UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

PRÁCTICA OBLIGATORIA PROGRAMACIÓN PARALELA: BiciMAD

Autores:

José Ignacio Alba Rodríguez
Álvaro Ezquerro Pérez
Alejandro Millán Arribas

Programación Paralela

2022-2023

1. Introducción

Esta práctica consiste en realizar un estudio sobre la base de datos de la información de uso del servicio *BiciMAD* porporcionada por el Ayuntamiento de Madrid. El objetivo principal es aplicar los métodos de Spark aprendidos en la asignatura para estudiar diferentes cuestiones sobre este conjunto de datos.

1.1. Problemática planteada

Lo que nos hemos propuesto es llevar a cabo un estudio sobre diferentes preguntas para, a partir de las conclusiones obtenidas, poder ofrecer distintos planes de mejora para el funcionamiente del servicio *BiciMAD*, como por ejemplo en qué momentos se requiere una mayor número de personal, qué estaciones sufren un mayor uso y por tanto un mayor desgaste a tener en cuenta o también a que público es más necesario hacer llegar la disponibilad de este servicio, entre otros.

Para llegar a todas estas medidas hemos deciddo intentar dar respuesta a las siguientes cuestiones:

1. Determinar los trayectos más realizados y los menos.
2. Cuáles son las estaciones más utilizadas y las que menos.
3. Determinar la hora punta de uso.
4. Calcular los porcentajes de uso dependiendo del rango de edad y el tipo de usuario.
5. Calcular la cantidad de bicicletas rotas, es decir, cuyo viaje consta de un tiempo bajo, digamos $\leq$60 segundos.
6. Intentar ver el número de clientes habituales que tiene este servicio.

1.2. Base de datos

Las bases de datos que nosotros hemos utilizado están en formato *.json*. Nuestro estudio lo hemos enfocado sobre los archivos $sample\_10e2.json$, $sample\_10e3.json$, $sample\_10e4.json$, que se corresponden con los datos de junio, julio y agosto del año 2019. Sin embargo, en la web del Ayuntamiento de Madrid. Los distintos tipos de datos de los que disponemos son:

- Tipo de usuario: cliente o personal de mantenimiento.
- Código de usuario.
- Número de la estación donde se desengacha la bicicleta.
- Número de la estación donde se enchancha la bicicleta.
- Número de la base de la que se desengancha la bicicleta.
- Número de la base en la que se engancha la bicicleta.

- Tiempo transcurrido entre el enganche y el desenganche de la bicicleta.
- Hora a la que se realiza el desenganche de la bicicleta. El fórmato es: "2017-09-22T14:00:00.000+0200".
- Rango de edad del usuario.
- Tipo de usuario: trabajador de empresa, usuario anual y usuario ocasional.

1.3. Tratamiento de los datos

Como ya hemos mencionado para el tratamiento de los datos lo realizaremos con la herramienta *Spark*. Para leer los archivos en formato textit.json gracias al paquete *json* de Python. Estos datos se guardan en una estructura RDD que es la que manejaremos a partir de ahora. El primer paso, para que sea más cómodo el manejo de los datos utilizamos la siguiente función, que nos permite manejar por separado cada uno de los distintos datos que hemos mencionado anteriormente.

```
    def mapper(line):
        data = json.loads(line)
        user_type = data['user_type']
        user_age = data['ageRange']
        user_day_code = data['user_day_code']
        start_station = data['idunplug_station']
        end_station = data['idplug_station']
        duration = data['travel_time']
        date = datetime.strptime(data['unplug_hourTime']['\$date'
           ][:-4],)
        return user_type, user_day_code, start_station, end_station,
            duration, date, user age
```

Una vez que hemos aplicado esta función ya podemos enfocarnos a responder cada una de las cuestiones que nos hemos planteado utilizando distintas metodologías.

2. Cuestiones planteadas

A continuación veremos como hemos abordado cada una de las cuestiones y que conclusiones hemos obtenido de cada una de ellas.

2.1. Determinar los trayectos más realizados y los que menos

La función que hemos diseñado para resolver esta cuestión es:

```
    def trayectos_habituales(rdd):
        trayectos_ordenados = rdd.map(lambda x: (x[2],x[3]) ).\
                              countByValue().\
                              sortBy(lamda x: x[1] , ascending =
                                 False)
```

```
    return trayectos_ordenados
```

Los resultados que hemos obtenido en cada uno de los ficheros son:

- $sample\_10e2 - json$:

```
    Las 5 rutas mas repetidas son:
    [(156, 5), (67, 83), (114, 96), (118, 42), (46, 90)]
    realizadas cada una este numero de veces:
    [3, 2, 2, 2, 2]
```

- $sample\_10e3 - json$:

```
    Las 5 rutas mas repetidas son:
    [(139, 141), (160, 161), (168, 131), (82, 102), (19, 90)]
    realizadas cada una este numero de veces:
    [9, 9, 8, 7, 5]
```

- $sample\_10e4 - json$:

```
    Las 5 rutas mas repetidas son:
    [(133, 135), (135, 35), (135, 132), (133, 133), (64, 64)]
    realizadas cada una este numero de veces:
    [11, 10, 10, 9, 9]
```

2.2. Cuáles son las estaciones más utilizadas y las que menos

La función que hemos diseñado para resolver esta cuestión es:

```
    def rutas_ordenadas(rdd):
        rutas_ordenadas = rdd.map(lambda x: ((x[2], x[3]), 1).\
            reduceByKey(lambda x, y: x+y).\
            sortBy(lambda x: x[1], ascending = False)
        return rutas_ordenadas
```

El resultado obtenido en cada uno de los ficheros es:

- $sample\_10e2 - json$:

```
    Estas son las 5 estaciones mas transitadas:
    [90, 135, 129, 169, 54]
    con este numero de usos cada una:
    [7, 7, 6, 6, 5]
    Ademas, estas son las 5 estaciones menos transitadas:
    [22, 28, 150, 29, 173]
    con este numero de usos cada una:
    [10, 27, 42, 43, 43]
```

- $sample\_10e3 - json$:

```
Estas son las 5 estaciones mas transitadas:
[57, 135, 90, 42, 129]
con este numero de usos cada una:
[61, 50, 39, 35, 34]
Ademas, estas son las 5 estaciones menos transitadas:
[146, 92, 112, 121, 11]
con este numero de usos cada una:
[1, 1, 1, 1, 1]
```

- *sample_10e4 – json*:

```
Estas son las 5 estaciones mas transitadas:
[57, 135, 163, 43, 129]
con este numero de usos cada una:
[294, 285, 265, 263, 258]
Ademas, estas son las 5 estaciones menos transitadas:
[22, 28, 150, 29, 173]
con este numero de usos cada una:
[10, 27, 42, 43, 43]
```

2.3. Determinar la hora punta de uso

La función que hemos diseñado para resolver esta cuestión es:

```
def horas_ordenadas(rdd):
    horas = rdd.map(lambda x: (x[5].hour, 1)).\
        reduceByKey(lambda x, y: x+y).\
        sortBy(lambda x: x[1], ascending = False)
    total = horas.map(lambda x: x[1]).sum()
    horas = horas.map(lambda x: (x[0], x[1]*100/total))
    return horas
```

El resultado obtenido en cada uno de los ficheros es:

- *sample_10e2 – json*:

```
Las horas ordenadas en cuanto a mayor uso son:
[0]
con porcentajes
[100.0]
```

- *sample_10e3 – json*:

```
Las horas ordenadas en cuanto a mayor uso son:
[0, 1]
con porcentajes
[59.4, 40.6]
```

- *sample_10e4 – json*:

```
Las horas ordenadas en cuanto a mayor uso son:
[19, 18, 14, 15, 20]
con porcentajes
[7.48, 7.33, 6.28, 6.28, 6.23]
```

2.4. Calcular los porcentajes de uso dependiendo del rango de edad

La función que hemos diseñado para resolver esta cuestión es:

```
def edades_ordenadas(rdd):
    edades = rdd.map(lambda x: (user_ages[x[0]], 1)).\
        reduceByKey(lambda x, y: x+y).\
        sortBy(lambda x: x[1], ascending = False)
    total = edades.map(lambda x: x[1]).sum()
    edades = edades.map(lambda x: (x[0], x[1]*100/total))
    return edades
```

El resultado obtenido en cada uno de los ficheros es:

- $sample\_10e2 - json$:

```
Las edades de los usuarios ordenadas en cuanto a mas uso
   son:
['<17']
con porcentajes
[100.0]
```

- $sample\_10e3 - json$:

```
Las edades de los usuarios ordenadas en cuanto a mas uso
   son:
['<17', '19-26', '17-18']
con porcentajes
[95.4, 3.6, 1.0]
```

- $sample\_10e4 - json$:

```
Las edades de los usuarios ordenadas en cuanto a mas uso
   son:
['<17', '19-26', 'Sin datos', '17-18']
con porcentajes
[87.55, 9.25, 2.01, 1.19]
```