**Notes**

**Reddit\_average\_df.py**

Table

Description automatically generated Text

Description automatically generated

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:

There’s a combiner-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.

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.

Why large for RDDs:

*Spark output file:*

0 in some output files:

We did groupby and reducebykey after shuffling, so some output files can have no output records.

*different output partitions each time:*

#partition is not necessarily equal to #executors.

for Spark DataFrame, the partition is set to 200 automatically. But the output only shows the files with record# > 0. For Spark RDD, here we get an input dataset in 16 partitions, so we have 16 output files here.

MapReduce: # of reducers decide # of output files.

Spark: no specified “reducer”, but has reducer-like step. # partitions (related to # executors) decide # output files.

**Wikipedia**

Both are doing the same things -> adding a new column.

Both create a new dataframe.

*df.withColumn(‘new\_col\_name’, func)*

*df.select(‘a\_col’, ‘b\_col’, …, func.alias(‘new\_col\_name’))*

*df.groupby(col).max()*

Will return the col with a new column (max here), but no other columns in the df.

For wordcount-1:

49 items output

A picture containing text, receipt

Description automatically generated

**Q3.**

**W/ broadcast:**

1m19.996s

time spark-submit wikipedia\_popular\_df.py /courses/732/pagecounts-3 output-3

Graphical user interface, text

Description automatically generated 110 items output

Text

Description automatically generated

**W/o broadcast hint:**

2m11.562s

time spark-submit --conf spark.sql.autoBroadcastJoinThreshold=-1 wikipedia\_popular\_df.py /courses/732/pagecounts-3 output-2

Text

Description automatically generated110 items output

Text

Description automatically generated

**Q4.**

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

**Q5.**

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