Stock Analysis using Hadoop

Pratham Malik
(prathamm)

1. Method and Implementation

The project requirement was to calculate monthly volatility for each stock. The final output was to show the top ten and bottom ten stocks based on the value of volatility.

The method used in order to calculate the top 10 and bottom 10 stocks with respect to volatility are explained with the use of three phases corresponding to three mappers-reducers functions.

We have used three mappers and reducers in order to process the input data and deduce the final result. Thus there are three phases which are used to calculate the volatility and then deduce the top ten and bottom ten stock names.

- **First Phase**

In this phase the mapper is extracting each line from each stock file and forming a key value pair for each fund. In this function we parse each line and make a key value pair for the fund and date and the adjusted close price. The reducer receives all the values for one fund and forms a treemap. Next we retrieve the month end adjusted close price and month beginning adjusted close price from the treemap and calculate the month end adjusted close price for each fund. This price is written as the output of the reducer for each month for every fund.

- **Second Phase**

In this phase, mapper extracts each line from the output file of first phase reducer and makes a key value pair as per the fund name. The reducer receives the key value pair for each fund. In the reducer function, we calculate the volatility of each fund and write it to the output of the reducer.

- **Third Phase**

The final phase is just for sorting out the volatility values in order to get the top ten stocks and bottom 10 stocks with respect to volatility values. The mapper appends the same key to the each fund. The reducer has the complete volatility values of all the stocks which are stored in a hashmap. The value of volatility of each fund is also stored in an array list. The array is then sorted in ascending order and descending order and the top ten stocks and bottom ten stocks are written to the output file of the reducer. In order to properly map the values, hashmap is used which has constant time complexity for retrieval of values.

In Hadoop, the number of input files are equal to the number of mapper which can be run simultaneously. The biggest task of our code is to parse the input stock files. Thus the parallelism obtained in parsing the input stock files will lead to faster time output. The parallelism surely brings more network overhead. The performance of a particular program depends on multiple factors, such as the platform, algorithm used to solve the problem and proper data structure.

In order to efficiently sort the values in the code, treemap has used. Treemap is used to store values in first and second phase. Using treemap guarantees that elements would be sorted in order of ascending key order. The first and last days of any month are also calculated based on this principle of a tree. The treemap has the values in sorted order and it is quite efficient and fast to retrieve the first and last values without iterating through the whole map of values.

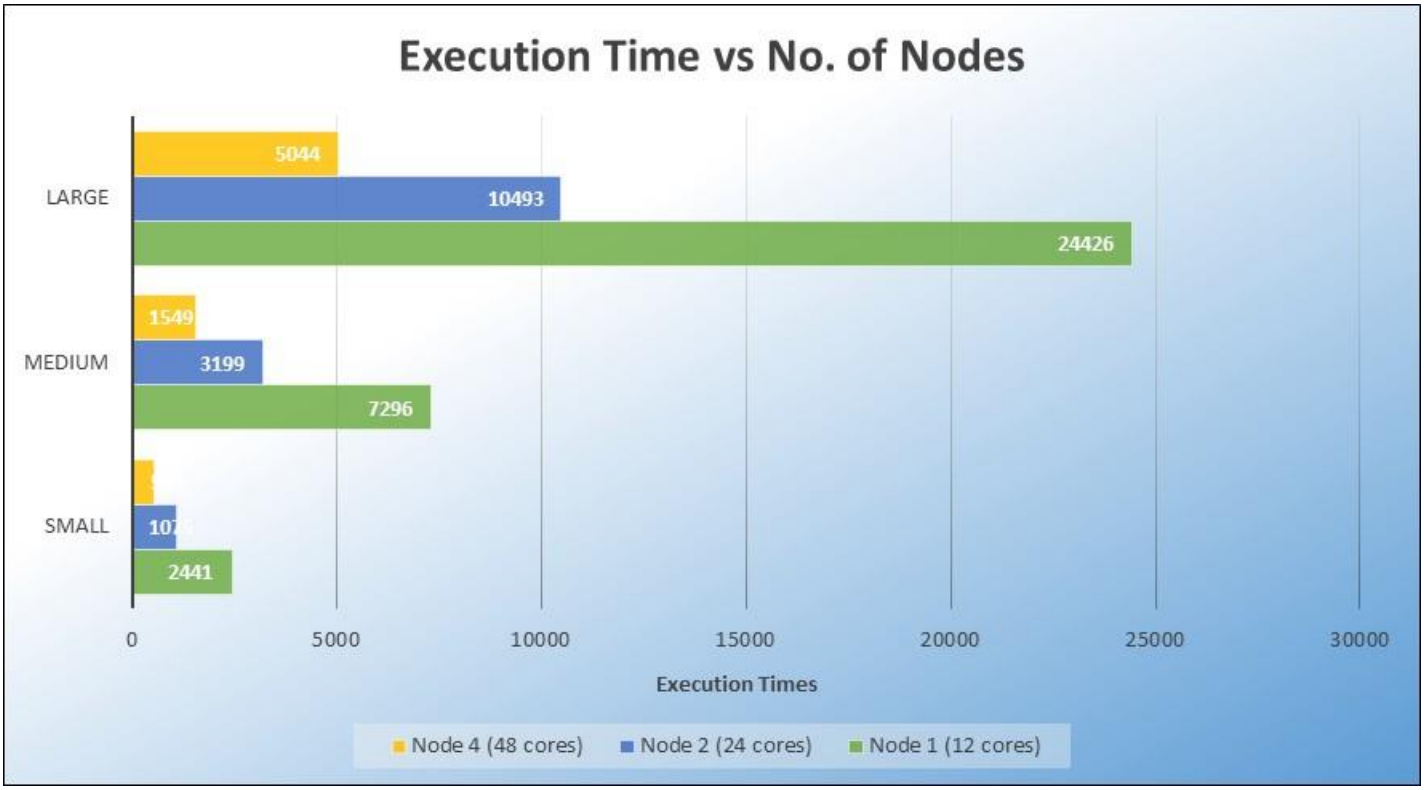
2. Experimentation and Discussion

The experiments are conducted on CCR in UB. The program is first developed the eclipse IDE in one single machine, then the jar file is exported and delivered to the CCR and executed for three problem size: Small, Medium and Large over 1 node, 2 nodes and 4 nodes. Each node specified 12 cores.

The experiment was executed on small data with increasing the number of nodes with each node using 12 cores. The experiment result is shown in Figure 1 below. The result shows that on increasing the number of cores, the execution time reduces almost by a factor of 0.5

	Execution Times		
	Node 1 (12 cores)	**Node 2 (24 cores)**	**Node 4 (48 cores)**
Small	2441 seconds	1075 seconds	544 seconds
Medium	7296 seconds	3199 seconds	1549 seconds
Large	24426 seconds	10493 seconds	5044 seconds

Table 1.0 – Execution Times on various Dataset size with different number of nodes



Graph 1.0 – Execution Times on various Dataset size with different number of nodes

The speed up is achieved by increasing the number of cores as it can be clearly observed from the graph above. The problem has a major bottleneck of initial parsing of the input files. The speed up is majorly achieved by increasing the number of cores and thereby increasing the number of mappers. This increase in the number of simultaneous mapper operations on the input files reduces the operation time of parsing through each stock file by a major amount.

The performance (execution times) is clearly dependent of the number of cores. As we can clearly observe from the graph above that on increasing the number the cores for each problem set, the execution time has decreased by almost a factor of half.

3. Scalability

In order to achieve better performance, we could use more number of nodes so that maximum number of mappers could work in parallel and thus give better performance. We should also note that increasing the number of nodes would also result in increase of network overhead. We could also have nodes with increased number of cores.