06 JUNE 2022 – MONITOR DATA CENTERS

MAPREDUCE & HADOOP

```
class Mapper extends org.apache.hadoop.mapreduce.Mapper<
                    LongWritable, // Input key type
                    Text,         // Input value type
                    Text,         // Output key type
                    Text> {// Output value type

    protected void map(
            LongWritable key,   // Input key type
            Text value,         // Input value type
            Context context) throws IOException, InterruptedException {

            String[] fields = value.toString().split(",");
            String year = fields[1].split("/")[0];
            String codDC = fields[0];
            double pwrCons = Double.parseDouble(fields[2]);

            if(year.equals("2020") || year.equals("2021"))
                if(pwrCons >= 1000)
                    context.write(new Text(codDC), new Text(year));
    }
}


class Reducer extends org.apache.hadoop.mapreduce.Reducer<
                Text,           // Input key type
                Text,    // Input value type
                Text,           // Output key type
                NullWritable> {  // Output value type

    @Override
    protected void reduce(
            Text key, // Input key type
            Iterable<Text> values, // Input value type
            Context context) throws IOException, InterruptedException {

        int count2020 = 0;
        int count2021 = 0;

        for(Text year : values) {
            String val = year.toString();
            if(val.equals("2020"))
                count2020++;
            else
                count2021++;
        }
        if(count2021 > count2020)
            context.write(key, NullWritable.get());
    }
}
```

SPARK

```
import pyspark

from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext

# companiesPath = "data/Companies.txt" # Useless for this program
dataCenterPath = "data/DataCenters.txt"
dailyPwrConsPath = "data/DailyPowerConsumption.txt"

outputPath1 = "outPart1/"
outputPath2 = "outPart2/"

# Define the rdds associated with the used input files
# CodDC,CodC,City,Country,Continent
dataCenterRDD = sc.textFile(dataCenterPath).cache()
# CodDC,Date,kWh
pwrConsRDD = sc.textFile(dailyPwrConsPath)

#########################################
# PART 1
#########################################

# Count the total number of data centers worl-wide and compute the threshold (90% of the data centers)
threshold = int(dataCenterRDD.count() * 0.9)

# Filter only the dates associated with high power consumption (>=1000 KWh)
#
# Prepare a pairRDD with
# key = date
# value = +1
#
# Finally, to count the number of data centers with high power consumption for each day/date.
highPwrConsDCPerDay = pwrConsRDD.filter(lambda line: float(line.split(",")[2]) >= 1000)\
                    .map(lambda line: (line.split(",")[1], 1))\
                    .reduceByKey(lambda v1, v2: v1 + v2)
```

```
# Select the dates with at least 90% of the data centers associated with a high
power consumption
res1 = highPwrConsDCPerDay.filter(lambda t: t[1] >= threshold)\
                            .keys()

res1.saveAsTextFile(outputPath1)

#########################################
# PART 2
#########################################

# Consider the power consumptions and keep only the entries related to year 2021
# and obtain the following pairRDD
# key = codDC
# value = kWh
# and use a reduceByKey to sum the power consumption for the entire year for
# each data center

def mapCodDCpwrCons(line):
    fields = line.split(",")
    codDC = fields[0]
    pwrCons = float(fields[2])
    return (codDC, pwrCons)


yearlyPwrCons = pwrConsRDD.filter(lambda line:
line.split(",")[1].startswith("2021"))\
                        .map(mapCodDCpwrCons)\
                        .reduceByKey(lambda v1, v2: v1 + v2)

# for each data center, keep the continent information
# key = codDC
# value = continent
def mapCodDCContinent(line):
    fields = line.split(",")
    codDC = fields[0]
    continent = fields[4]
    return (codDC, continent)


dcAndContinent = dataCenterRDD.map(mapCodDCContinent)

# Join yearlyPwrCons with dcAndContinent and
# returns pairs
# key = continent
# value = (+1, kWhPerDataCenter2021)
continentOnePwr = yearlyPwrCons.join(dcAndContinent)\
                .map(lambda t: (t[1][1], (1, t[1][0])))

# Sum the value parts to compute for each continent
# the number of data centers and the total power consumption in the year 2021.
# key = continent
# value = (the number of data centers, avg power consumption in the year 2021)
#
# Finally, compute the avg power consumption for each continent
numDCandAvgPwrCons = continentOnePwr\
            .reduceByKey(lambda t1, t2: (t1[0] + t2[0], t1[1] + t2[1]))\
            .mapValues(lambda t: (t[0], t[1]/t[0])).cache()

# compute the maximum number of data centers and the maximum avg consumption in
# the 2021 among the continents
maxDCAndConsumptionPerContinentThresholds = numDCandAvgPwrCons\
            .values()\
            .reduce(lambda t1, t2: (max(t1[0],t2[0]), max(t1[1],t2[1])))

# select only those continents for which both constraints are satisfied
res2 = numDCandAvgPwrCons\
            .filter(lambda t:
(t[1][0]==maxDCAndConsumptionPerContinentThresholds[0] and \
                              t[1][1]==maxDCAndConsumptionPerContinentThresholds[
1]))\
            .keys()

res2.saveAsTextFile(outputPath2)
```