**Assignment 5**

Wanyi Su

Student # 301445656

**Q1.**

Graphical user interface

Description automatically generated

**Q2.**

a).

What fraction of the input file was prefiltered by S3 before it was sent to Spark?

As the screenshots show below, input size is 97.7 KiB after S3 select prefiltering compared to 2.6MiB before it was added to the code.

Fraction that after filtering: 97.7 KiB / 2.6 MiB = 3.67%

The fraction being filtering away: 1- 3.67% = 96.33%

Since S3 Select can execute SQL queries directly on data stored in Amazon S3 bucket and retrieve the subset of data for our need, input file was prefiltered by S3 when reading the csv file before it was sent to Spark.

b).

Comparing the different input numbers for the regular version versus the prefiltered one, what operations were performed by S3 and which ones performed in Spark?

As the screenshots show below, input size is 97.7 KiB after S3 select prefiltering compared to 2.6MiB before it was added to the code. The amount of input data is greatly decreased.

S3 reads the data, performs the **filter** (weather.filter((weather['qflag'].isNull()) & (weather['station'].startswith('CA')) & (weather['observation'] == 'TMAX') )) and **select** function (weather.select('station', 'date', (weather['value']/10).alias('tmax'))) in code for input, and then send the filtered data to Spark. Among this **select** clause, the “weather['value']/10” computation part is done by Spark and send back to S3.

According to our code, these parts of input are prefiltered (which means included in input sent to Spark):

1. field ‘qflag’ (quality flag) that is null
2. field ‘station’ which starts with 'CA'
3. field ‘observation’ that is 'TMAX'

Spark implements calculation of dividing the temperature by 10 and writes the final output to the output file.

Before S3 filtering:

Graphical user interface, text, application

Description automatically generated

After S3 filter:

Graphical user interface, text, application

Description automatically generated

**Q3.**

a).

Reviewing the job times in the Spark history, which operations took the most time? Is the application IO-bound or compute-bound?

We spent the most of time on IO-bound. On local cluster, the time is about 14.7 mins (8.9min+5.8min) in input reading and output writing stages (job 0 and 3 here). (see screenshots below)

On local cluster, SortBy process in writing process costs the most time. While on EMR cluster, reduceByKey in reading stage costs the most time. In general, read/write uses the most time.

The compute bound costs a time of 11.9 mins (6.2m+5.7m), which is lower than IO-bound time.

The situation is similar running on EMR cluster.

*I/O bound*: a condition where the time spending to complete a computation is determined primarily by the time waiting for input/output operations to be completed.

*Compute bound (CPU bound):* the time for it to complete a task is determined principally by the speed of the central processor.

On local cluster:

Graphical user interface, text, application, chat or text message

Description automatically generated

Table

Description automatically generated

On EMR cluster:

Text, application

Description automatically generated with medium confidence

b).

Look up the hourly costs of the m6gd.xlarge instance on the EC2 On-Demand Pricing page. Estimate the cost of processing a dataset ten times as large as reddit-5 using just those 4 instances. If you wanted instead to process this larger dataset making full use of 16 instances, how would it have to be organized?

Graphical user interface, application

Description automatically generated

Since the dataset is 10 times larger than reddit-5 and we spend 4.3min running it on EMR cluster, the cost can be 4.3 / 60 \* 0.1808 \* 4 \* 10 = $0.5183

Since now we have 16 instances and each with vCPU of 4, we can repartition the RDD into 64 partitions or more (select a reasonable range, for example, 64 to 128 partitions) to facilitate the running process.