ECE4721J - Lab 4 Report

Methods and Tools for Big Data

Yiding Chang, Yifan Shen, Kexuan Huang, Qinhang Wu

June 27, 2022

Ex.2 Simple Drill queries

2.1 Generate a file of size at least 5 GB, and copy it onto the Hadoop cluster.

A file with 200,000,000 records of students' grades with a size of 5.4 GB is generated.

2.2.1 Determine the name of the student who had the lowest grade

```sql
SELECT name, MIN(CAST(score AS INTEGER)) AS minScore FROM (SELECT
    COLUMNS[0] AS name, COLUMNS[2] AS score FROM hdfs.root.`/user/root/
    input/grades_200000000.csv`)  GROUP BY name ORDER BY minScore LIMIT
    1;
```

2.2.2 Determine the name of the student who had the highest average score

```sql
SELECT name, AVG(CAST(score AS INTEGER)) AS avgScore FROM (SELECT
    COLUMNS[0] AS name, COLUMNS[2] AS score FROM hdfs.root.`/user/root/
    input/grades_200000000.csv`) GROUP BY name ORDER BY avgScore DESC
    LIMIT 1;
```

2.3 Calculate the median over all the scores

```sql
SELECT AVG(CAST(score AS INTEGER)) FROM (SELECT COLUMNS[2] AS score
    FROM hdfs.root.`/user/root/input/grades_200000000.csv`) ORDER BY
    score LIMIT 2 OFFSET 99999999;
```

EX.3 Simple Spark

When installing and configuring Spark, we tried and failed to configure the cluster mode after digging into the problems for a few days. As a result, we use client mode instead. The differences between the client mode and the cluster mode are:

> In the client mode, the driver process runs on the client node where the job was submitted. The client node provides resources, such as memory, CPU, and disk space to the driver program, but the executors run on the cluster nodes and are maintained by YARN.

> In the cluster mode, driver process runs on one of the cluster workers machines, and YARN is responsible for both driver and executor processes.

> As a result, running multiple applications at the same time becomes possible because cluster managers will distribute the driver load across the cluster.

Production Spark jobs should always be ran in cluster mode. In our case, as all of the three machines we use are the servers on campus and we only run one application at a time, we expect a minimum difference between the two modes. Also, according to the tutorials “Spark Modes of Deployment – Cluster mode and Client Mode”:

> When job submitting machine is within or near to “spark infrastructure”. Since there is no high network latency of data movement for final result generation between “spark infrastructure” and “driver”, then, this mode works very fine.

Therefore, we chose to use client mode instead. Moreover, the connection between spark and hdfs failed so we can’t directly use the files on the hdfs as `get_rdd_hdfs()` function suggests, instead, we use local files. Finally, we test the program with the command:

```
spark-submit --master yarn --deploy-mode client --driver-memory 1g --executor-memory 1g --conf spark.driver.port=28888 --conf spark.blockManager.port=38888 /home/s/lab4/ex3.py
```

The spark program is written in python shown below:

```python
from pyspark import sql
from pyspark import SparkConf, SparkContext


def get_rdd_hdfs():
    spark = sql.session.SparkSession.builder.master("yarn").appName("ex3").getOrCreate()
    return spark.read.text("hdfs://hadoop-master:9000/input/lab2/grades_1000.csv").rdd

def get_rdd_local():
    conf = SparkConf().setMaster("yarn").setAppName("ex3")
    sc = SparkContext(conf=conf).getOrCreate()

    with open("data/grades_1000.csv", "r") as f:
        data = f.read().splitlines()

    return sc.parallelize(data)

def mapReduce(rdd):
    # map
    pairs = rdd.flatMap(lambda r: [r.split(",")[1:3]])

    # reduce
    max_grade = pairs.reduceByKey(lambda x, y: max([x, y]))

    for entry in max_grade.collect():
        print("{}\t{}".format(entry[0], entry[1]))


if __name__ == '__main__':
    rdd = get_rdd_local()
    #rdd = get_rdd_hdfs()
    mapReduce(rdd)
```

As file size increases, the table of speed is as follows,

File Size	# of Records	Speed on MapReduce (s)	Speed on pyspark(s)
29 KB	1000	2.6	42.33
287 KB	10000	20.3	42.43
2.8 MB	100000	22.0	39.63
28.7 MB	1000000	23.3	41.95
286.9 MB	10000000	44.0	45.33
2.87 GB	100000000	99.3	NA

Note that due to the RAM of our servers are limited, spark failed to handle the last file. Regarding why the speed on pyspark is similar even though the file size grows, we think the reason lies in we use `get_rdd_local()` instead of `get_rdd_hdfs()` with the client mode. As a result, reading from the log, a majority of time is reading inputs from local, sending it across the workers, and uploading many resources to hdfs.

If we can use `get_rdd_hdfs()` with the cluster mode, we expect the pyspark to be slower than MapReduce when the data size is small, for example, smaller than 50 MB. However, when data size is between several hundred MB and a few GB, we expect pyspark to have better performance. When the data size is really really big, then, it's hard to say as we can't test it. According to what's said in class, spark can fail, but mapreduce might not.