CSE587 : Data Intensive Computing (Spring 2015)

Project 1: Map Reduce on Hadoop

Naveen Narayan

UB ID : nnarayan

UB Person # : 50134647

Aim :

The aim of this project is to use principles of Map Reduce to calculate volatility for given stocks. The application is built on Hadoop. The application is run on data sets of varying size and also on varying number of nodes(on CCR) to compare and analyze performance on these factors.

Introduction :

MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation.

Apache Hadoop is a set of algorithms (an open-source software framework written in Java) for distributed storage and distributed processing of very large data sets (Big Data) on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.

The core of Apache Hadoop consists of a storage part (Hadoop Distributed File System (HDFS)) and a processing part (MapReduce). Hadoop splits files into large blocks (default 64MB or 128MB) and distributes the blocks amongst the nodes in the cluster.

To process the data, Hadoop Map/Reduce transfers code (specifically Jar files) to nodes that have the required data, which the nodes then process in parallel. This approach takes advantage of data locality to allow the data to be processed faster and more efficiently via distributed processing than by using a more conventional supercomputer architecture that relies on a parallel file system where computation and data are connected via high-speed networking.

Center for Computational Research(CCR) provides high performance computing resources to the University at Buffalo. Users run programs on the cluster(a collection of linux computers, a shared file system and a job scheduler) as jobs submitted to queues.

Model:

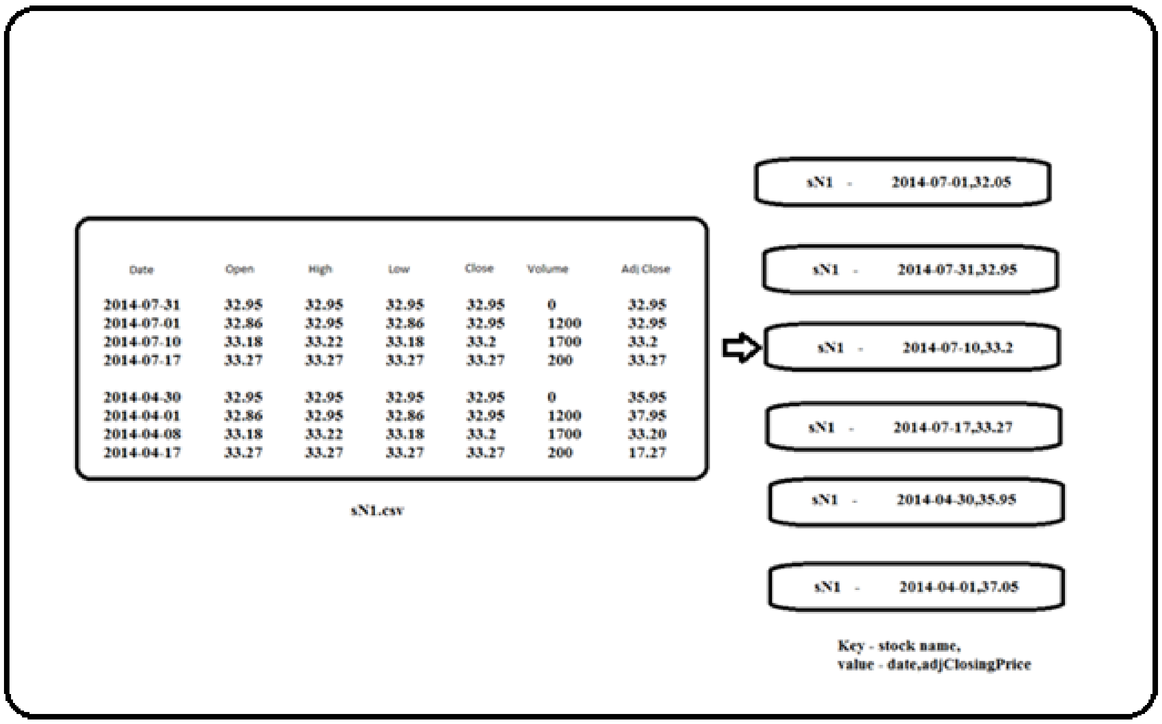
Stock data is provided in .csv files(<stockName.csv>). Each line in file corresponds to a day's trading information for that stock.

The entire process is divided into 2 MapReduce tasks –

a) Calculating the volatility for a given stock
b) Sorting stocks based on volatility

a) Calculating the volatility for a given stock -

Map phase : The input file name is extracted and is assigned as the key value. Each line is read from the file. Date and Adj close column's values are concatenated and set as the value.

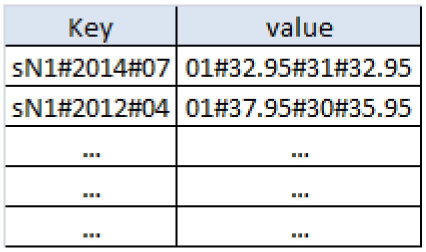


Date	Open	High	Low	Close	Volume	Adj Close
2014-07-31	32.95	32.95	32.95	32.95	0	32.95
2014-07-01	32.86	32.95	32.86	32.95	1200	32.95
2014-07-10	33.18	33.22	33.18	33.2	1700	33.2
2014-07-17	33.27	33.27	33.27	33.27	200	33.27
2014-04-30	32.95	32.95	32.95	32.95	0	35.95
2014-04-01	32.86	32.95	32.86	32.95	1200	37.95
2014-04-08	33.18	33.22	33.18	33.2	1700	33.20
2014-04-17	33.27	33.27	33.27	33.27	200	17.27

Fig 1 – Map phase for calculating volatility

Reducer phase – The output from the Map phase is input to Reducer phase. A HashMap is maintained to store the adjusted closing price for the beginning and ending days of the month.

The key for the HashMap is '<stockName>#<year>#<month>'. The format of the corresponding value is '<begDayOfMonth>#<acpForBegDay>#<cloDayOfMonth>#<acpForCloDay>'.



Key	value
sN1#2014#07	01#32.95#31#32.95
sN1#2012#04	01#37.95#30#35.95
...	...
...	...
...	...

Fig 2 – HashMap used to store adj closing price for each month

Once all the input key and value pairs are read, calculate xi, xBar and volatility.

For each entry in the HashMap, xi values are calculated for each month using the formula –

(Month end adjusted close price – Month beginning adjusted close price)

xi = --

(Monthly beginning adjusted close price)

xBar (mean of all xi values) is calculated using the below formula –

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

N – is the number of months for which we have trading data for a given stock (HashMap's size).

Volatility of the stock is calculated as –

$$\text{volatility} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

The output of the reducer phase is a key value pair, where the key is stock name and value is the volatility.

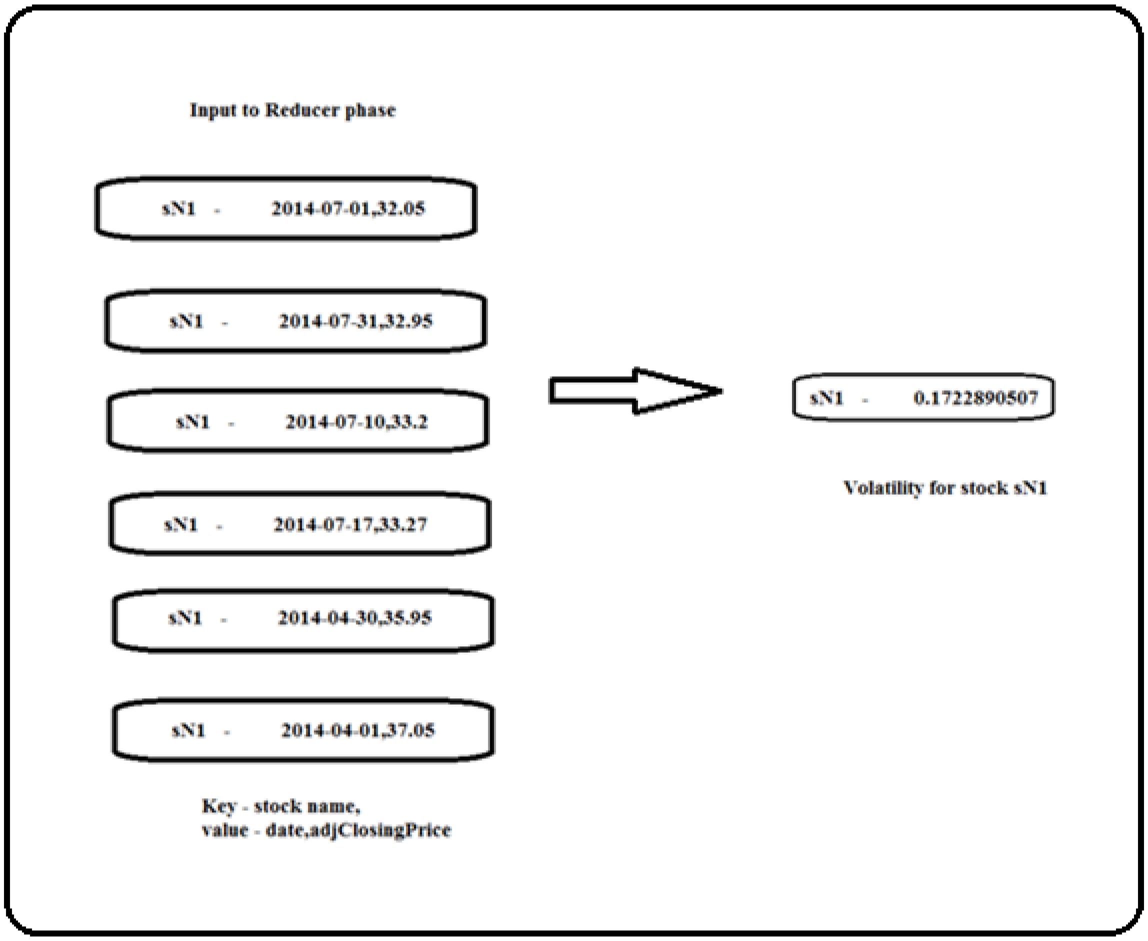


Fig 3 – Reducer phase for calculating volatility

b) <u>Sorting stocks based on volatility</u>

Map Phase - The output of the first MapReduce task is input to the second MapReduce task. Volatility of all the stocks are assigned to a single key in this phase and output to the Reducer.

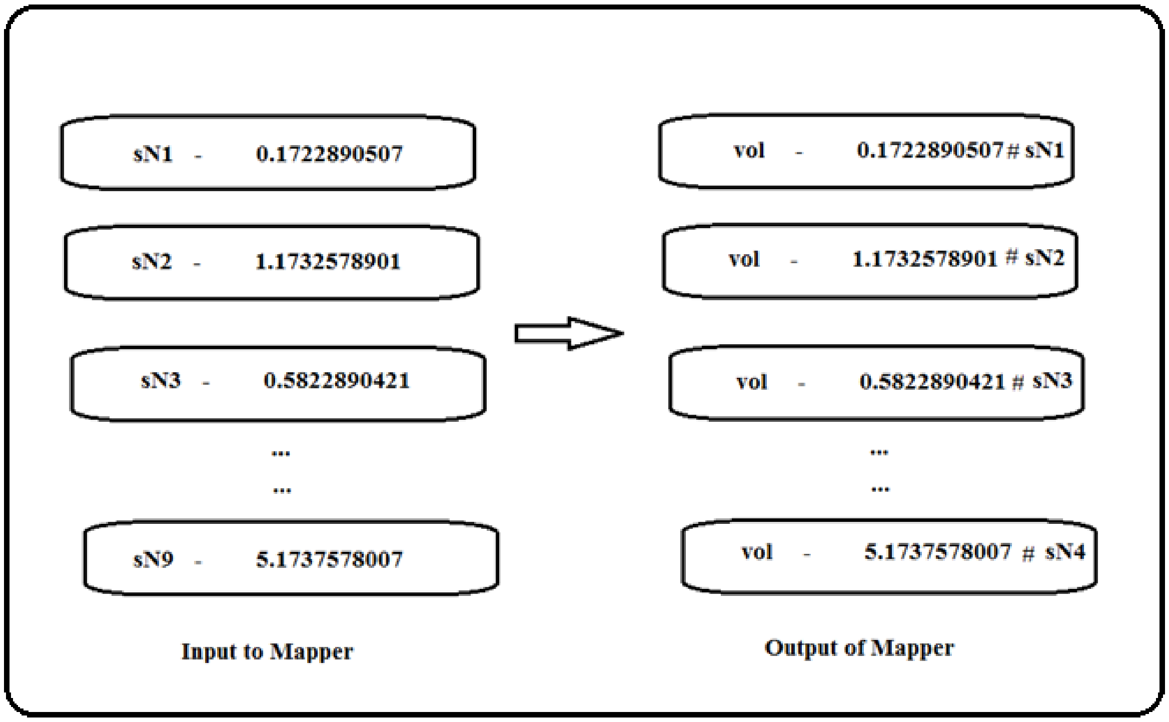


Fig 4 – Map phase for sorting stocks based on volatility

Reducer phase – Each key value pair is parsed and the values are put into a TreeMap (which sorts the stocks based on volatility). The top 10 and last 10 stocks are retrieved from the TreeMap and written to the context.

Implementation :

Launcher.java – Job instances are instantiated. Mapper and Reducer classes are set for each job.

Phase1CalcVolatility.java – Reads .csv data files. Performs the first Map Reducer task and writes context to an intermediate file.

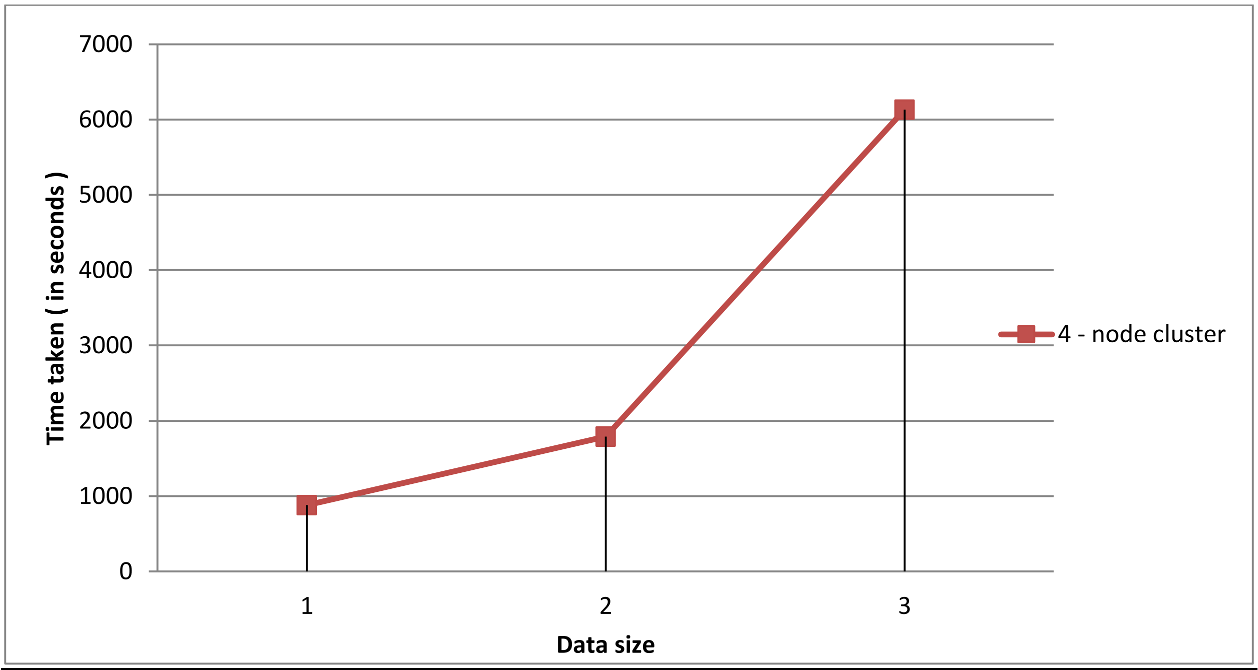
Phase2StockSort.java – Reads the intermediate file generated from Phase1CalcVolatility and writes the sorted list to the output folder.

Observations :

Nodes ======== Problem size	1 - core	2 - cores	4 - cores
Small	2917	1317	879
Medium	10,378	4325	1789
Large	31,366	14,123	6134

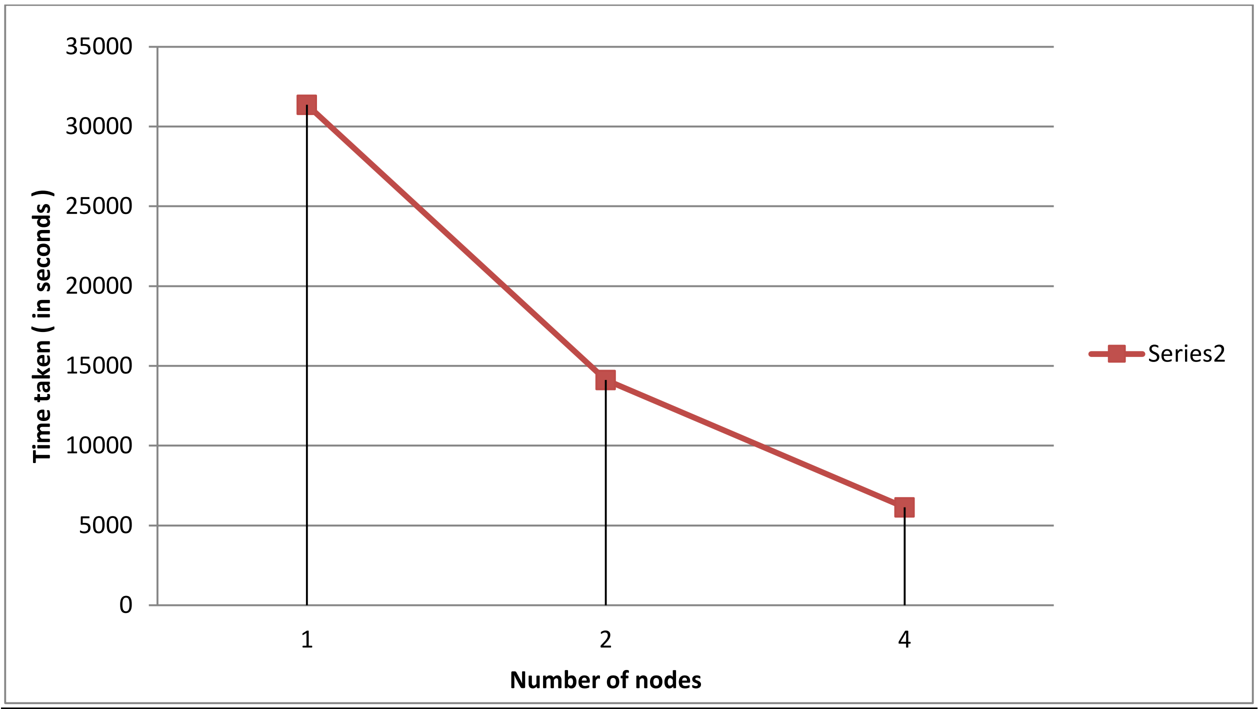
Fig 5 : Performance on varying number of nodes and data size

For a given number of nodes, as the data set size increases, the execution time increases.



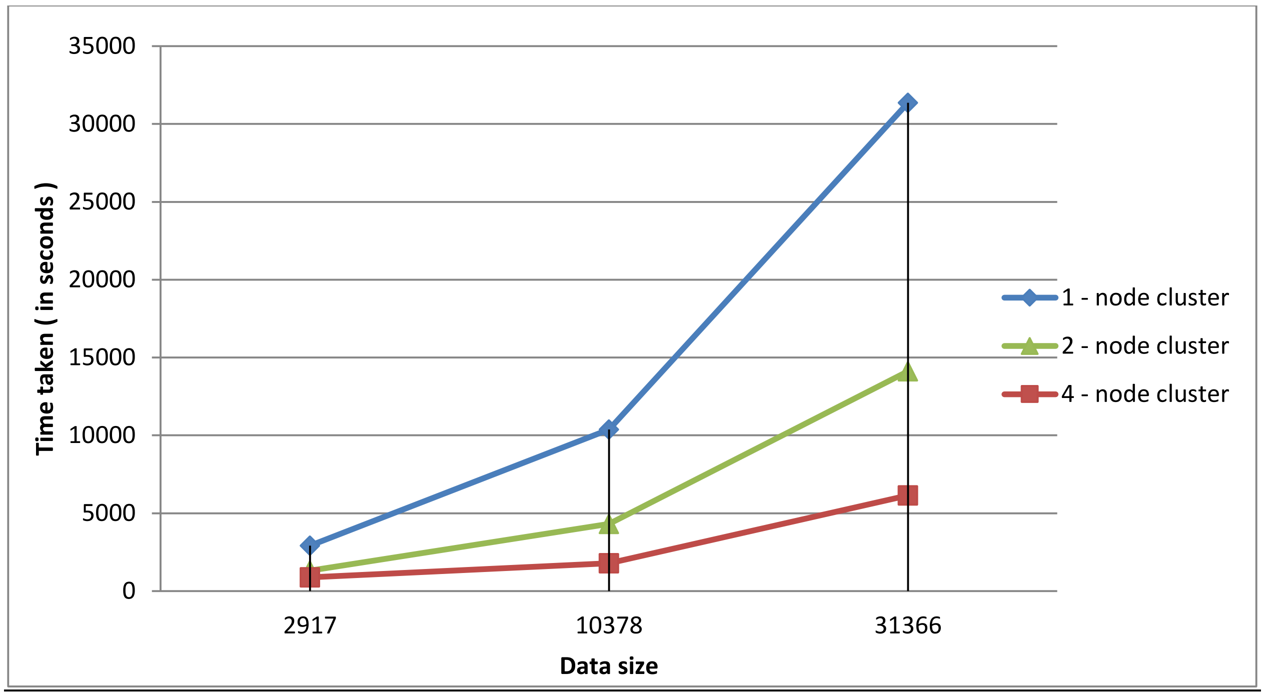
Graph plot 1 - Performance on varying data sizes for a fixed number of nodes

For a data set of fixed size, as the number of nodes increases the execution time decreases.



Graph plot 2 - Performance on varying number of nodes for a fixed data set

The difference in execution times is more obvious in the large data set than in the small data set as the actual time taken to process the data files in the small data set is less. Most of the execution time is taken up in setting up nodes, communication between the nodes and other house keeping procedures.



Graph plot 3 - Performance based on data size and number of nodes

Time taken on local installation of Hadoop – for small data set – 88 seconds.

Although a single machine outperforms a 4 node cluster (comprising 48 cores), this vast difference in execution times is mainly attributed to the overhead in CCR in assigning of the tasks, communication between nodes and other house keeping procedures.

As the data set size increases, we will observe a decreased difference in execution times. Hadoop is mainly used in the processing of big data (data in the order of peta bytes).

Softwares used :

Hadoop 2.5.2

References :

http://en.wikipedia.org/wiki/MapReduce

http://en.wikipedia.org/wiki/Apache_Hadoop

Lecture - Intro-CCR-Spring-2015.pdf by Cynthia Cornelius