Data Intensive Computing

Report for Assignment 1

Compute the Volatility of Stocks in NASDAQ

Name: **Mimanshu Shisodia**
Person No: **50133318**

Instructor: Prof. Vipin Chaudhary

1. Method and Implementation

In this assignment we were given the data of 2970 stocks on NASDAQ market for 3 years from 01/01/2012 to 12/31/2014.

I have used a method with **2 Map Reduce jobs**. The work happening in these specific jobs is explained below.

Type of Data Set:

1. We have been provided with a data set of comma separated files with the monthly trading value of the stocks.
2. The data rows are in descending order of the monthly dates. For ex. the data for a particular month starts from the last traded day in the month and ends with the first traded day of the month.
3. This structure of data is being utilized in the 1st Map Reduce job.

1st MAP REDUCE JOB

Sample Input Record:

Date, Open, High, Low, Close, Volume, Adjusted Close Price
2014-12-31, 50.68, 50.68, 50.68, 50.68, 000, 50.55

What 1st Mapper will do?

The Basic Strategy

- The key will be the name of the file in which the current exist and that is being read by the Mapper.
- I am keeping the values of previous record's company, year, month, day and adjusted close price information.
- Since the dates are coming in sorted order of the days of the months, so I am utilizing this information to filter out the records and minimizing my keys in the 1st Mapper itself.
- If all the values are same as that of previous record, then no key, value will be written.
- If the company name, year, month are same but the date is different than the previous record is updated, so that when the month gets changed or a company gets changed I can write this previous record at that time.

- When there is a change in company or the month, then the previous year records gets emitted and the values are updated with the new values. Since this is a change and values are coming in a sorted order, hence the very first records of this changed data will be written off.
- Hence the strategy is to emit as fewer keys as possible while emitting all the required keys needed to calculate the volatility.

Therefore, after the first Mapper, my output key value pairs are:

The input and output of the mapper in the 1st job are:

File Name as Key

Input	**Output**
Date, Open, High, Low, Close, Volume, Adj Close	
AAPL	
2014-12-31, 50.68, 50.68, 50.68, 50.68, 000, 50.55	< AAPL, (2014, 12, 31, 50.55) >
GNOPI-1	
2012-09-18, 50.54, 50.57, 50.54, 50.57, 900, 50.44	<GNOPI-1, (2012, 09, 18, 50.44) >

What Reducer will do?

Formula for Stock Volatility of Monthly Rate of Return:

x = Monthly Rate of Return = (Month end adjusted close price – Month beginning adjusted close price) / (Monthly beginning adjusted close price)

volatility = $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$, $\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$, where N is set to 36 (months), ending close price is the close price of the last trading day in each month, beginning close price is the close price of the first trading day in each month.

The Basic Strategy

- The Reducer will get all the values related to that company including the data for all the month, since the key is the company name (considering the different File Names represents different company).
- The first aim is to calculate the **xi** values, for which we will be needing the monthly start and ending values. Since we have all the values for a month (many of most I filtered out in the first step) I am using a **Dictionary of Dictionary** concept like to store my values.

    - **HashMap<String, TreeMap<String, String>> map**
    - **The key of map will be {Company_Name--Year--Month}.**
    - **The value is a TreeMap with the day as the key and closing price as the value.**

- The reason behind using such architecture is that I will have all the monthly values of a company in the TreeMap and the TreeMap is because I will be needing only the first and last value.
- After storing all the information in a dictionary, I am iterating over this and getting the first and last value from the tree and calculating the **xi**.
- After calculating the xi, I am summing up the calculated values together with a counter. After the iteration step, I will find the x_bar (or x prime) value dividing the sum by counter. Also I am storing the xi values in an another list.
- No comes the volatility calculation part, since I have all the required data, I will use the above mentioned formula to calculate this.
- After calculating, the reducer will emit the volatility and company name as the key value pair.
- **Volatility as Key and Company Name as Value.**

Input	Output	
AAIT 10 25.46 2012 02	0.04397065503965248	AAIT-1
AAIT 10 25.46 2012 02	0.04397065503965247	AAIT-2
AAIT 29 26.55 2012 02	0.04397065503965247	AAIT-3
AAIT 01 26.55 2012 03	0.16471174982556286	AAL-1
AAIT 30 25.66 2012 03	0.16471174982556283	AAL-2
AAIT 02 25.66 2012 04	0.16471174982556283	AAL-3
AAIT 30 27.20 2012 04	0.06414854859871265	AAME-1
AAIT 01 27.20 2012 05	0.06414854859871265	AAME-2
AAIT 31 24.14 2012 05	0.06414854859871265	AAME-3
AAIT 01 23.12 2012 06		
AAIT 29 24.69 2012 06		
AAIT 02 24.49 2012 07		
AAIT 31 23.17 2012 07		
AAIT 01 24.01 2012 08		
AAIT 31 24.27 2012 08		
AAIT 04 24.27 2012 09		
AAIT 28 25.75 2012 09		
AAIT 01 25.75 2012 10		
AAIT 31 25.30 2012 10		
AAIT 01 25.47 2012 11		
AAIT 30 26.60 2012 11		
AAIT 03 26.73 2012 12		

Input and Output of Reducer 1

Note: Not all the values of Mapper 1 Output is shown as they are too many, hence should not be matched with actual calculations

2nd MAP REDUCE JOB

What Mapper will do?

The Basic Strategy

- The Mapper will receive the values from the first Reducer where the key is volatility and company name as the value.
- The second mapper will not do any calculations and will just emit the values it is receiving.
- The idea is to sort the value in the order of volatility values. Since the key-value pairs when emitted from the mapper will be sorted on the basis of keys when reached to reducer, hence this will be utilized for getting the max and min Company names.

Input		Output	
0.04397065503965248	AAIT-1	0.04397065503965248	AAIT-1
0.04397065503965247	AAIT-2	0.04397065503965247	AAIT-2
0.04397065503965247	AAIT-3	0.04397065503965247	AAIT-3
0.16471174982556286	AAL-1	0.16471174982556286	AAL-1
0.16471174982556283	AAL-2	0.16471174982556283	AAL-2
0.16471174982556283	AAL-3	0.16471174982556283	AAL-3
0.06414854859871265	AAME-1	0.06414854859871265	AAME-1
0.06414854859871265	AAME-2	0.06414854859871265	AAME-2
0.06414854859871265	AAME-3	0.06414854859871265	AAME-3

What Reducer will do?

The Basic Strategy

- It received the records sorted on the basis of volatility values.
- My goal is to find the top and last records. Hence I emitted the first 10 keys as the lowest volatility keys and stored all the other keys in a HashMap
- When all the values are stored in a HashMap, the reducer called the cleanup method.
- In the cleanup method using the TreeMap concept I have provided all the HashMap values to the TreeMap object and sorted the tree on the basis of values using a Comparator.

Input		Output	
0.04397065503965248	AAIT-1	AAL-1	0.16471174982556286
0.04397065503965247	AAIT-2	AAL-2	0.16471174982556283
0.04397065503965247	AAIT-3	AAL-3	0.16471174982556283
0.16471174982556286	AAL-1	AAME-1	0.06414854859871265
0.16471174982556283	AAL-2	AAME-2	0.06414854859871265
0.16471174982556283	AAL-3	AAME-3	0.06414854859871265

Hence the output of second reducer will be like:

The top 10 stocks with the lowest (min) volatility are

LMBS	**0.00935438916783161**
ACGL	**0.034218238099872095**
AAXJ	**0.04026834416672968**
LMCK	**0.041062902126481726**
AAIT	**0.04397065503965248**
LMCA	**0.04967810154383621**
LORL	**0.051489755958371596**
ABDC	**0.05211313045695302**
ACAS	**0.057710939347078036**
AAME	**0.06414854859871265**

The top 10 stocks with the highest (max) volatility are

ACHN	**0.36045035440574785**
ACAD	**0.2775314640493186**
ABIO	**0.24684758511131555**
AAOI	**0.2449988018226614**
ACFN	**0.2088223175792808**
LPCN	**0.19433143954199625**
ABAC	**0.18803623616587364**
ABGB	**0.1853537442513966**
LRAD	**0.16755758412179297**
AAL	**0.16471174982556286**

2. Experiment and Discussion

The program was developed on Eclipse IDE first and was converted into a jar file.
The jar was ran on CCR at UB. Configurations used are:

12 cores at each node
24 GB Memory
Connected to Infiniband Network

Performance on Data Scaling

As the size of the data increased, the processing time got increased. This increase was not very linear, since the records read by the Mapper will gets increase and the reducer was getting more records. Although the in the 1st Map task, disk write increase has contributed majorly on the increase of the time, since number of blocks have been increased. But the data filtering strategy at mapper will not increase the keys at the reducer level, in the same order as the data is scaling up. Therefore, the major difference created at the scaling up was at the mapper level.

Performance on Node Scaling

Since the number of nodes got increased with the increase in the number of processors, the execution time was also reduces. This shows the parallel nature of Hadoop Distributed file system performance. Since now the reducers are doing task in parallel, hence the reduction in time was noticed.

Speedup graphs

Problem Size	Execution Time: 1 node (12 cores)	Execution Time: 1 node (24 cores)	Execution Time: 1 node (48 cores)
Small	**2408 Seconds (40.133 Mins)**	**1098 Seconds (18.33 Mins)**	**517 Seconds (8.6167 Mins)**
Medium	**7189 Seconds (119.81 Mins)**	**3102 Seconds (51.7 Mins)**	**1560 Seconds (26 Mins)**
Large	**31617 Seconds (8.67 Hours)**	**10695 Seconds (2.97 Hours)**	**4880 Seconds (1.35 Hours)**

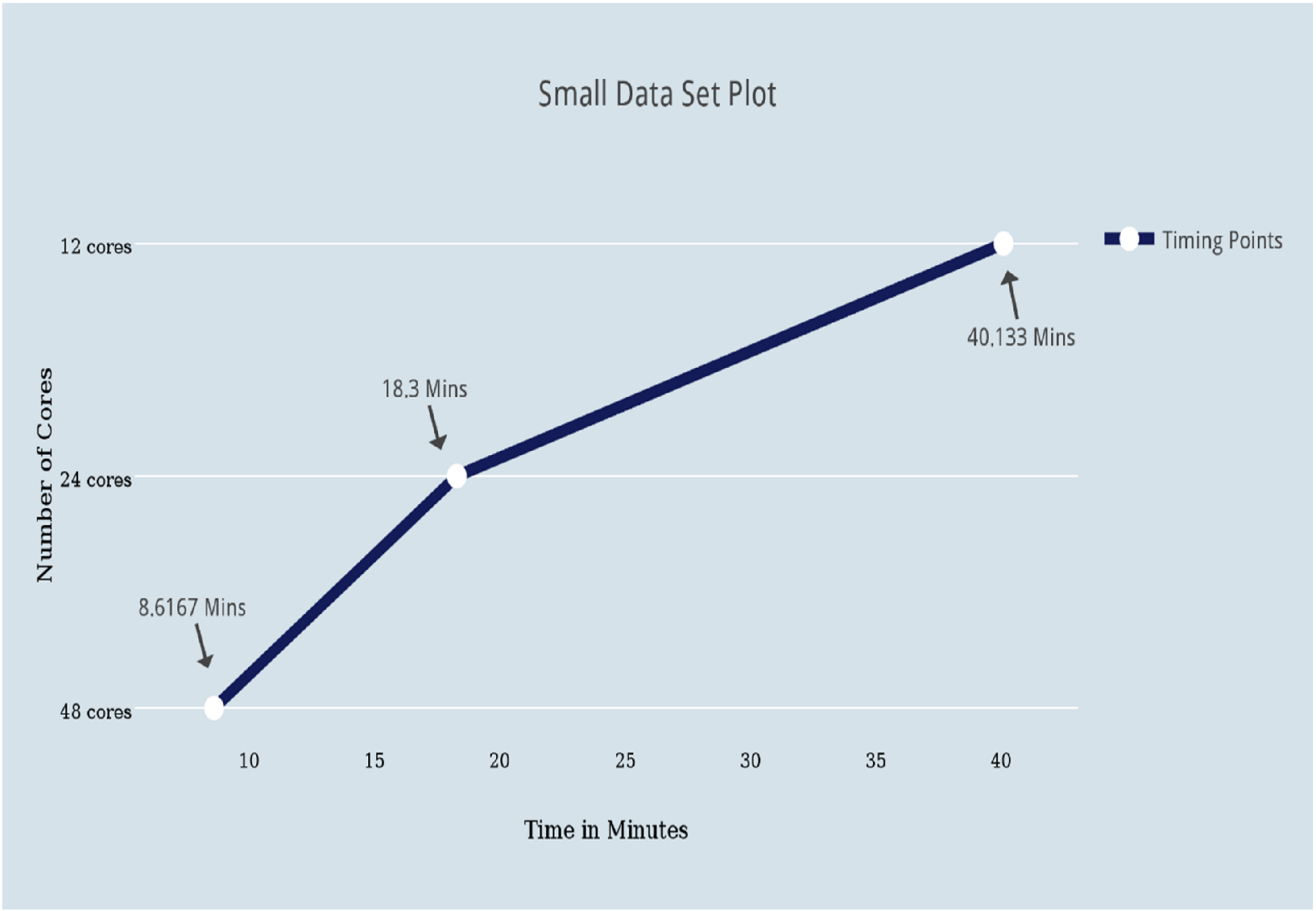

Figure 1: Small Data Set Execution times

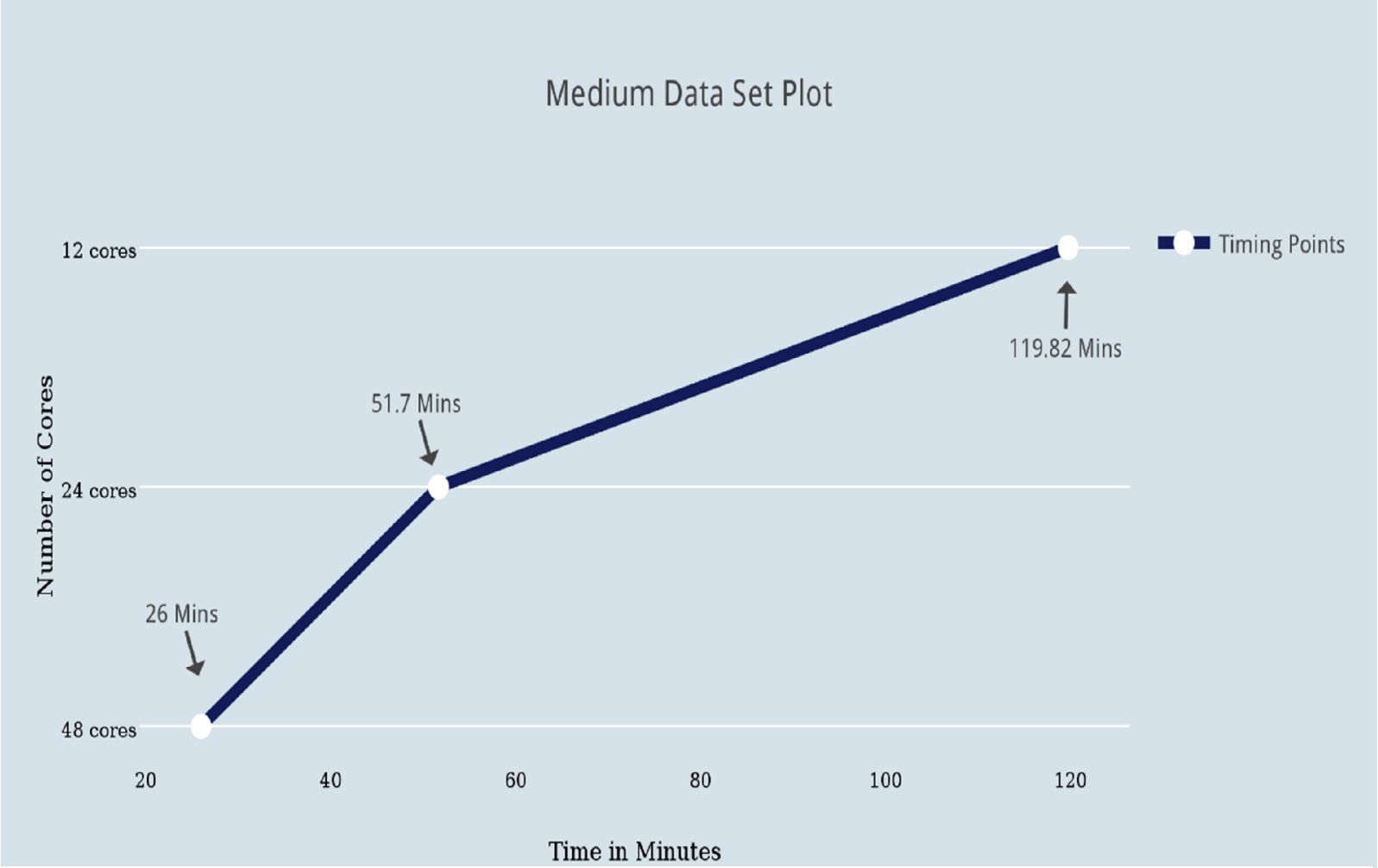

Figure 2 Medium Data Set Execution times

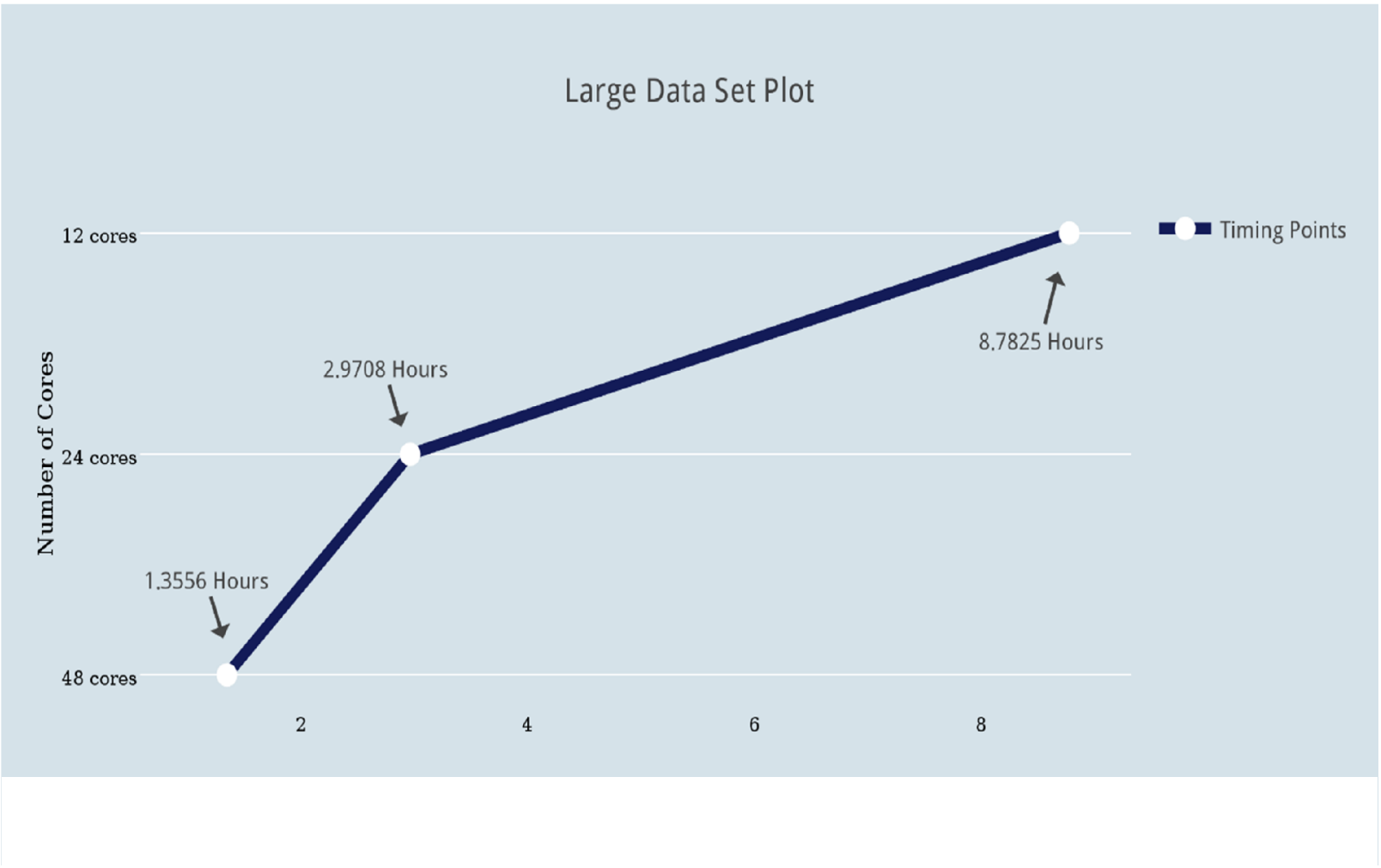

Figure 3 Large Data Set Execution times