Exam 12 September 2024 - Spark

PoliSales is an international e-commerce company that asked you to develop two applications:

- A MapReduce program (Exercise 1)
- A Spark-based program (Exercise 2)

to address the analyses they are interested in. The applications can read data from the input files described below.

Products.txt

- `Products.txt` is a textual file containing information about the Products that are sold by PoliSales.
- There is one line for each Product and the total number of distinct Products is greater than 5,000,000.
- This file is large, and you cannot suppose the content of `Products.txt` can be stored in one in-memory Java variable.
- Each line of `Products.txt` has the following format:

  ```
  ProductID,Name,Category
  ```

  where `ProductID` is the unique identifier of the Product, `Name` is the name of `ProductID`, and `Category` is its category.
- For example, the following line:

  ```
  ID1,Galaxy S24 256GB Black,Smartphone
  ```

  means that the Product with `ProductID` ID1 is characterized by the name "Galaxy S24 256GB Black" and belongs to the "Smartphone" category. Note that many Products can be associated with the same category.

Prices.txt

- `Prices.txt` is a textual file containing information about the prices of the Products.
- The price of each Product varies over time. There are potentially multiple lines for each Product.
- This file is large, and you cannot suppose the content of `Prices.txt` can be stored in one in-memory Java variable.
- Each line of `Prices.txt` has the following format:

```
ProductID,StartingDate,EndingDate,Price
```

where `ProductID` is the identifier of a Product, and `StartingDate` and `EndingDate` are the beginning and end of the period of validity of the price reported in the attribute `Price` for Product `ProductID`.

- For example, the following line:

```
ID1,2021/01/31,2022/12/31,98.7
```

Note that the price of each Product varies over time. Every time there is a price variation, a new line is inserted in `Prices.txt` with information about the new price and its validity period. Each Product is associated with a single price in each period.

Sales.txt

- `Sales.txt` is a textual file containing information about daily sales for each Product.
- `Sales.txt` contains historical data about the last 30 years. This file is big, and its content cannot be stored in one in-memory Java variable.
- There is one line for each combination (ProductID, Date) for the last 30 years.
- Each line of `Sales.txt` has the following format:

```
ProductID,Date,NumberOfProductsSold
```

where `ProductID` is the identifier of a Product, `Date` is a date, and `NumberOfProductsSold` is an integer value representing the number of times `ProductID` was sold on that `Date`.

- For example, the following line:

```
ID1,2021/04/30,1234
```

means that on April 30, 2021, the Product identified by `ID1` was sold 1234 times. The format of `Date` is “YYYY/MM/DD”.

- Note that there is a many-to-many relationship between Products and Dates (i.e., the combination of attributes (ProductID, Date) is the "primary key" of `Sales.txt`). Each Product is associated with all the dates of the last 30 years, and each date of the last 30 years is associated with all Products. Even if a Product was not sold on a specific date, there is a line for that combination in `Sales.txt` anyway, with `NumberOfProductsSold` set to 0 (zero).

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliSales asked you to develop a single Spark-based application based either on RDDs or Spark SQL to address the following tasks. The application takes the paths of the three input files and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. **Products that decreased their total sales in 2021 with respect to the sales in 2019.**
    - The first part of this application considers only the years 2019 and 2021.
    - It selects the Products with a total number of products sold in 2021 lower than the total number of products sold in 2019.
    - Store the selected Products in the first output folder (one product per output line).
2. **Most sold product(s) for each year.**
    - The second part of this application considers all input data.
    - It selects, for each year, the product(s) associated with the highest total annual sales for that year.
    - For each product and year, the total annual sales for that product is given by the sum of the number of products sold in all days of that year.
    - Store the result in the second output folder, as pairs (year, ID of the most sold product for that year).

Part 1: Products that Decreased Sales in 2021 Compared to 2019

Steps

1. **Filter the Sales Data by Year**:
   We first filter the `Sales.txt` data to keep only the records for the years 2019 and 2021.
2. **Map Sales Data**:
   We map the filtered sales data into a key-value pair:
    - Key: `product_id`
    - Value: A tuple `(sales_in_2019, sales_in_2021)`, where sales in the other year are set to 0 initially.
3. **Reduce by Key**:
   We use `reduceByKey` to sum up the sales for each product in both years.
4. **Filter for Decreased Sales**:
   After calculating the total sales for 2019 and 2021 for each product, we filter the products where sales in 2019 were greater than in 2021.
5. **Save the Results**:
   We save the results to an output directory `out1/`.

Part 2: Most Sold Products for Each Year

Steps

1. **Map Sales Data**:
   We create a key-value pair from the `Sales.txt` data:
   - Key: `(product_id, year)`
   - Value: `number_of_products_sold`
2. **Sum Sales by Product and Year**:
   We use `reduceByKey` to sum the sales for each product in each year.
3. **Find the Maximum Sales Per Year**:
   We map the result to find the maximum sales for each year.
4. **Filter Products with Maximum Sales**:
   We use a `join` to keep only the products that have the maximum sales for each year.
5. **Save the Results**:
   We save the results to an output directory `out2/`.

Code

```python
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName('Exam 12 Sept 2024')
sc = SparkContext(conf = conf)

products_path = 'data/Products.txt'
prices_path = 'data/Prices.txt'
sales_path = 'data/Sales.txt'

# Define the input rdds
# Products: product_id, name, category
productsRdd = sc.textFile(products_path)
# Prices: product_id, starting_date, ending_date, price
pricesRdd = sc.textFile(prices_path)
# Sales: product_id, date, number_of_products_sold
salesRdd = sc.textFile(sales_path)

# Part 1
# products that decreased their total sales in 2021 with respect to sales in
2019
# start with salesRDD and filter only year == 2019 or year == 2021
# then, compute the following RDD
# key = prodID
# value = #sales in 2019, #sales in 2021

def filterYears(line):
    fields = line.split(',')
    date = fields[1]

    return date.startswith('2019') or date.startswith('2021')

def mapProductSales(line):
    fields = line.split(',')
    pid = fields[0]  # product_id
    date = fields[1]
    numSales = int(fields[2])

    if date.startswith('2019'):
        return (pid, (numSales, 0))
    else:
        return (pid, (0, numSales))

salesPerYearRdd = salesRdd.filter(filterYears)\
                    .map(mapProductSales)\
                    .reduceByKey(lambda v1, v2: (v1[0]+v2[0], v1[1]+v2[1]))

# filter and keep only the entries associated with #sales19 > #sales 21
# retrieve the resulting productids
res1 = salesPerYearRdd.filter(lambda p: p[1][0]>p[1][1])\
                        .keys()

res1.saveAsTextFile("out1/")

##############################################################################

# Part 2
# compute the most sold products for each year

# compute the following pairRDD
# key = productID, year
# value = #sales

def mapPidYearSales(line):
    fields = line.split(",")
    pid = fields[0]
    year = fields[1].split("/")[0]
    numSales = int(fields[2])

    return ((pid, year), numSales)


salesPerYear = salesRdd.map(mapPidYearSales)

# use a reduceByKey to sum all sales within that year and cache the RDD
totalSalesPerYear = salesPerYear\
                .reduceByKey(lambda v1, v2: v1+v2).cache()

# determine, for each year, the maximum value
# by first doing a map to pairs
# key = year
# value = count
# and use a reduceByKey to compute the max for each year
maxPerYear = totalSalesPerYear.map(lambda p: (p[0][1], p[1]))\
                                .reduceByKey(lambda v1, v2: max(v1, v2))

# map maxPerYear to
# key = (year, max count per year)
# value = None
yearMaxNone = maxPerYear.map(lambda p: (p, None))

# for each product, keep only the pairs (product, year) associated with the max
of the year
# first, we transform totalSalesPerYear in
# key = (year, count)
# value = pid
#
# and then use a join to keep only the pairs (product, year) associated with
the maximum valur of the year
maxPidPerYear = totalSalesPerYear.map(lambda p: ( (p[0][1], p[1]), p[0][0])).\
                    join(yearMaxNone)

# Extract (year, pid)
res2 = maxPidPerYear.map(lambda p: (p[0][0], p[1][0]))

res2.saveAsTextFile("out2/")
```