CSE587 : Data Intensive Computing (Spring 2015)

Project 4 : HBase

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 using HBase. 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 :

HBase is an open source, non-relational, distributed database modeled after Google's BigTable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed Filesystem), providing BigTable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API. In the parlance of Eric Brewer's CAP theorem, HBase is a CP type system.

Model:

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

Loading data into tables in HBase – The file name along with Date and Adj.closingPrice columns are loaded to a table.

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.

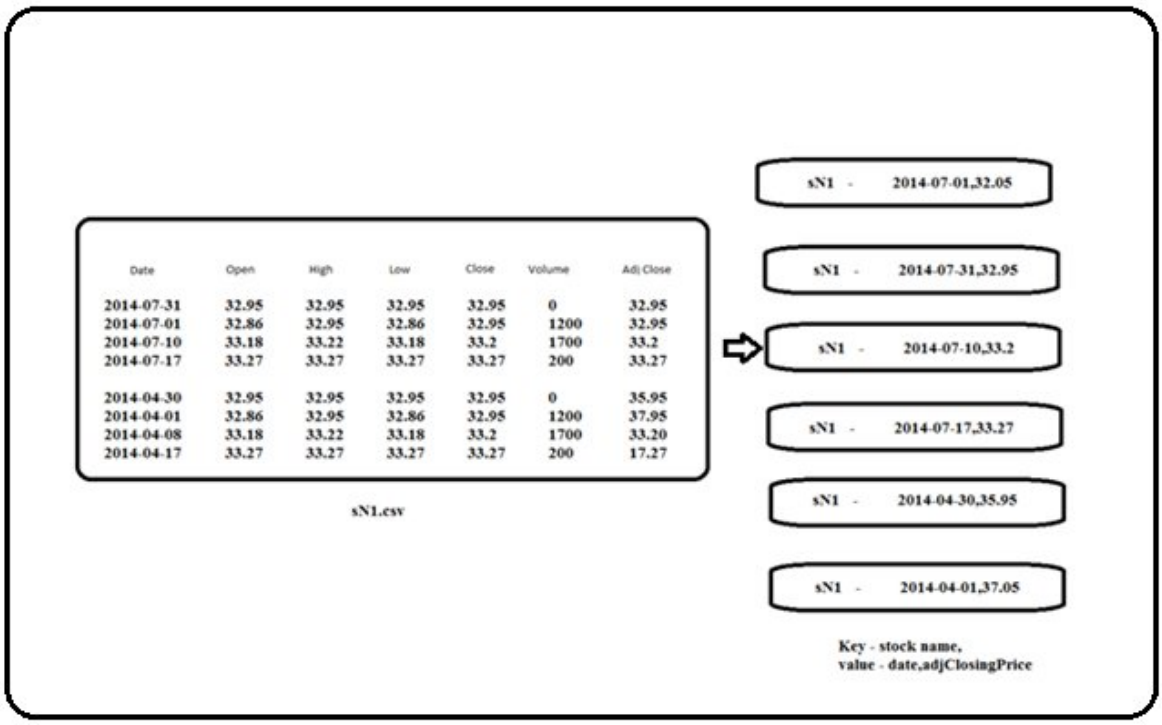


Fig 1 – Map phase for calculating volatility

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 –

$$x_i = \frac{\text{(Month end adjusted close price – Month beginning adjusted close price)}}{\text{(Monthly beginning adjusted close price)}}$$

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.

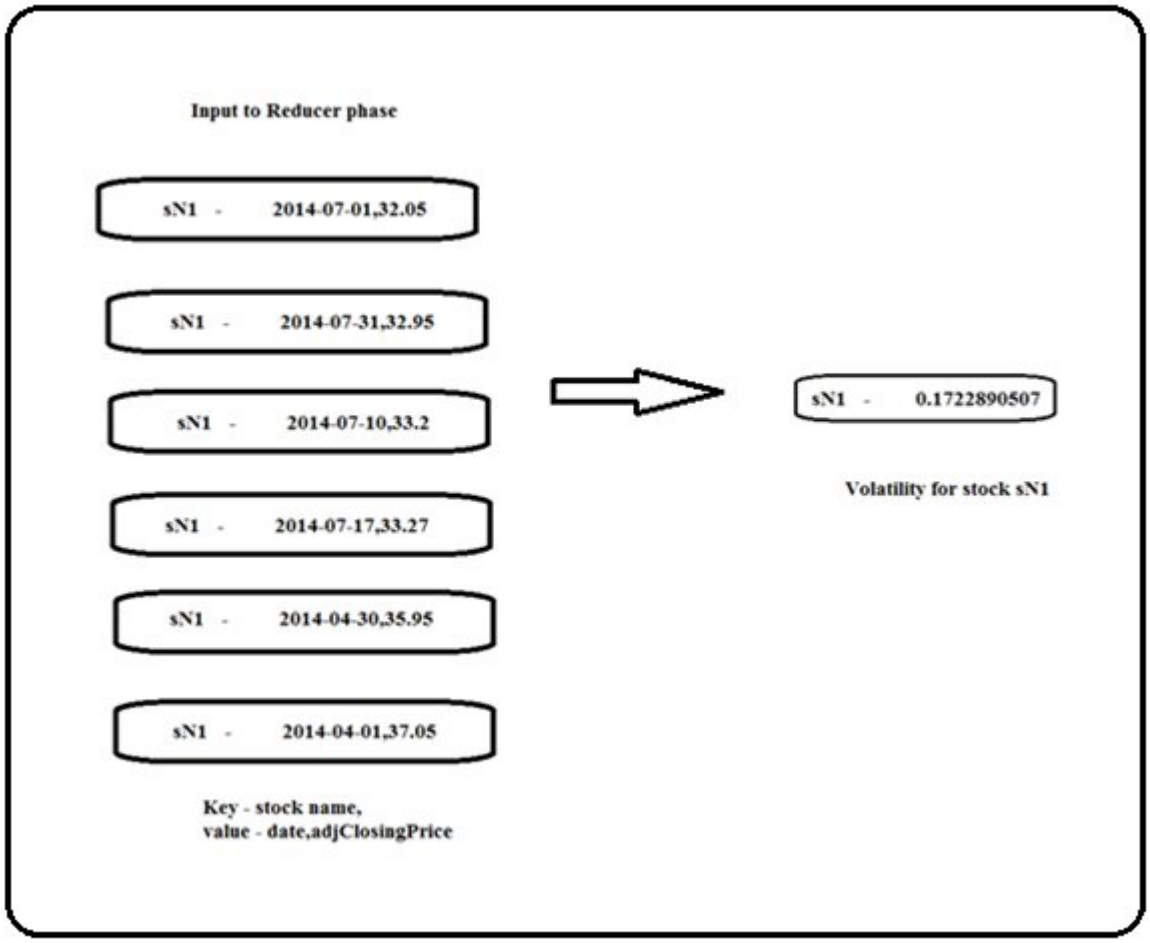


Fig 3 – Reducer phase for calculating volatility

b) Sorting stocks based on volatility

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.

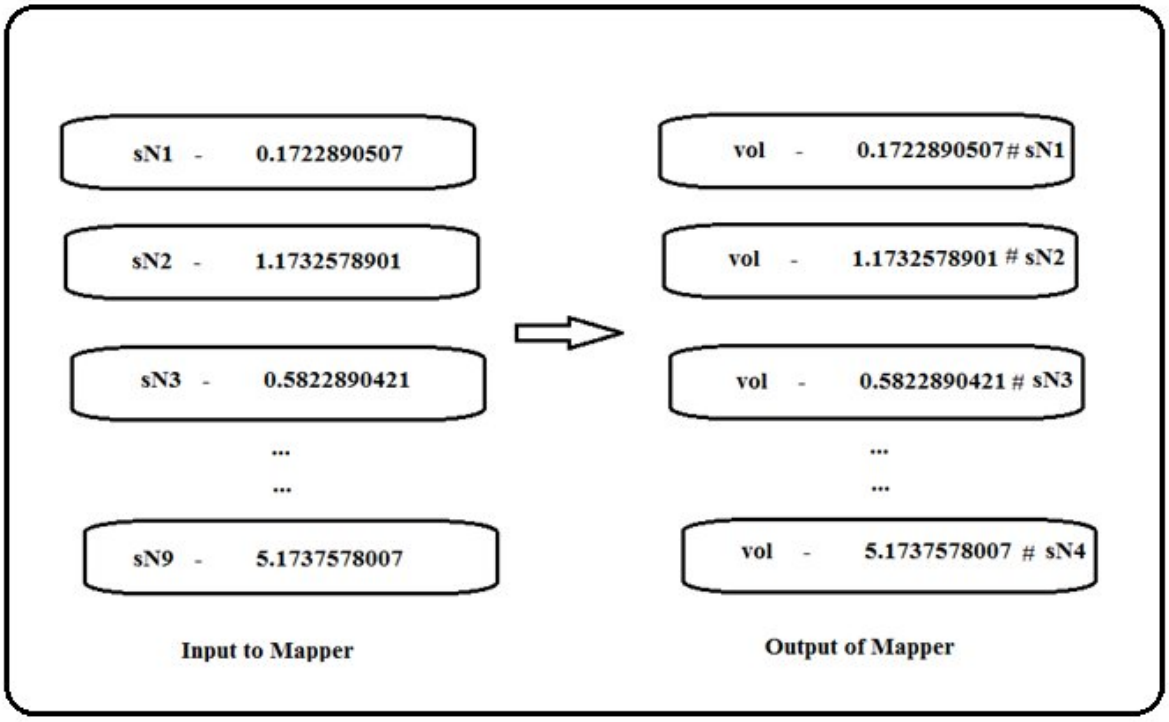


Fig 4 – Map phase for sorting stocks based on volatility

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 :

Main.java – Job instances are instantiated. Mapper and Reducer classes are set for each job. Tables ('raw' – to hold the raw data, 'target1Table' – intermediate table to hold result of first Map-Reduce phase & 'resultsTable' – to hold the first 10 and last 10 stocks based on volatility) are also created.

Job1.java – Raw data from the dataset is loaded to a HBase table.

Phase1.java – Reads raw data from 'raw' table. Performs the first Map Reducer task and writes context to 'target1Table', an intermediate table.

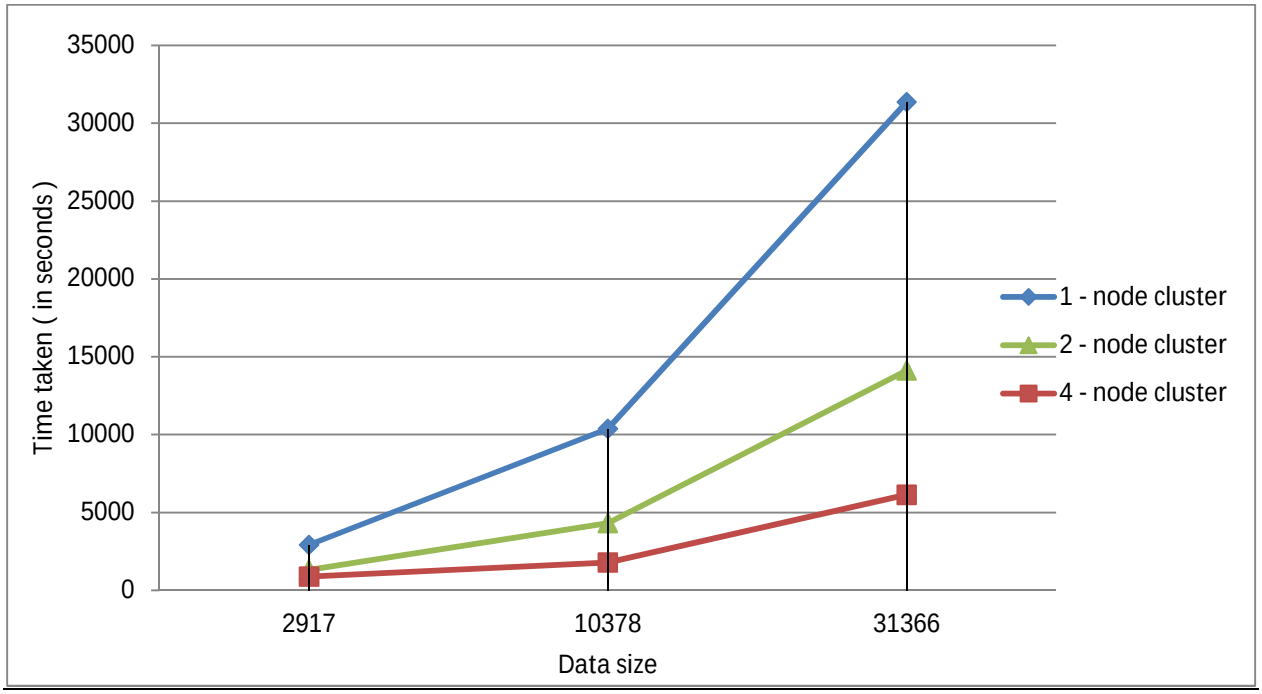
Phase2.java – Reads the intermediate file generated from Phase1 and writes the sorted lists to the 'resultsTable' table.

Observations :

Map Reduce -

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 1 : Performance on varying number of nodes and data size

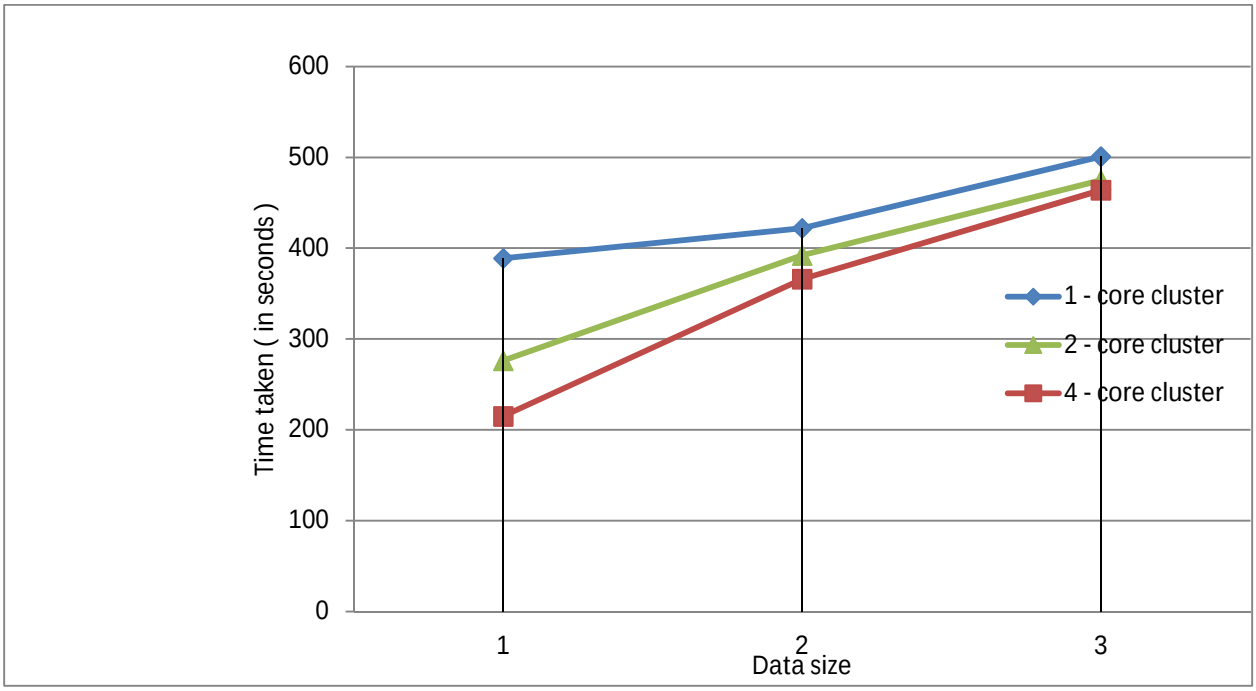


Graph plot 1 - Performance based on data size and number of nodes

Pig -

Nodes ======== Problem size	1 - core	2 - cores	4 - cores
Small	389	276	215
Medium	422	392	366
Large	501	475	464

Fig 2 : Pig - performance on varying number of nodes and data size

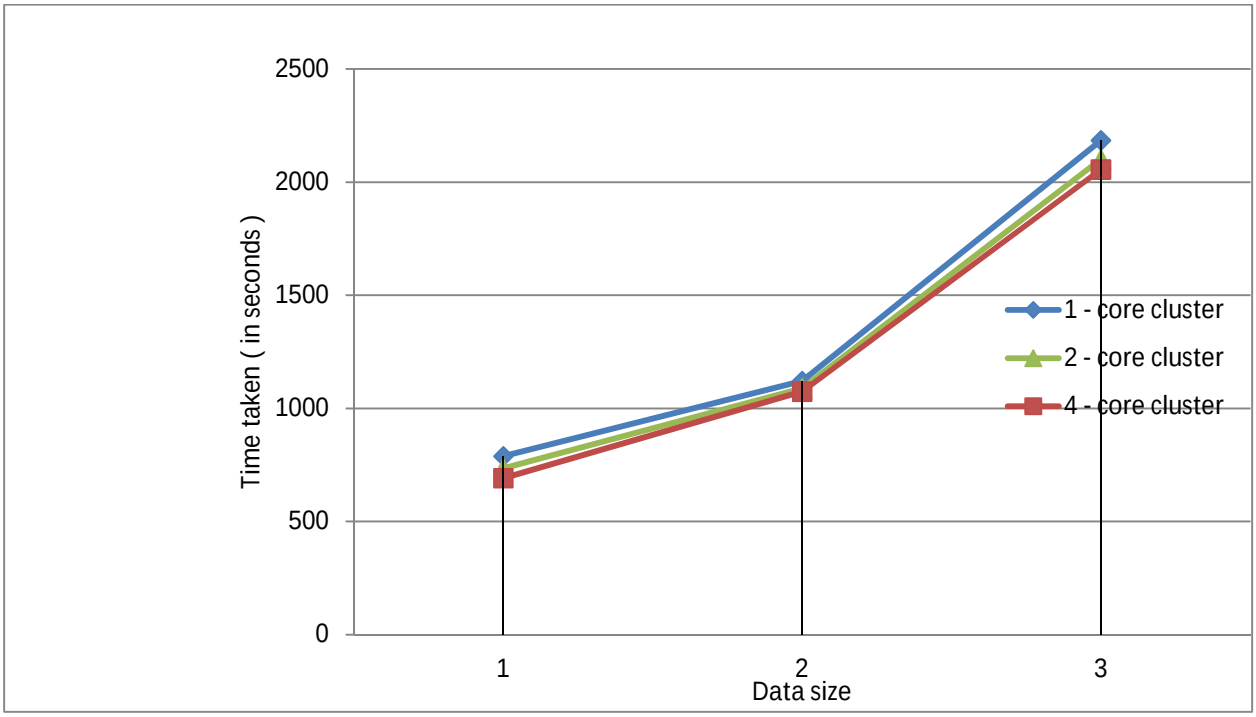


Graph plot 2 – Pig - Performance based on data size and number of nodes

Hive -

Nodes ======== Problem size	1 - core	2 - cores	4 - cores
Small	789	735	692
Medium	1,122	1089	1075
Large	2,185	2,102	2056

Fig 3 : Hive - performance on varying number of nodes and data size

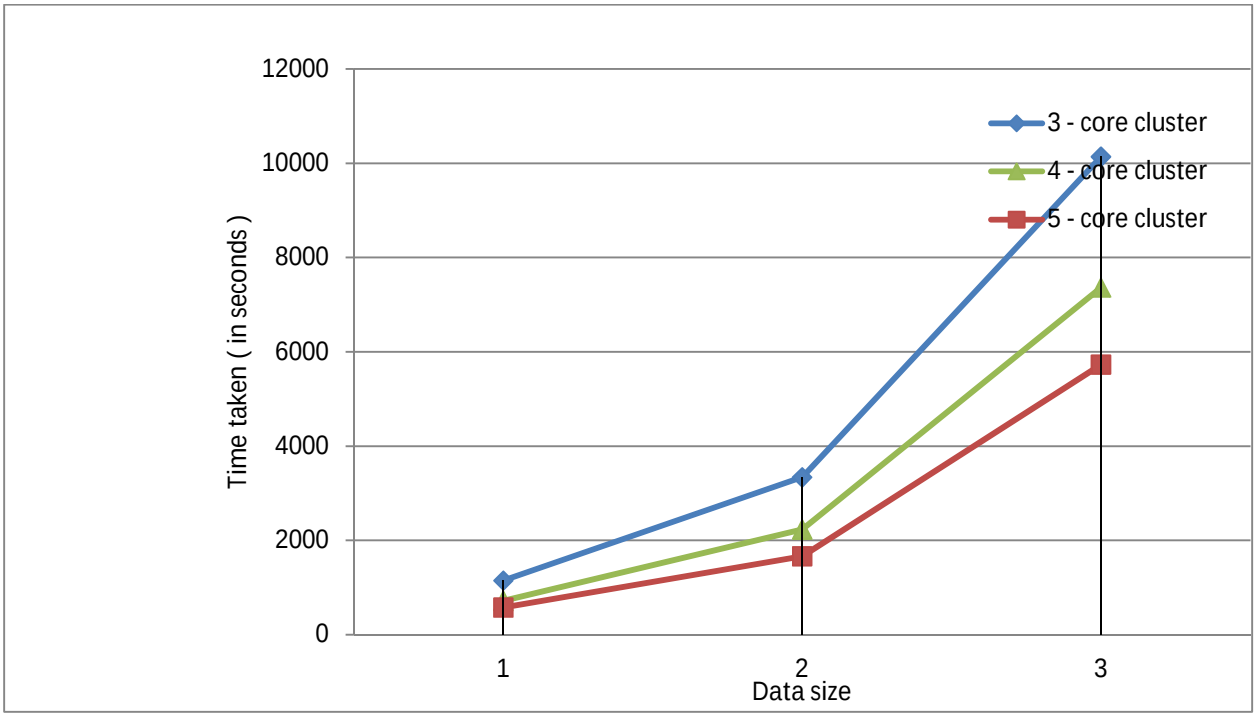


Graph plot 3 – Hive - Performance based on data size and number of nodes

HBase -

Nodes ======== Problem size	3 - nodes	4 - nodes	5 - cores
Small	1149	717	578
Medium	3341	2234	1666
Large	10141	7367	5732

Fig 4 : HBase - performance on varying number of nodes and data size



Graph plot 4 – HBase - Performance based on data size and number of nodes

- Map-Reduce, Hive and Pig perform better than HBase.
- As the size of the dataset increases, time consumed also increases.

Softwares used :

Hadoop 2.5.2

HBase – 0.98

References :

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

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

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

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

http://www.tutorialspoint.com/hbase/hbase_installation.htm

http://hbase.apache.org/book.html#mapreduce.example - Section 51.4