CSE 603: Programming Assignment 3
MATRIX CORRELATION COMPUTATION USING HADOOP MAPREDUCE

INTRODUCTION:

Implemented a Hadoop MapReduce program which takes a text file with VARIABLES (columns) and OBSERVATIONS (rows) as INPUT, and compute all possible CORRELATION VALUES into an output text file. I later compare the correlation results obtained using Hadoop MapReduce with Netezza.

STEPS TO RUN MY PROGRAM:

Executables required are placed in Hadoop_2 folder.

➔ Step 1: Input Matrix Generation into a .txt file

Source Code: 'Inp_Mat.java'

Input Arguments : Arg1: Variable Size / Columns

Arg2: Observation Size/ Rows

Output: Creates and outputs a .txt file with file name 'MAT_<Arg1>_<Arg2>.txt' which contains required Input Matrix.

Command:

$$java Inp_Mat <VAR/Cols> <OBS/rows>

➔ Step 2: Compute Correlation values using Hadoop MapReduce

Source Code: 'Corr_Mat.java' imports 'IndexPairsWritable.java' and 'ValuePairsWritable.java'

Input Arguments : Arg1: Path to MAT_<Arg1>_<Arg2>.txt (obtained in Step1)

Arg2: Path to Output File

Arg3: Number of Observations

Output: Creates and outputs a .txt file which contains our Correlation Values.

Command:

$$bin/hadoop jar Cal_Cor_Hd.jar org.chandu.Corr_Mat <Arg1> <Arg2> <Arg3>

Note: Add above command below 'Run Computation' section of the SLURM script in Hadoop_3 folder and also make sure you copy the Input File in <Arg1> into DFS.

➔ Step 3: Fetch Top 10 pairs with high correlation from Output.txt file generated in Step 2

Source Code: 'Get_Top10.java'

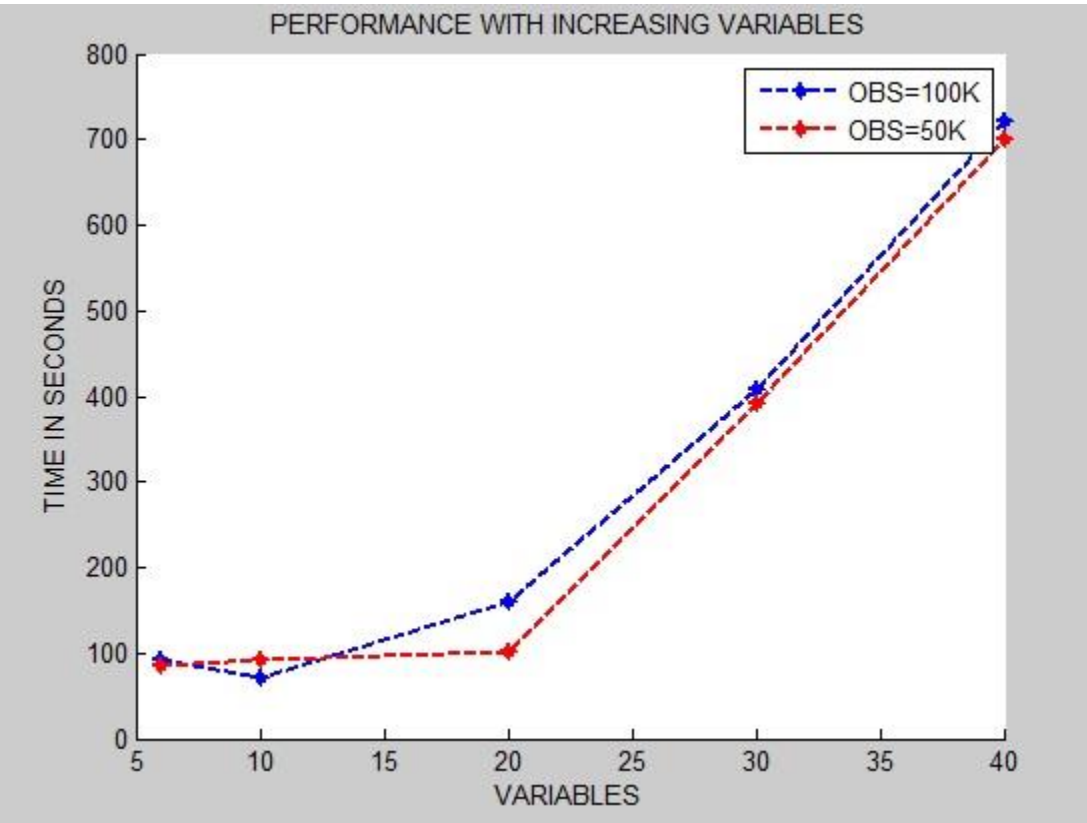
Input Arguments : Arg1: Output.txt

Arg2: Variables Size

Output: Outputs top 10 pairs with high Correlation Values on console.

OBSERVATIONS:

1. Impact on performance by increasing Variable Size:



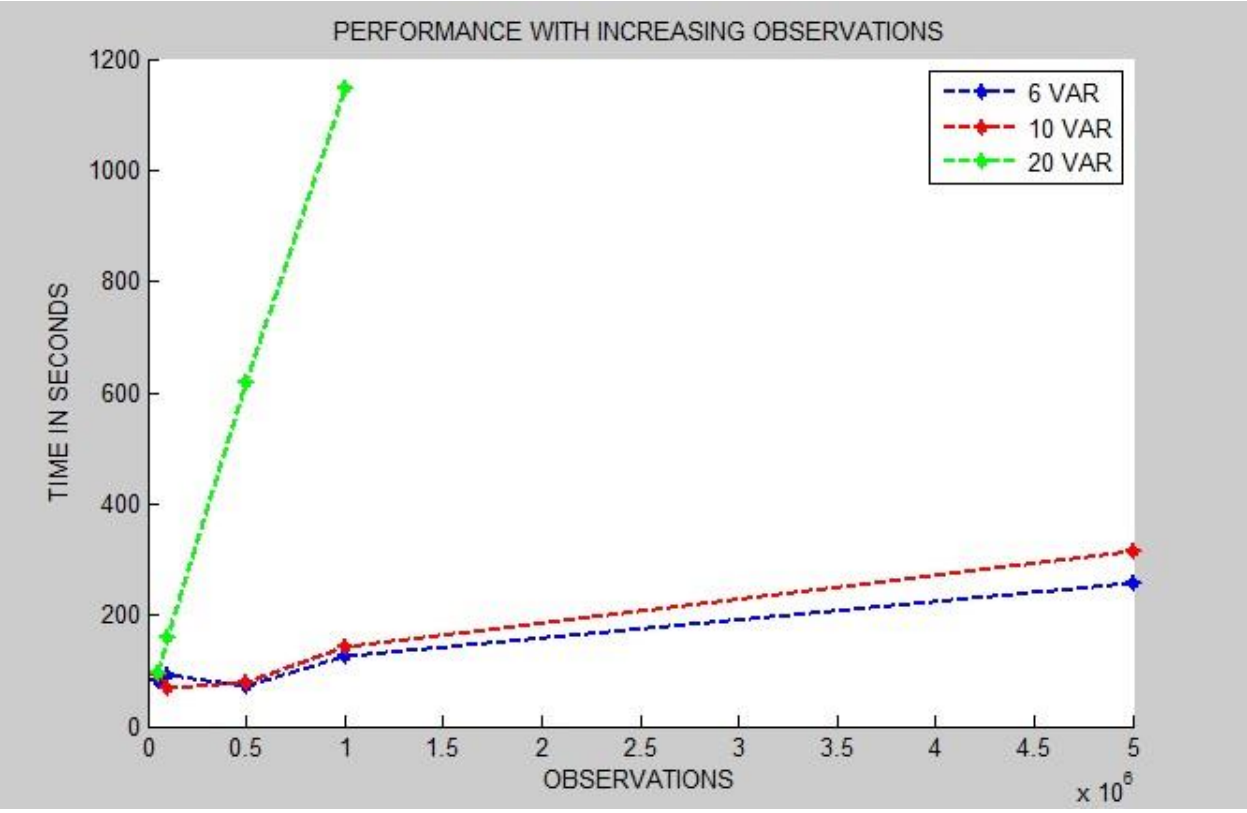
1. Time taken by MapReduce is significantly increased with increasing variable sizes after 20.

2. You can see that, for Variable size >= 20 there is sharp increase in time taken by the UDX for observations 50k to 5000k.

3. We compute correlation values for every pair of variables possible. Therefore number of possible pairs increases with increase in the number of columns/variables which explains increase in time with more variables.

4. Jobs time out and fail for Variable sizes > 50. It could be memory related issue.

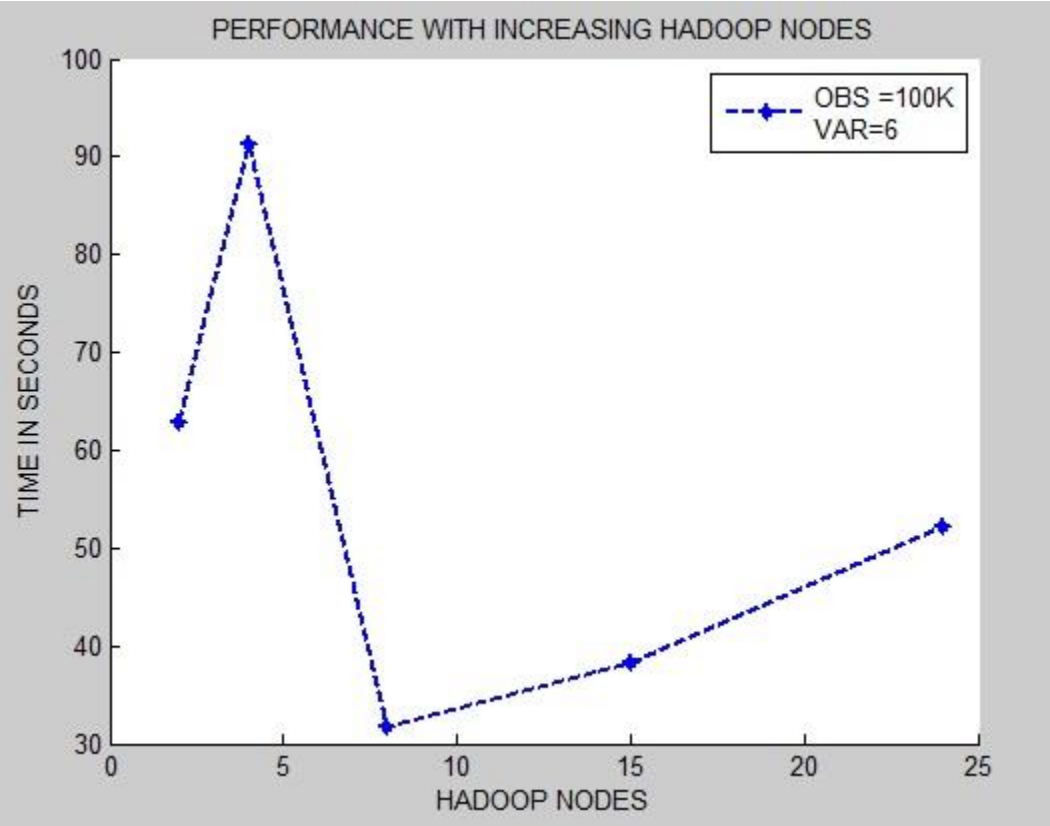
2. Impact on performance by increasing Observations with increasing Variables:



1. Time taken by MapReduce significantly increased with increasing observation sizes higher than 10^6.
2. Rows less than 10^6 doesn't show consistent behavior with respect to time taken. You can see the graph fluctuating up and down with observations <10^6.
3. You can also observe in this graph, the increase in TIME taken for 10 variables compared to 6 variables is less but performance dropped significantly for 20 variables and more.

You can conclude that increasing Variable Sizes after certain point (var >= 20) and also increasing Observations higher than 10^6 reduce the performance of MapReduce. If you compare both factors, Variable Sizes show significant impact rather than Observation count.

3. Impact on performance by increasing number of Hadoop Nodes:



1. Computing over 10^6 by 6 Input Matrix over 8 Nodes gave best performance.
2. 15 Node Hadoop cluster also gave much better performance than 4 Nodes or less.
3. Computing over 24 Node Hadoop clusters for this Matrix size however reduced the performance.

You can conclude that reduction in the performance of MapReduce with increasing Hadoop Nodes over 24 maybe due to the communication overhead generated over Hadoop cluster.

4. Comparing Hadoop MapReduce and Netezza :

1. Netezza UDXs I implemented earlier takes 11 seconds – 54 seconds for 10^5 Matrix – 5*10^6 Observations with 6 variables while Hadoop 4 Node cluster takes 71 seconds – 259 seconds. You cannot conclude based on the above observations because you can always add more Hadoop Nodes to achieve speed up. Hence Netezza UDXs and Hadoop Map Reduce offers similar performance.
2. NetezzaAnalytic Function i.e CORR_MATRIX_AGG offers far better performance than my UDXs and MapReduce program.
3. In Hadoop we don't need to load humungous amount of Input data into Table like we do in Netezza which is a huge performance overhead.

5. Limitations:

1. Job occasionally fails with DataStreamerException while loading data into hadoop distributed file system (hdfs).
2. Job also fails i.e TimeOut sometimes and also JobTracker on a hadoop node stops running.
3. Resubmitting the jobs will do for the above failures.