2024 - 2025

ACTIVIDAD 2

Máster en Big Data y Data Science

03MBID – Procesamiento de Datos Masivos

Nombre: Gonzalo Antonio Delgado Rubio

Fecha: 22/06/2024

viu Universidad Internacional de Valencia

Problema 1

Dado un dataset que contenga entradas con la forma "persona;método_pago;dinero_gastado", crea un programa llamado personaGastosConTarjetaCredito que para cada persona indique la suma del dinero gastado con tarjeta de crédito, con el formato persona;gastoconTDC. Ejemplo:

Entrada	Salida
Alice;Tarjeta de crédito;100	Alice;250
Alice;Tarjeta de crédito;150	Bob;201
Alice;Bizum;200	Luis;0
Bob;Tarjeta de crédito;201	
Luis;Bizum;300	

Notar que Alice gasta en total 450 euros, pero sólo 250 son con tarjeta de crédito (100 + 150). Se valorará positivamente la eficiencia del programa, por ejemplo, no usar transformaciones innecesarias.

1.1 Solución

- Se Inicializa creando el contexto spark y seteando ando variables para el input y output de la data. (Se debe cambiar según ejecución)

```
from pyspark import SparkContext, SparkConf

# Crear un contexto de Spark
conf = SparkConf().setAppName("problema1")
sc = SparkContext(conf=conf)

# Constantes -- cambiar según requerimiento
PATH_INPUT = './problema1/casoDePrueba1.txt'
PATH_OUT = './problema1/output/personaGastosConTarjetaCredito'
```

- Se inicializa programa con función main, generando una lectura de los datos mediante un textFile
- Aplicando una función map separamos la información según el separador de archivo ';'
- Seleccionamos columnas y damos formato a columna de gasto
- Empleamos una función map para seleccionar la persona y gasto con TDC
- Finalmente empleamos función reduceByKey para agrupar datos.

```
# 1. lectura de datos
    input_rdd = sc.textFile(PATH_INPUT)

    # 2. Procesamiento de datos
    # 2.1 Convirtiendo datos en tupla
    tuple_rdd = (input_rdd
        .map(lambda line: line.split(";"))
        .map(lambda x: (x[0], x[1], float(x[2])))
    )
    # 2.2 Seleccionamos persona y gasto en caso este sea con TDC en otro
caso se coloca 0
    process_rdd = tuple_rdd.map(lambda x: (x[0], x[2] if x[1] == 'Tarjeta
de crédito' else 0))

    # 2.3 Sumarizamos los gastos por persona
    reduce_rdd = process_rdd.reduceByKey(lambda x, y: x + y)
```

- Por ultimo se genera función de escritura para almacenar datos en output path.

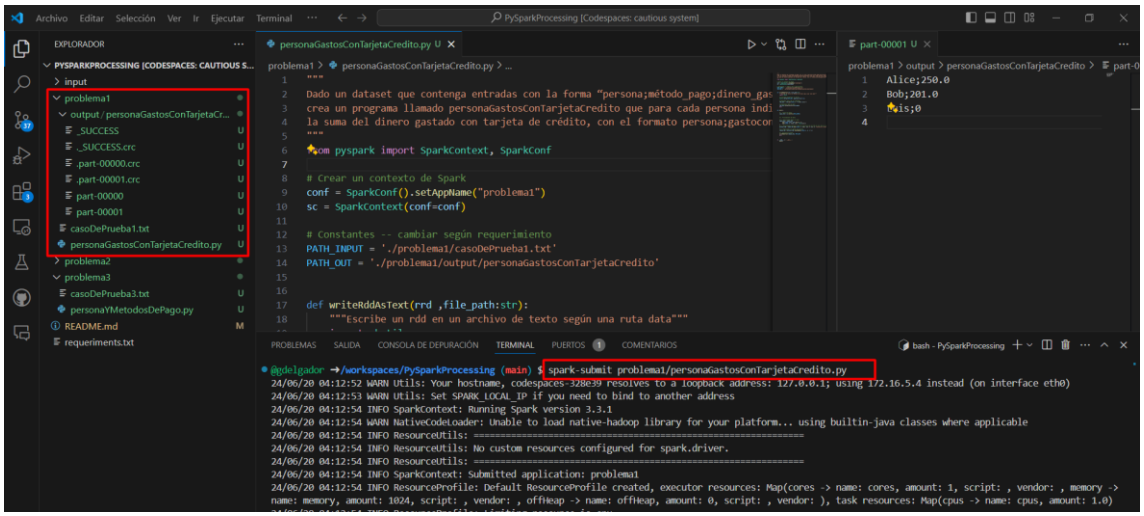
```
def writeRddAsText(rrd ,file_path:str):
    """Escribe un rdd en un archivo de texto según una ruta data"""
    import shutil

    # Eliminamos directorio si existe
    shutil.rmtree(file_path, ignore_errors=True)

    # Generamos nuevo rdd con separador
    rdd_save = rrd.map(lambda x: f"{x[0]};{x[1]}")

    # almacenamos data en path
    rdd_save.saveAsTextFile(file_path)
    pass
```

1.2 Ejecución de Programa

Se ejempla comando de ejecución desde el repositorio

```
spark-submit problema1/personaGastosConTarjetaCredito.py
```

1.2 Evidencia Ejecución


EXPLORADOR
PYSPARKPROCESSING [CODESPACES: CAUTIOUS S...
input
problema1
output / personaGastosConTarjetaCr...
_SUCCESS
_SUCCESS.crc
.part-00000.crc
.part-00001.crc
part-00000
part-00001
casoDePrueba1.txt
personaGastosConTarjetaCredito.py
problema2
problema3
casoDePrueba3.txt
personaYMetodosDePago.py
README.md
requeriments.txt
personaGastosConTarjetaCredito.py U
problema1 > personaGastosConTarjetaCredito.py > ...
Dado un dataset que contenga entradas con la forma "persona;método_pago;dinero_gas
crea un programa llamado personaGastosConTarjetaCredito que para cada persona indi
la suma del dinero gastado con tarjeta de crédito, con el formato persona;gastocor
from pyspark import SparkContext, SparkConf
# Crear un contexto de Spark
conf = SparkConf().setAppName("problema1")
sc = SparkContext(conf=conf)
# Constantes -- cambiar según requerimiento
PATH_INPUT = './problema1/casoDePrueba1.txt'
PATH_OUT = './problema1/output/personaGastosConTarjetaCredito'
def writeRddAsText(rrd ,file_path:str):
"""Escribe un rdd en un archivo de texto según una ruta data"""
part-00001 U
problema1 > output > personaGastosConTarjetaCredito > part-0
Alice;250.0
Bob;201.0
is;0
PROBLEMAS
SALIDA
CONSOLA DE DEPURACIÓN
TERMINAL
PUERTOS
COMENTARIOS
bash - PySparkProcessing
@gdelgador ➜ /workspaces/PySparkProcessing (main) $ spark-submit problema1/personaGastosConTarjetaCredito.py
24/06/20 04:12:52 WARN Utils: Your hostname, codespaces-328e39 resolves to a loopback address: 127.0.0.1; using 172.16.5.4 instead (on interface eth0)
24/06/20 04:12:53 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
24/06/20 04:12:54 INFO SparkContext: Running Spark version 3.3.1
24/06/20 04:12:54 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/06/20 04:12:54 INFO ResourceUtils: ==========
24/06/20 04:12:54 INFO ResourceUtils: No custom resources configured for spark.driver.
24/06/20 04:12:54 INFO ResourceUtils: ==========
24/06/20 04:12:54 INFO SparkContext: Submitted application: problema1
24/06/20 04:12:54 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory ->
name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor:), task resources: Map(cpus -> name: cpus, amount: 1.0)

Problema 2

Dado un dataset que contiene información sobre los videos de Youtube (https://netsg.cs.sfu.ca/youtubedata/), crear un programa llamado CategoriaDeVideosMenosVista que obtenga cuál es la categoría de videos menos vista de la plataforma Youtube y el número total de visualizaciones que hay en esa categoría. El programa debe recibir dos parámetros de entrada: la carpeta en la que está el dataset y la carpeta en la que se guardará el resultado. En la carpeta donde está el dataset se tienen que descomprimir UNO de los archivos 0222.zip, 0301.zip, etc., que se encuentran en el enlace anterior. Importante: si la persona que hace la actividad dispone de pocos recursos computaciones, entonces se recomienda que únicamente descomprima algún .zip pequeño para que pueda desarrollar el programa. La carpeta de datos de entrada debería quedar como se ve en la Figura 1. Los datos de entrada están en los archivos 0.txt, 1.txt, etc y cada fila contiene la información de un video tabulada con el siguiente formato: id del video de youtube, usuario que subió el video, número de días desde que se subió el video y la fecha en la que obtuvieron los datos, categoría del video, longitud del video, número de visitas del video, puntuación del video, número de puntuaciones del video, número de comentarios del video, y una lista de ids de videos relacionados. Se valorará positivamente la eficiencia del programa, por ejemplo no usar transformaciones innecesarias.

Ejemplo:

Entrada	Salida
... Gadgets & Games ... 30 ... Gadgets & Games ... 10 ... Music ... 90 ... Sports ... 20 ... Music ... 50 ... Gadgets & Games ... 95	Sports;20

Notar que la categoría “Sports” es la que menos visitas tiene: 20 en un único vídeo. “Music” es la que más visitas tiene: 90 en un video + 50 en otro video, es decir, en total 140 visitas, y la categoría “Gadgets & Games” tiene en total 135 visitas obtenidas de 30 + 10 + 95. El programa debe funcionar independientemente del número de categorías, para cualquier cantidad de filas que se pueda llegar a tener, para cualquier cantidad de ficheros e ignorar el log.txt.

2.1 Solución

- Se inicializa programa generando contexto spark
- Se genera programa de lectura de datos a partir de una ruta data, se descromprime archivo zipeado

- Se emplea método wholeTextFiles para lectura de los archivos .txt
- Se ignora el archivo 'log.txt'
- Se procede a realizar un flapMap para separar los datos en líneas por cada uno de los archivos leídos.

```
from pyspark import SparkContext, SparkConf
from zipfile import ZipFile
import os
import sys
import shutil

# Crear un contexto de Spark
conf = SparkConf().setAppName("problema2")
sc = SparkContext(conf=conf)


def read_rdd_from_dataset(dataset_path:str):
    """Lectura de un rdd desde un archivo zip"""

    folder_path = os.path.dirname(dataset_path)
    filename = os.path.basename(dataset_path).split('.')[0]

    # 1. Eliminamos la carpeta data si existe
    shutil.rmtree(os.path.join(folder_path, 'data', filename))

    # descomprimiendo el archivo
    with ZipFile(dataset_path, 'r') as zip_ref:
        zip_ref.extractall(os.path.join(folder_path, 'data'))


    # lectura de los archivos, eliminando el archivo log.txt
    rdd = sc.wholeTextFiles(os.path.join(folder_path,'data', filename))
    rdd = rdd.filter(lambda x: "log.txt" not in x[0])

    # separando por lineas
    rdd_lines = rdd.flatMap(lambda x: x[1].splitlines())

    return rdd_lines
```

- Se separa la información por el separador del archivo 'tabulador'
- Se eligen las columnas de interés, según posición: categoría y nro de visitas

```
    # 4. Lectura manteniendo columnas de interes: categoria y nro de
    visitas
        select_rdd = (
            input_rdd
```

```
            .map(lambda line: line.split("\t"))
            .filter(lambda fields: len(fields) > 5) # quito nulos
            .map(lambda fields: (fields[3], int(fields[5])))
        )
```

- Agrupo los datos para sumarizar el nro de visitas
- Finalmente ordeno y elijo el elemento con menor nro de visitas

```
# 5. Agrupo por categoria y sumo las visitas
    grouped_rdd = select_rdd.reduceByKey(lambda a, b: a + b)

    # 6. Encontrar la categoria con menos visitas -> retorna una lista
con la categoria y el nro de visitas
    min_category = grouped_rdd.takeOrdered(1, key=lambda x: x[1])
```

- Escribimos data en ruta especificada

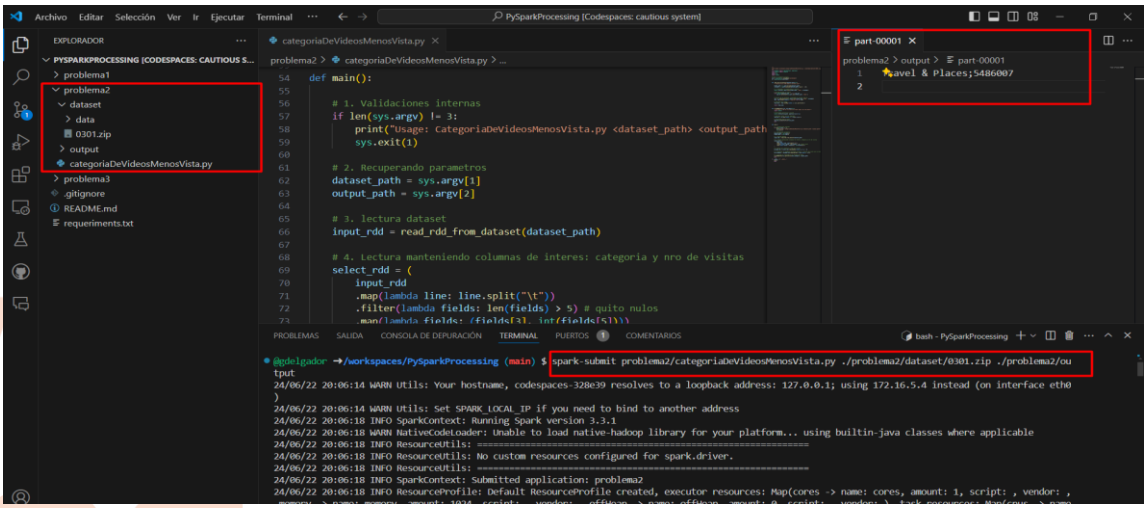
```
 # 7. Escribir el resultado en un archivo de texto
    writeRddAsText(sc.parallelize(min_category) ,output_path)
```

2.2 Ejecución Programa

Se ejempla comando de ejecución desde el repositorio

```
spark-submit problema2/categoriaDeVideosMenosVista.py
./problema2/dataset/0301.zip ./problema2/output
```

2.3 Evidencia Ejecución



Problema 3

Dado un dataset que contenga entradas con la forma "persona;método_pago;dinero_gastado", crea un programa llamado personaYMetodosDePago que:

a) Por cada persona indique en cuántas compras pagó más de 1500 euros con un medio de pago diferente a tarjeta de crédito. La solución se tiene que guardar en un archivo llamado comprasSinTDCMayorDe1500.
b) Por cada persona indique en cuántas compras pagó menos o igual a 1500 euros con un medio de pago diferente a tarjeta de crédito. La solución se tiene que guardar en un archivo llamado comprasSinTDCMenoroIgualDe1500.

Se valorará positivamente la eficiencia del programa, por ejemplo, no usar transformaciones innecesarias.

Ejemplo:

Entrada	Salida (a)	Salida (b)
Alice;Tarjeta de crédito;1000	Alice;0	Alice;1
Alice;Tarjeta de crédito;1800	Bob;1	Bob;0
Alice;Tarjeta de crédito;2100		
Bob;Bizum;2000		
Alice;Bizum;1000		
Bob;Tarjeta de crédito;1100		

Notar que Alice y Bob solo hacen una compra con pago diferente a tarjeta de crédito.

3.1 Solución

- Creamos nuestro context spark
- Agregamos las rutas del archivo input y output

```
from pyspark import SparkContext, SparkConf

# Crear un contexto de Spark
conf = SparkConf().setAppName("problema3")
sc = SparkContext(conf=conf)

# Constantes
PATH_INPUT = './problema3/casoDePrueba3.txt'
PATH_OUT = './problema3/output'
```

- Leemos la data con método textFile
- Separamos la información por separador ';' y generamos lista de tuplas de rdd

```
-  # 1. lectura de datos
-      input_rdd = sc.textFile(PATH_INPUT)
-
```

```
-       # 2. Procesamiento de datos
-
-       # 2.1 Convirtiendo datos en tupla
-       tuple_rdd = (input_rdd
-           .map(lambda line: line.split(";"))
-           .map(lambda x: (x[0], x[1], float(x[2])))
-       )
```

- Para las compras sin TDC mayores a 1500 generamos una función map que se encargue de realizar la tarea
- Finalmente agrupamos la información sumarizando los datos

```
def processComprasSinTDCMayorDe1500(rdd):
    """Procesa las compras sin TDC mayores a 1500"""

    # aplico funcion de mapeo
    process_rdd = rdd.map(lambda x: (x[0], 1 if x[1]!='Tarjeta de
crédito' and x[2]>1500 else 0))

    # agrupamiento
    group_rdd = process_rdd.reduceByKey(lambda x,y: x+y)
    return group_rdd
```

- Para procesar las compras sin TDC con montos menores e iguales a 1500 se emplea función maper
- Agrupamos y sumarizamos los datos

```
-   def processcomprasSinTDCMenoroIgualDe1500(rdd):
-       """Procesa las compras sin TDC menores o iguales a 1500"""
-
-       # aplico funcion de mapeo
-       process_rdd = rdd.map(lambda x: (x[0], 1 if x[1]!='Tarjeta de
    crédito' and x[2]<=1500 else 0))
-
-       # agrupamiento
-       group_rdd = process_rdd.reduceByKey(lambda x,y: x+y)
-       return group_rdd
```

- Se recupera rdd de las funciones y por último escribimos resultados en rutas definidas

```
 # 2.2 Procesando compras sin TDC mayores a 1500
    compras_rdd = processComprasSinTDCMayorDe1500(tuple_rdd)

    # 2.3 Procesando compras sin TDC menores o iguales a 1500
    reduce_rdd = processcomprasSinTDCMenoroIgualDe1500(tuple_rdd)

    # 3. Escritura de datos
    writeRddAsText(compras_rdd, f"{PATH_OUT}/comprasSinTDCMayorDe1500")
    writeRddAsText(reduce_rdd,
f"{PATH_OUT}/comprasSinTDCMenoroIgualDe1500")
```

3.2 Ejecución Programa

Se ejempla comando de ejecución desde el repositorio

```
spark-submit problema3/ personaYMetodosDePago.py
```

3.3 Evidencia Ejecución