**Assignment 6**

Wanyi Su

Student# 301445656

**Q1.**

1. In the Reddit averages execution plan, which fields were loaded? How was the average computed (and was a combiner-like step done)?

Loaded fields: “score”, “subreddit”.

How avg computed:

Spark does partitioning on key/value pairs and then does HashAggregate to perform average computation to each key in each partition (a combiner-like step). After this, it exchanges the results from last steps (like a shuffling step) and then use HashAggregate to do total average calculation (a reducer-like step).

The partial averages are computed (“partial\_avg(score)” in HashAggregate) before shuffling and an overall average (“avg(scores) in ”HashAggregate) is calculated in the end.

**Q2.**

1. What was the running time for your Reddit averages implementations in the five scenarios described above? How much difference did Python implementation make (PyPy vs the default CPython)? Why was it large for RDDs but not for DataFrames?

|  |  |
| --- | --- |
|  | Time |
| MapReduce | 3m19.659s |
| Spark DataFrames (with CPython) | 1m45.245s |
| Spark RDDs (with CPython) | 2m32.373s |
| Spark DataFrames (with PyPy) | 1m51.194s |
| Spark RDDs (with PyPy) | 2m9.317s |

Time difference for Spark DataFrames (with CPython) and Spark DataFrames (with PyPy) is about 6s, while that for Spark RDD (with CPython) and Spark RDD (with PyPy) is almost 23s.

I don’t quite get what specific the last question is asking for, so I provide two perspectives here:

* If it is asking “Why boosting time difference (PyPy vs. CPython) is larger for RDDs but not for DataFrame (not consider RDD and DataFrame themselves)”:

We know Spark is implemented in Scala on JVM.

As the class note mentions, Python RDD is always opaque to the underlying Scala/JVM code: they are just serialized Python objects, and all work on them was done by passing the data back to Python code. There is some overhead for the compile and communication since Spark is implemented in Scala on JVM. Therefore, PyPy, which is a just-in-time compiler, can make up the fact that Python lagging behind its JVM counterparts and boost the running time for RDDs.

Spark DataFrames contain JVM (Scala) objects, and all manipulation is done directly in the JVM. There is little overhead for communication and compiling. So, DataFrame implementation with or without PyPy doesn’t differ too much.

* If it’s asking “Why running time is larger for RDDs but not for DataFrame (not consider PyPy)”:

In Spark, DataFrame provides an easy API to perform aggregation operations and it is faster than RDDs in performing simple operations (for example, grouping and aggregation). RDD needs to initialize and later drop row objects, which is costly and wastes more time than DataFrame does.

**Q3.**

1. How much of a difference did the broadcast hint make to the Wikipedia popular code's running time (and on what data set)?

On dataset pagecounts-3:

W/ broadcast: 1m19.996s

W/o broadcast hint: 2m11.562s

The difference is almost 1 minute. Broadcast boosts the running time for almost 50%.

**Q4.**

1. How did the Wikipedia popular execution plan differ with and without the broadcast hint?

With broadcast, the physical plan has a step of BroadcastExchange HashedRelationBroadcastMode. In the join stage, there is a BroadcastJoin step.

Without broadcast, the physical plan has no broadcast exchange step and only has sort. In join step, it uses SortMergeJoin instead.

**Q5.**

1. For the weather data question, did you prefer writing the “DataFrames + Python methods” style, or the “temp tables + SQL syntax” style for solving the problem? Which do you think produces more readable code?

I would prefer the SQL-syntax style, since it is SQL-like and more intuitive and readable. We just need to creates or replaces a local temporary view with this [DataFrame](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) by adding createOrReplaceTempView() clause after each dataframe creation (if it would be used thereafter), while mainly focus on following SQL syntax and write SQL codes in the spark.sql() function.

From my point of view, SQL-syntax style is more readable.