Big Data Processing Systems Project Report

António Mendes, *Aluno 55093,* João Naves, *Aluno 55091,* and Ricardo Afonso, *Aluno 56336*

Abstract—This Report explains how our group solved the project assignment for the Big Data Processing Systems Course, using all of the studied technologies as well as what we learned from our work and any other observations regarding this project.

Index Terms—Big Data Processing, Map-Reduce, Spark, Dataframes, SQL, Hive.

1 Introduction

For our Big Data Processing Systems course, we were asked to process a Data Set of 2.6 GB that contains information regarding USA's Environmental Protection Agency (EPA) monitors tracking air pollutants, using all the technologies we studied during our course to prepare indexes that would help answer the 5 questions stated on our project assignment PDF file [1].

2 Experimental Work

2.1 Question 1

The first question regarding our experimental work required us to rank states regarding the number of EPA monitors/sensors. To answer this question it is important to understand the concept of *key*. As we learned early on in our course, a key is an attribute or group of attributes that can uniquely identify a record, and for this case, our group determined that the group *State Code*, *County Code* and *Site Num* would form the *key* that uniquely identifies monitors/sensors since *Site Num* identifies a unique monitor site whitin a unique county and the *State Code*[1] and *County Code* together form the (FIPS) code to uniquely identify a county. The next and final step will be to group the monitors per state, count the rows and rank them to produce our desired output and answer the question. The code written for all technologies in our Github page [2] is similar to the following pseudo-code adopting the strategy mentioned above:

```
map(Key, Value):
((stateCode/Name, countyCode, siteNum), 1);
for each distinct tuple:
 Emit(tuple);
for each stateCode/Name in Key:
    result = 0;
    for each count in Value:
        result += count;
    Emit(stateCode/Name, result);
```

• *António Mendes, João Naves and Ricardo Afonso are students from the NOVA University of Lisbon School of Science and Technology and members of Group 05.*

Professor João Lourenço, NOVA School of Science and Technology Computer Science Department.

1. Note that it is also possible to use State Name in the *key* combination instead of State Code since all States have unique names.

2.1.1 Produced Index and Execution Times

Execution times were registered using the terminal command 'time' to obtain the 'user time'(Amount of CPU time outside of kernel within the process) and are as follows:

- Map-Reduce: 32.9060 sec
- Spark: 0.7280 sec
- SparkDF: 1.1460 sec
- SparkSQL: 1.1720 sec
- Hive: 9.676 sec

Produced Index sample:

state_name	number_monitors
California	162
Texas	132
Minnesota	94
Ohio	89
Michigan	84
New York	66
South Carolina	64
Pennsylvania	60
Montana	60
Indiana	52
Colorado	51
Florida	50
Illinois	50
North Carolina	49
Washington	42
Louisiana	40
Arizona	38

2.2 Question 2

For our second question we were tasked with ranking counties considering pollutant's level. As previously stated, we can identify a unique county by combining the attributes *State Code* and *County Code* which form the (FIPS) code. All that's left is to calculate the pollutant's level which can be done by calculating the average of the daily *Arithmetic Mean*. The following pseudo-code describes the adopted strategy to our code:

```
map(Key, Value):
(FIPS, (countyName, mean));
Emit(Key, Value);
for each Key do:
    Emit(countyName, Average(mean));
SortBy(Average(mean));
for each tuple:
    Emit(countyName, Average(mean));
```

2.2.1 Produced Index and Execution Times

Execution times were registered as follows:

- Map-Reduce: 20.7430 sec
- Spark: 0.7720 sec
- SparkDF: 1.1820 sec
- SparkSQL: 1.3070 sec
- Hive: 2.451 sec

Produced Index sample:

County	Pollutant_levels
Tipton	2556.0
Nassau	19.0
Columbiana	7.385690735785953
Park	5.611212121212121
CHIHUAHUA STATE	4.5121875
Caldwell	4.116666666666667
Kings	3.9843770491803276
Madera	3.7393
Franklin	3.3499999999999996
Jefferson	3.07
Oakland	2.888877848101266
Lake	2.879328647058823
Duval	2.7794603978494625
Middlesex	2.6500000000000004
Kearny	2.3753333333333333
Bucks	2.3674999999999997
San Luis Obispo	2.3333333333333335

2.3 Question 3

The third question ends up being very similar to the previous question where instead of ranking counties based on pollutant's level, now the task is to rank states pollutant's level each year. Since we are ranking states per year, this time our *key* will result from the junction of *State Code* and the year on *Date Local*. To obtain the year from *Date-Local*, we will have to split the string knowing that the date format is *'yyyy-mm-dd'*. The calculus process for the pollutant's level will be the same as in the previous question. Due to the similarities shared between question 2 and question 3, the pseudo-codes to describe the strategies of both questions will be fairly similar:

```
map(Key, Values):
((State Name, Date-Local[4:]), mean)
Emit(Key, Value);
for each Key do:
    Emit(Key, Average(mean));
SorBy(Average(mean));
for each tuple:
    Emit(Key, Average(mean));
```

2.3.1 Produced Index and Execution Times

Execution times were registered as follows:

- Map-Reduce: 25.7890 sec
- Spark: 0.8310 sec
- SparkDF: 1.2810 sec
- SparkSQL: 1.3130 sec
- Hive: 2.792 sec

Produced Index sample:

Year	State	Pollutant_levels
1990	Wisconsin	0.0
1990	Oklahoma	0.0
1990	Virgin Islands	0.0
1990	West Virginia	0.0
1990	Hawaii	1.970370370370370...
1990	Nevada	4.208000000000000...
1990	Alaska	4.420833333333333...
1990	South Dakota	5.705E-4
1990	Washington	5.974999999999999E-4
1990	Wyoming	6.045454545454545E-4
1990	Utah	7.970588235294118E-4
1990	New Mexico	8.222222222222222E-4
1990	Oregon	8.596296296296297E-4
1990	Arizona	8.620134228187919E-4
1990	Maine	9.789285714285713E-4
1990	Colorado	0.002162374100719...
1990	Mississippi	0.002666666666666...

2.4 Question 4

Up until now all the questions aboarded had been similar to previous examples worked/talked in class, but question 4 asked for something more specific, the average distance of the state's monitors to the state center. To answer this question, we will have to use the secondary Data Set containing the latitude and longitude limits for each state which

allows us to get the state center's coordinates by doing a simple linear interpolation since we are approximating each state's area to a rectangle. Once again we start by uniquely indentifying our monitors, then we check to see if the *State Name* matches between the two Data Sets and use the *latitude* and *longitude* coordinates alongside the state's center coordinates to calculate the distance[2]. The final step will be to group the data by state and calculate the average of the distance to the state center. This question required us for the first time to read and use two log files and allowed us to learn how to work with two different tables of data, it was particularly confusing how to do that for map-reduce at first but eventually our group arrived at the conclusion to import the data into a python dictionary since it was a small Data Set. The following pseudo-code explains the strategy adopted:

```
map1(Key, Values):
((FIPS, Site Num), Coordinates);
map2(Key, Values):
(State Name, Center Coordinates);
for each distinct monitor in map1:
    Emit(monitor);
where State Name matches:
    Emit(State Name, Distance);
for each State Name do:
    Emit(State Name, Average(Distance));
```

2.4.1 Produced Index and Execution Times

Execution times were registered as follows:

- Map-Reduce: 35.9820 sec
- Spark: 0.9140 sec
- SparkDF: 1.481 sec
- SparkSQL: 1.3290 sec
- Hive: 5.970 sec

Produced Index sample:

```
+--------------+----------------------+
|         State|Avg_Dist_Monitor_Center|
+--------------+----------------------+
|          Utah|     184.91876919510136|
|        Hawaii|     155.72848584906387|
|     Minnesota|     195.06726533715846|
|          Ohio|     175.74265339396345|
|      Arkansas|     151.13713852810923|
|        Oregon|      270.4530697644639|
|         Texas|      512.0338063321824|
|  North Dakota|     248.43249153972548|
|  Pennsylvania|     250.65578227993026|
|   Connecticut|      49.99224954412992|
|      Nebraska|     307.13572198974225|
|       Vermont|      521.9872635614976|
|        Nevada|     325.85331502151854|
|   Puerto Rico|     32.733405599511656|
|    Washington|     223.06324162734512|
|      Illinois|     435.24508277080764|
|      Oklahoma|     236.71168497753382|
+--------------+----------------------+
```

2. To calculate the distance we assume that each degree of latitude or longitude equals 111km as suggested by the Professor.

2.5 Question 5

For the 5th and final question, we were given the task of identifying the number of sensors per quadrant in each state by approximating each state's area to a rectangle that would then be divided in 4 quadrants related to the 4 intercardinal directions (NW, NE, SE or SW). To solve this question we will use the exact same data we used on question 4 but instead of calculating the distance between coordinates we will use the State's center coordinates to determine the intermediate longitude and latitude values of each state's area, resulting in the division in 4 quadrants. To determine whether a state belonged or not to a certain quadrant, each monitor's latitude and longitude values found in the main data set were compared to the intermediate values mentioned before. Finally, the number of sensors in each state's quadrant was calculated by grouping all unique monitors with the same state name and quadrant and counting the number of said monitors. The pseudo-code that represents these processes can be defined as follows:

```
map1(Key, Values):
((FIPS, Site Number), Coordinates);
map2(Key, Values):
(State Name, Center Coordinates);
for each distinct monitor in map1:
    Emit(monitor);
for each monitor:
    if lat < center lat and lon < center lon
    then Quadrant = NW
    if lat < center lat and lon > center lon
    then Quadrant = NE
    if lat > center lat and lon < center lon
    then Quadrant = SW
    if lat > center lat and lon > center lon
    then Quadrant = SE
for each tuple:
    Emit(State Name, count(Quadrants));
```

2.5.1 Produced Index and Execution Times

Execution times were registered as follows:

- Map-Reduce: 35.6380 sec
- Spark: 0.974 sec
- SparkDF: 1.2040 sec
- SparkSQL: 1.5860 sec
- Hive: 4.306 sec

Produced Index sample:

```
+--------------+--------+------------+
|         State|Quadrant|Num_Monitors|
```

Utah	SW	6
Utah	NE	3
Utah	NW	3
Hawaii	SW	2
Hawaii	SE	2
Hawaii	NE	1
Minnesota	NW	21
Minnesota	SE	11
Minnesota	NE	50
Minnesota	SW	12
Ohio	SE	30
Ohio	NE	10
Ohio	SW	15
Ohio	NW	34
Arkansas	SW	3
Arkansas	NW	4
Arkansas	SE	2

3 Conclusion

This project allowed us to consolidate all of the practical work we did during the semester as well as some theoretical knowledge regarding the used technologies for processing Big Data. During our work, we were able as a group to work together towards overcoming any problems with the logic and result consistency of our programms as well as figuring out the different pros and cons of the technologies used from a practical point of view(ease of use, readability, execution time). We concluded that map-reduce with hadoop ends up taking a lot of time to code for very specific questions and execution time is slow even though the results show up fast. Spark(RDD, DataFrames, SQL) and Hive are much more suited to work with for more complex problems. RDD's are very versitile due to the possible use of *lambda* functions but DataFrames are usually better to use when you are short on time since they help with optimizing the code which is a big help for longer pieces of code. The integration of SQL functions into Spark definitly helps with readibility and ease of use(due to the user-friendly and straightforward syntax of SQL). Hive and Spark SQL end up being similar when it comes to writting the code but Hive has its own advantages from a theoretical prespective such as its Data sharding method which allows for a selective replication factor when it comes to storing data as well as the fact it was built on top of hadoop to make use of the map-reduce primitives as proposed by the Facebook Data Infrastructure Team [3]. Lastly, this work was also a great first experience for our group to get a first impression on writting papers following the IEEE model and thus preparing us for our future work as professionals.

References

[1] João Lourenço, *Big Data Processing Systems - 2021/22 Project Assignment*. NOVA SST, November 17, 2021.

[2] (2021, Dec) Group's Project GitHub Link. [Online] Available: https://github.com/j-navesx/spbd.

[3] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff and Raghotham Murthy, *Hive - A Warehousing Solution Over a Map-Reduce Framework*. Facebook Data Infrastructure Team, August 2009.