HW04 Report

Ishan Bhatt
5013-4076

The results were same as that of Hadoop using Map Reduce, Pig and Hive.

Results

Small

The top 10 stocks with highest volatility:	**The top 10 stocks with lowest volatility:**
ACST.csv	LDRI.csv
NETE.csv	GAINO.csv
XGTI.csv	VGSH.csv
TNXP.csv	MBSD.csv
EGLE.csv	TRTLU.csv
PTCT.csv	AGZD.csv
GOGO.csv	SKOR.csv
MEILW.csv	CADT.csv
ROIQW.csv	AXPWW.csv
CFRXZ.csv	VCSH.csv

Medium

The top 10 stocks with highest volatility:	**The top 10 stocks with lowest volatility:**
ACST-3.csv	LDRI-1.csv
ACST-2.csv	LDRI-2.csv
ACST-1.csv	LDRI-3.csv
NETE-3.csv	GAINO-1.csv
NETE-2.csv	GAINO-2.csv
NETE-1.csv	GAINO-3.csv
XGTI-3.csv	VGSH-1.csv
XGTI-2.csv	VGSH-2.csv
XGTI-1.csv	VGSH-3.csv
TNXP-3.csv	MBSD-1.csv

Large

The top 10 stocks with highest volatility:	**The top 10 stocks with lowest volatility:**
ACST-9.csv	LDRI-1.csv
ACST-8.csv	LDRI-10.csv
ACST-7.csv	LDRI-2.csv
ACST-6.csv	LDRI-3.csv
ACST-5.csv	LDRI-4.csv
ACST-4.csv	LDRI-5.csv
ACST-3.csv	LDRI-6.csv
ACST-2.csv	LDRI-7.csv
ACST-10.csv	LDRI-8.csv
ACST-1.csv	LDRI-9.csv

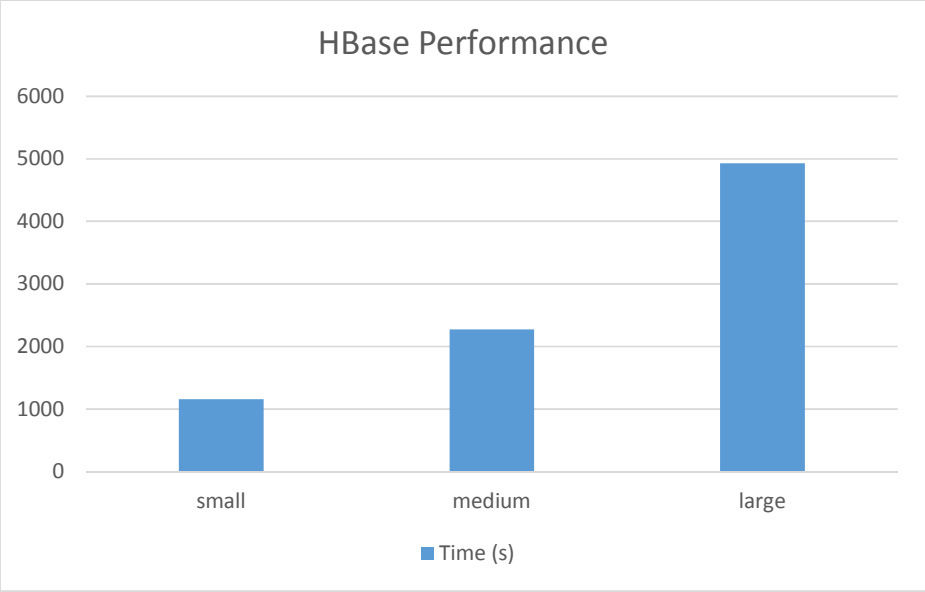
HW04 Report

Ishan Bhatt
5013-4076

I was not able to make a speed up graph because initially I was collecting results for 1 and 2 nodes. When later I came to know we have to find results for higher number of nodes then such high number of nodes were not available although I kept my job in queue for more than 24 hours.

For 3 nodes each having 12 cores I got the below results.

Data Load	Time (sec)
Small	1157
Medium	2273
Large	4928



Design Approach & Issues

Data was fetched using scan and get Java API's. Initially I tried implementin an approach where I was filtering data using SingleColumnValueFilter in HBase. However I figured out that performing aggregation on this data was not possible by using get/scan or put. Other option was to load filtered data in a Linked List and process the data obtained, but that is a very Heap expensive option. So I finally used Map Reduce to process as it seemed a viable option. I also changed the Row Key format in the Uploader part of the script provided so as to receive sorted data for processing.

HW04 Report

Ishan Bhatt
5013-4076

HBase Rationale

HBase is a distributed, scalable data store model. We used Pig and Hive by using the File System HDFS which although powerful but still is a File System and limits our scope when it comes to indexed access. However Hbase stores data as key/value pair, so when indexed access is required HBase does the job best.

Scalability and Performance Measure

For small data sets using Hbase doesn't reflect the performance difference, however for larger data sets as in our case the medium and large data sets we are clearly able to see the difference. For medium there is a very meagre difference in the time however for large data set the difference seems quite considerable.