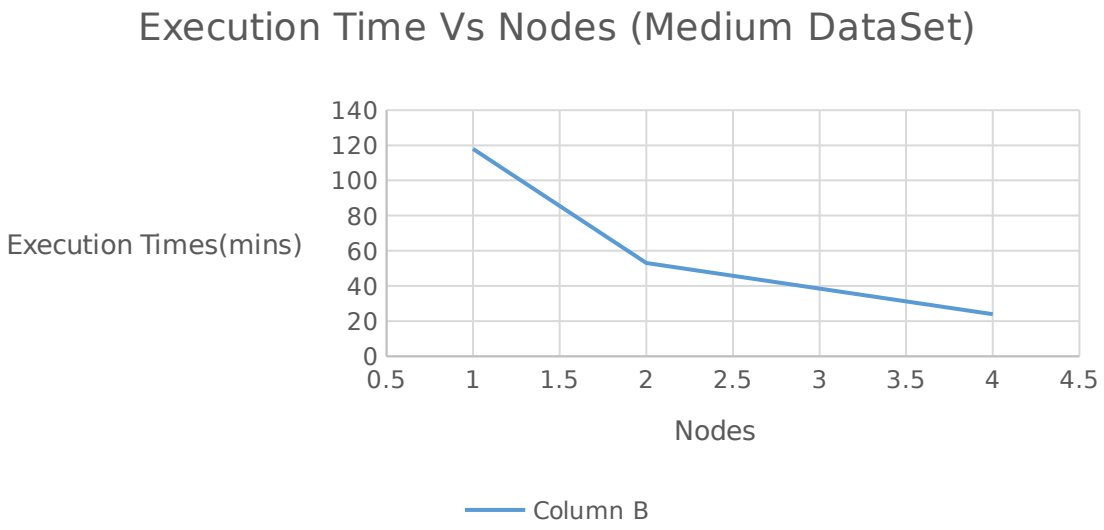
- Code begins by importing the standard hadoop implementation libraries from org.apache.hadoop as prescribed in the official apache hadoop 2.6.0 documentation.
- The mapper class and map function are then constructed. We use the standard TextInputFormat to read the individual csv files. Since no files exceed default split size of 64 MB, we can be reasonably confident that the files themselves will be used as split boundaries.
- We now define an important data structure, the *AdjClosePrice array list* which is used for storing the adjclose prices of each record. At a time, we store the adjcloseprices for exactly 1 month. Then we compute the rate of return(ROR) for that month and write a <key,value> pair that maps to <Stock,MonthlyROR>. After each write, the arrayList is cleared.Thus for a sample key, say AAPL.csv, we would have 36 values corresponding to AAPL's 36 months of data and monthly rates of return.
- We conclude the map method by writing the final months' rate of return during the cleanup phase of the mapping. This ensures that no boundary month is accidentally missed.
- We move on to the reducer which reads in a key and all its ROR's and calculates the volatility for the stock. Note that we **ignore** stocks which have no data or only 1 month of data/ stocks for which volatility is zero. We now

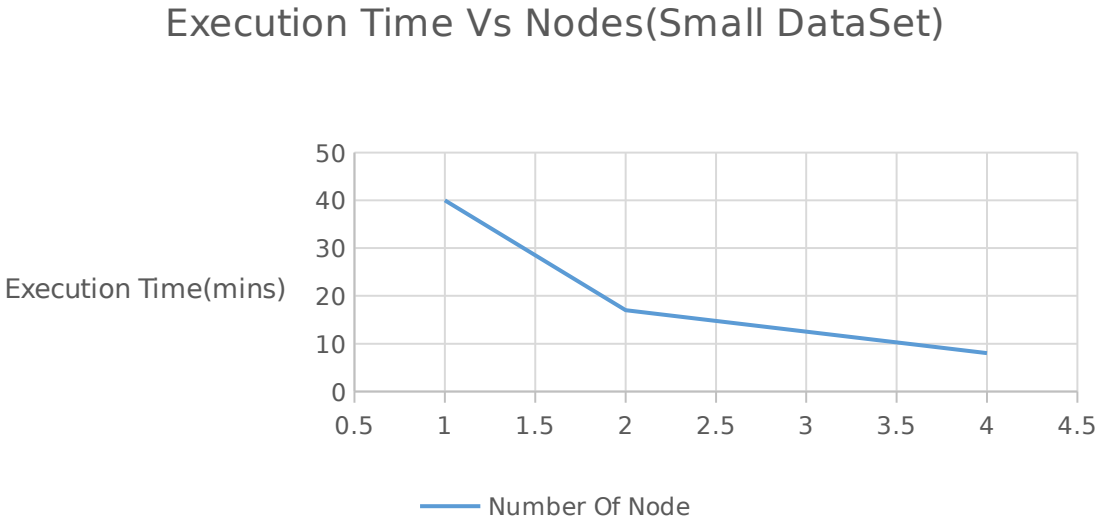
write the key(stock name) and volatility to two key and volatility array lists for sorting purposes.

- Now we sort the two array lists simultaneously in ascending order using an easy-to-implement O(n^2) sorting algorithm. Although this impacts our efficiency marginally, the slowdown is insignificant compared to the processing power at our disposal.
- Finally we write the first and last 10 elements of the arrays as keys and values to the output file. After sorting, the first 10 are stocks with least volatility and last 10 are ones with highest.

Solution (b): The execution times for a variety of dataset sizes and allocated nodes are indicated below in tabular and graphical fashion. **It is observed that execution times scale in an approximately linear manner as no of nodes and cores allocated to the task increases**. This is depicted in the table and graph that follows. For a doubling of nodes/cores allocated, a reduction in execution by a factor of >=2 is also observed, that is execution time at least halves.

Problem Size	1 node/12 core Exec Time(sec)	2 node/24 core Exec Time(sec)	4 node/48 core Exec Time(sec)
Small	2395sec/~40 mins	1037sec/~17mins	449sec/~8mins
Medium	7083sec~118mins	3176sec/~53 mins	1444sec/~24mins
Large	Pending(could not complete)	9847sec/~164 mins	4573sec/~76mins




Execution Time Vs Nodes(Small DataSet)
Execution Time(mins)
50
40
30
20
10
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Number Of Node