CSE587- DATA INTENSIVE COMPUTING

UNIVERSITY AT BUFFALO, THE STATE UNIVERSITY OF NEW YORK

PROGRAMMING ASSIGNMENT 1

STOCK VOLATILITY COMPUTATION USING HADOOP MAPREDUCE

Submitted By:

Aravind Kumar Ramesh

UB PERSON#: 5013-2610

Email: aravindk(at)buffalo(dot)edu

Introduction

This project computes stock volatility of NASDAQ stocks using Hadoop-MapReduce. For the purpose of this project we have made use of stock information for the last 3 years starting from 01/01/2012 to 12/31/2014 for about 2970 stocks from NASDAQ. The input dataset has been provided in the csv (comma seperated value) format. In order to better understand the Node and Data scaling aspect of Hadoop-Mapreduce, we have three file size corresponding to small, medium and large dataset. The smallest of the dataset contains approximately 88MB of data while the largest dataset contains about 1GB of data. Varying the data size with corresponding number of cores will help us understand the efficiency of both the MapReduce code and the Hadoop System. The aim of this project is to accomplish this and to compute the top 10 stocks with lowest and maximum volatility by applying the suitable volatility computation formula (See Appendix A).

What is Volatility ?

Volatility (or) Stock volatility Index is often the measure of the relative performance of a stock. The popular site investopedia defines volatility as “the amount of uncertainty or risk about the size of changes in a security's value. A higher volatility means that a security's value can potentially be spread out over a larger range of values. This means that the price of the security can change dramatically over a short time period in either direction. A lower volatility means that a security's value does not fluctuate dramatically, but changes in value at a steady pace over a period of time.”(http://www.investopedia.com/terms/v/volatility.asp)

The Hadoop Framework

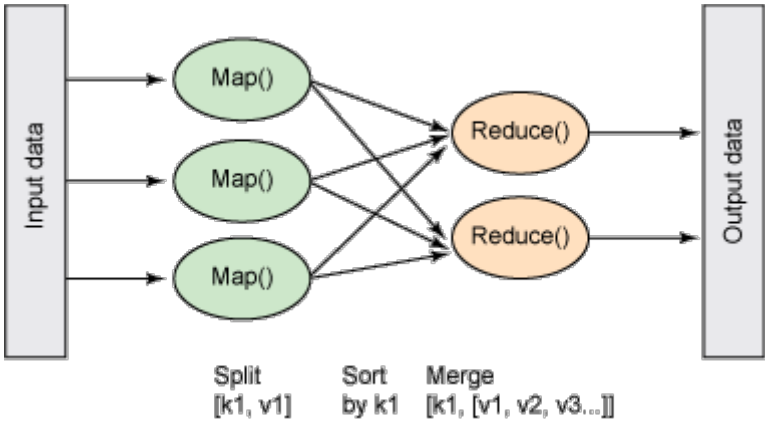
Hadoop is an opensource java framework that brings “compute as physically close to data as possible” paradigm. Hadoop basically consists of two parts: Hadoop Distrubuted File System (HDFS) and Hadoop MapReduce.

The Hadoop Distributed file system will take a file and split it into many small 'chunks', ut then distributes and stores these chnks across many servers. These chunks are also replicated to ensure redundency (mainly to handle fault tolerence). Storing data acorss multiple nodes in this way will boost performance and potentially save computing cost.

The MapReduce provides a programming framework to work on top of HDFS. After HDFS has distributed the data to many different servers MapReduce sends a fragment of a program to each server to execute. In other words, MapReduce is a framework that enables you to write an application that process vast amounts of data in parallel by sharing the work to be completed out to a large number of servers (clusters)

The MapReduce Design

The MapReduce framework draws its inspiration from functional programming but is not equivalent to it. This framework thereby allows for problems to be parallelized. The MapReduce framework can be summerized as shown in the figure.



MapReduce Framework (Image taken from http://www.ibm.com/developerworks/cloud/library/clopenstack-deployhadoop/figure3.gif)

The Input data from the HDFS is split and accordingly given to the Mapper function. The Mapper function performs the split function and outputs the key,value pair [k1, v1]. The results from the Mappers are then shuffled and sorted by hadoop (which essentially sorts it according to the keys). The result arrive at the reducer which merge them by creating a list of elements corresponding to the same key values.

ALGORITHM

For the computation of stock volatility, we have divided the job into three phases. In the first phase of MapReduce, we,

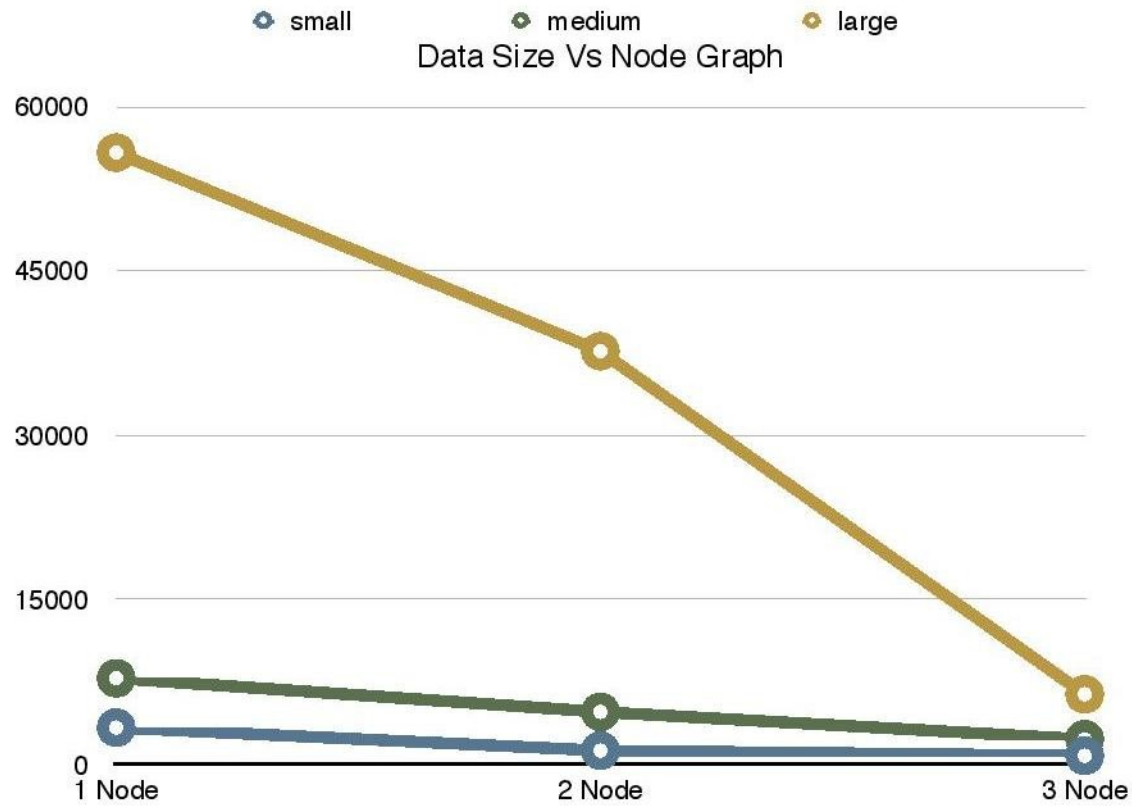
- Extract the input fields from the csv file and pass it to the Mapper 1
- The Mapper 1 parses the file line by line and extracts the date and closing balance values.
- We set the output key of the Mapper to be of the format KEY= STOCK_NAME YEAR MONTH and the output value to be VALUE= DATE CLOSING BALANCE
- In the shuffle and sort phase, the output of the Mapper will be sorted according to the key values and the outputs will be sent to the respective reducer.
- The Reducers here compute the Monthly rate of return (xi) by finding the minimum and maximum adjusted closing balance of every month.
- The output from the reducer are passed to the Mapper 2 of the next phase.
- The Mapper 2 now formats the key and value pair such as the KEY=STOCK_NAME and VALUE= MONTHLY CLOSING BALANCE (Computed in the previous step)
- The Reducer 2 now computes the volatility by applying the formula (discussed in appendix A)
- The output of the reducer now is passed to Mapper 3 which passes the input as output directly to the final reducer.
- At final reducer we sort the stock prices and their volatility and accordingly display the Top 10 most and least volatile stocks.

RESULTS

We conducted various experiments in order to better understand the relationship between the execution times as the data size is increased for different cluster size. For the purpose of this experiment, we ran our code on 1 node (12 core) , 2 node (24 core) and 4 node (48 cores). The result of the experiements are as shown below.

TIME TAKEN FOR EXECUTION (In Seconds)

NO. OF NODES/ SIZE OF DATA	1 NODE/ 12 CORES	2 NODES/ 24 CORES	4 NODES / 48 CORES
SMALL	***3357***	***1262***	***791***
MEDIUM	***7855***	***4783***	***2291***
LARGE	***55767***	***37662***	***6392***



Graph of Execution time for Node Size Vs Data Size

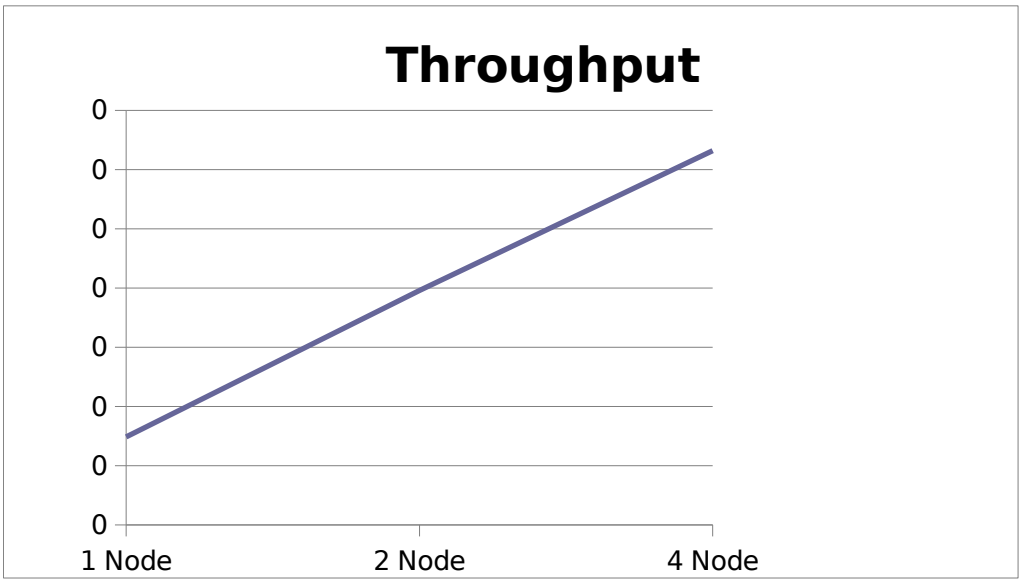
From the graph we observe that the execution time for small and medium data size on 1, 2 and 4 node clusters are all comparable and run fast . While running on a 4 core machine might seem like the best idea (due to lowest execution time) it may not be a good decision ideally as the rate of change of execution time for small and medium datasizes do not vary much with the cluster size. Hence using 4 core cluster for small datasizes may infact mean overutilizing of resources which could increase the net effective computing cost.

However, we notice that it is impractical to execute a large datasize in a 1 core machine as the execution time is extremely high. In such cases using 4 core cluster might be the most intuitive choice.

THROUGHPUT VALUES

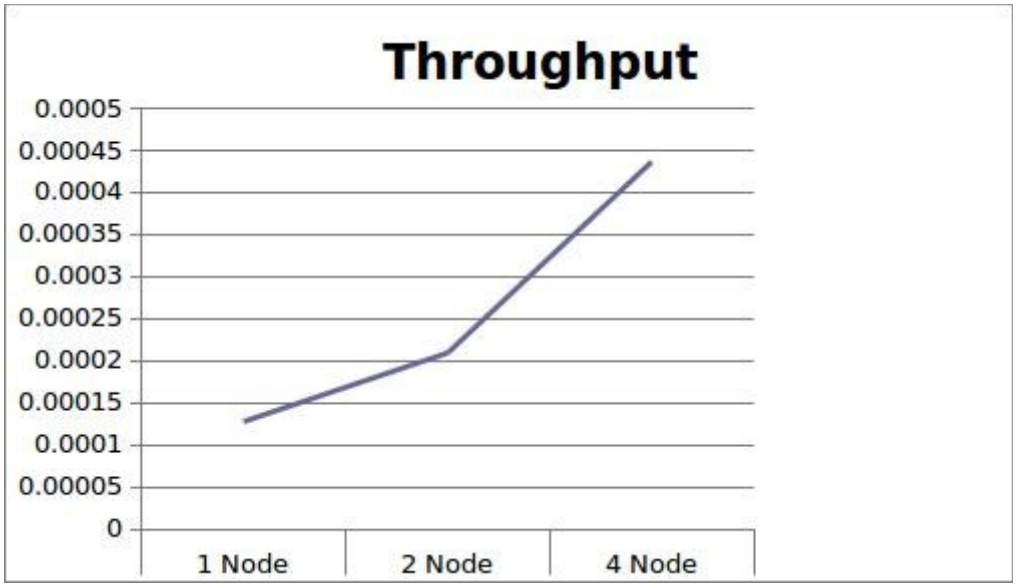
NO. OF NODES/ SIZE OF DATA	1 NODE/ 12 CORES	2 NODES/ 24 CORES	4 NODES / 48 CORES
SMALL	0.000297885	0.000792393	0.001264223
MEDIUM	0.000127307	0.000209074	0.000436491
LARGE	0.000017932	0.000026552	0.000156446

Comparing the throughput values against the Node Size and Data size would give us a better idea to base our choice of cluster for execution in Hadoop-MapReduce. We would be interested in infering optimum node selection for Datasize such that the execution is reasonably fast with the overall computing cost in mind. (It makes no sense to compute small datasize on a 4 cluster machine when the same performance can be obtained by running the job on a 2 core cluster).



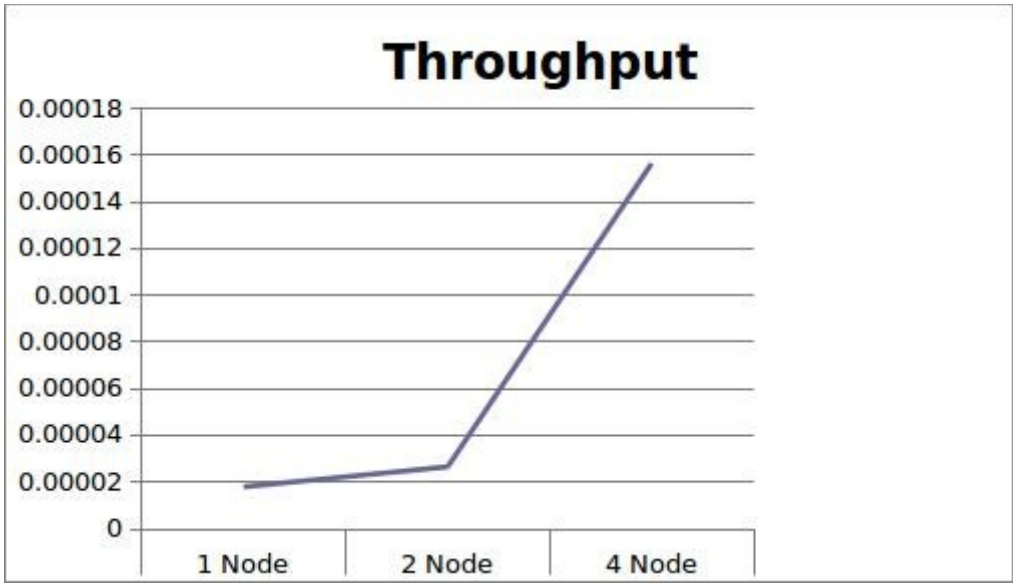
The Node Size Vs Throughput for Small Data Set

Throughput values increase linearly with increase in node size for small datasize. The throughput is maximum for 4 node cluster and minimum for 1 Node cluster. However we see a constant slope (rate of change is constant) throughtout indicating that not much can be benefited by choosing 4 nodes for small dataset.



The Node Size Vs Throughput for Medium Data Set

Throughput values in the Medium data size shows a slight bend in the throughput value. In such scenarious it would make intuitive sense to choose Nodes corresponding to the region of bend as our optimim cluster size. In this case a cluster size of 2 or more would be advisable for a medium datasize. We observe that the rate of change in throughput between 2 Node and 4 Node is almost linear. Hence reasonable performance can be expected by choosing 2 Node cluster.



The Node Size Vs Throughput for Large Data Set

This graph has a steep bend suggested that degraded performance is obtained when small cluster sizes are used for large data set. Also we notice that the rate of change in throughput between 2 nodes and 4 nodes is extremely high. Hence it would be a wise choice to choose 4 Node clusters in this case.

With these results explained, I strongly believe that the program is quite efficient. The running time of the code was measured by running the code on the local system with small and medium data size (without the HDFS). The code performed reasonable fast, executing the small dataset in the order of 120 seconds and about 500 seconds for the medium dataset. When the same program is run on the cluster, the HDFS splits the input data and replicates it into 3 copies to maintain redundency in case of failure. Thus most of the time is spent on the Name Node trying to communicate with the Data Node to reach a chunk. Thus under these constraints, the performance of the system is blazingly fast considering this complexity. This code will give better performance by increasing the number of node clusters, but cranking up the Node number would not be a viable solution as the HDFS process of Name Node communicating with the Data Node will now be more complex with more participating nodes than in the previous cases with less number of nodes. . Thus a tradeoff needs to be established and from the experiements performed so far, 2 Nodes for computing small and medium datasize and 4 Nodes for large datasize would be the reasonable choice.

APPENDIX A

The Stock Volatility Computation can be done as follows:

Xi = Monthly Rate of Return = (Month end adjusted close price – Month beginning adjusted close price) / (Month beginning adjusted close price)

Volatility = $$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}\ ,\quad \overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

Where N is the number of months. N is set to 36 (months), ending close price is the close price of the last trading day in each month, beginning close price is the close price of the first trading day in each month.