CSE 587

Data Intensive Computing

Homework-1

- Abdul Muqtadir Mohammed (5013-5042)

1.1 Introduction:

The project is aimed to find top ten most volatile stocks and top ten least volatile stocks using HDFS. HDFS is designed to store very large datasets reliably. Given are three datasets called large, medium and small. These datasets contain many files; each having stocks pricing data with respect to the company. We aim to use MapReduce in order to speed-up the process of reading files and processing them in parallel. The input files are read by multiple mappers and partitioned into smaller parts for their processing. The mapper then outputs <key, value> pair which serves as an input to the reducer. Reducer then takes the <key, List<value>> pair as it's input and processes and outputs the results to the user.

1.2 Project Description:

In our project, we are using two MapReduce jobs which are processing the stock data.

Our first MapReduce job is responsible for calculation of each company's stock volatilities. The mapper, takes the <File Line number, Line> as it's input and gives <CompanyName, Date|Price> as its output. The reducer then takes the <CompanyName, Date|Price> as it's input and calculated the volatility for each company and writes the volatilities of all the companies to an intermediate file called 'intermediate-output'.

Our second MapReduce job is responsible for sorting all stock volatilities and finding the top 10 most and least volatile stocks. The mapper reads each line and outputs it to the reducer. Here, we have explicitly used only one reducer for the purpose of comparing all the volatilities.

The MapReduce job is written in JAVA and exported as a RUNNABLE JAR, which is then run using the SLURM script. No extra libraries were used except the basic libraries for hadoop.

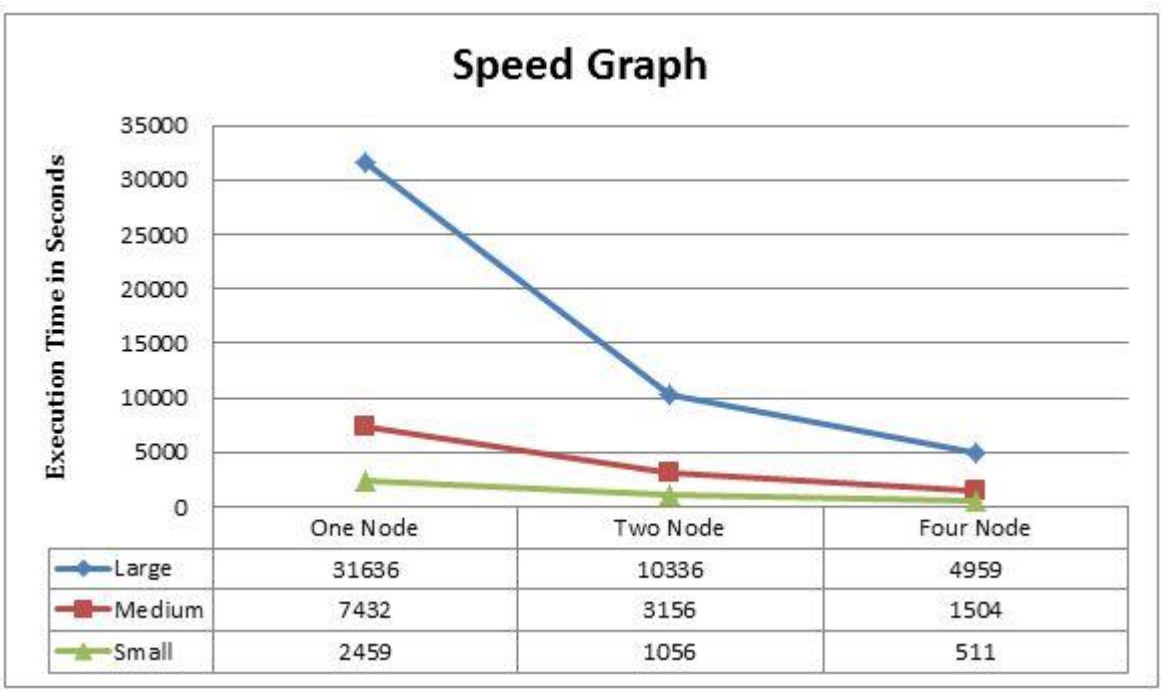
Our final output shows top 10 most and least volatile stocks along with their volatilities. Example,

```
The top minimum values
LDRI-3  5.149336582098077E-4
LDRI-2  5.149336582098077E-4
LDRI-1  5.149336582098077E-4
GAINO-3 5.650074160499658E-4
GAINO-2 5.650074160499658E-4
GAINO-1 5.650074160499658E-4
VGSH-3  0.0013014906189562881
VGSH-2  0.0013014906189562881
VGSH-1  0.0013014906189562881
MBSD-3  0.0025000459104618893
MBSD-2  0.0025000459104618893
MBSD-1  0.0025000459104618893
The top maximum values
ACST-3  9.271589761859984
ACST-2  9.271589761859984
ACST-1  9.271589761859984
NETE-3  5.396253961502244
NETE-2  5.396253961502244
NETE-1  5.396253961502244
XGTI-3  4.542344311472958
XGTI-2  4.542344311472958
XGTI-1  4.542344311472958
TNXP-3  3.2483321967818664
TNXP-2  3.2483321967818664
TNXP-1  3.2483321967818664
```

Fig. 1.2 Sample Output

1.3 Speed Graphs:

We have plotted the graphs for each of the dataset in seconds for 1, 2 and 4 nodes respectively.



	One Node	Two Node	Four Node
Large	31636	10336	4959
Medium	7432	3156	1504
Small	2459	1056	511

Fig 1.3, Performance speed graph

1.4 Performance & Scalability:

We have measured our performance against 1, 2 and 4 nodes for datasets that have around 3.6GB (Large), 1.1GB (Medium) and 363MB (Small) of stock prices for the companies distributed across several files on UB CCR which has 12 processors, 48GB memory. As you can clearly see from Fig 1.3, as number of nodes increase, our execution time has drastically come down. This is due to the increase in parallel execution. The results get better (in terms of execution time) as we scale to larger dataset, when compared to Small/Medium dataset. This echoes the point that MapReduce is highly scalable.

Reference:

https://ublearns.buffalo.edu/bbcswebdav/pid-3270184-dt-content-rid-8703372_1/courses/2151_24487_PsC/mapreduce-osdi04.pdf