Sistemas de Gestión de Datos y de la Información
Máster en Ingeniería Informática, 2022-23
Práctica MapReduce - Apache Spark

Fecha de entrega: miércoles 16 de noviembre de 2022, 23:55h

Entrega de la práctica
La práctica se entregará en un único fichero **ZIP** mediante el Campus Virtual de la asignatura. El fichero estará organizado en carpetas por cada apartado. Dentro de estas carpetas habrá un fichero `.py` por cada tarea resuelta.

Lenguaje de programación
3.9 o superior

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *Declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con otras personas. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.*

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

mrjob en Linux

En los laboratorios ya está instalado, pero si queréis usar *mrjob* en vuestro ordenador podéis instalarlo con:

```
$ pip install mrjob
```

Las tareas MapReduce de *mrjob* se definen implementando una clase que herede de `MRJob` y definiendo sus métodos `mapper`, `reducer` y opcionalmente `combiner` (ver el fichero `word_count_mr.py` y la documentación `https://mrjob.readthedocs.io/en/latest/guides/writing-mrjobs.html` para más detalles). Una vez se tiene implementada esta clase la tarea se lanza desde el **terminal** con el comando[1]:

```
$ python word_count_mr.py texto.txt
```

Esta tarea tomará como entrada el fichero `texto.txt` y su salida se mostrará por la salida estándar. Podéis redirigir la salida estándar a un fichero, e incluso redirigir la salida de error por la que el sistema muestra distinta información de depuración:

```
$ python word_count_mr.py fichero1 fichero2 ... > salida.txt 2> error.txt
```

Apache Spark en Linux

Al igual que *mrjob*, *Spark* se encuentra instalado en los equipos del laboratorio. No obstante, si queréis usarlo en vuestro ordenador podéis instalarlo **en Linux** con el siguiente comando (es posible que en Windows también funcione):

```
$ pip install pyspark
```

Para el desarrollo de esta práctica es interesante usar el terminal interactivo `pyspark`, que nos permite crear los RDDs paso y paso y comprobar su funcionamiento. El siguiente ejemplo muestra paso a paso cómo contar las apariciones de palabras del fichero `/tmp/texto.txt`:

```
$ pyspark
Python 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
()...)


     ____              __
    / __/__ ___ _____/ /__
   _\ \/ _ \/ _ `/ __/ '_/
  /__ / .__/\_,_/_/ /_/\_\ version 3.0.1
     /_/

Using Python version 3.8.3 (default, May 19 2020 18:47:26)
SparkSession available as 'spark'.
>>> lines = sc.textFile("/tmp/texto.txt")
>>> words = lines.flatMap(lambda x: x.split())
>>> pairs = words.map(lambda x: (x,1))
>>> counts = pairs.reduceByKey(lambda x,y: x+y)
```

[1] Si el intérprete `python` adecuado no está en el *PATH* deberéis proporcionar la ruta completa o actualizar el *PATH*.

```
>>> counts.collect()
[('hola', 1), ('que', 1), ('al', 1), ('sol', 3), ('el', 2), ('pega', 2),
 ('haces', 1), ('me', 1), ('voy', 1)]
```

Una vez hayáis hecho las pruebas pertinentes en `pyspark`, debéis escribir el programa Python completo, en este caso `word_count_spark.py`. Este se lanza desde el **terminal** con el comando:

```
$ spark-submit word_count_spark.py texto.txt
```

Los parámetros detrás del nombre del programa Python, si los hay, se pasarán como argumentos del programa a través de `sys.argv`. Al igual que con *mrjob*, la salida del programa principal se mostrará por la salida estándar y los mensajes informativos y de depuración se mostrarán por la salida de error. Para esta práctica, **los programas Spark deben mostrar el resultado del cómputo por la salida estándar y no escribirlos en disco**.

1) Palabras en `sci.space` [3.5pt]

En este apartado debes realizar una tarea MapReduce **usando mrjob** para procesar colecciones de mensajes procedentes de grupos de noticias. Concretamente, queremos conocer en qué documento se repite más veces cada palabra junto con el número de repeticiones en dicho documento. Si una palabra se repite más veces en varios ficheros, se debe mostrar el fichero cuyo nombre sea lexicográficamente menor. A la hora de detectar palabras no se debe hacer distinción entre mayúsculas y minúsculas, y las palabras no deben contener signos de puntuación[2]. Es necesario que la tarea sea capaz de procesar la colección de mensajes sobre el espacio (`space.json`)[3] que se puede descargar del Campus Virtual. Este fichero contiene un documento JSON por cada línea, que representa un mensaje diferente en el foro. Cada JSON tiene dos campos: `filename` y `content`.

A cada palabra del listado resultante le debe acompañar una pareja `[fichero, #repeticiones]`, donde `fichero` es el nombre del fichero y `#repeticiones` es el número de repeticiones de dicha palabra en ese fichero. La salida debe ser un listado con un aspecto similar al que sigue:

```
...
"surely"    ["59907.txt", 1]
"surface"   ["61435.txt", 48]
"surfaces"  ["60922.txt", 2]
...
```

Nota: Se valorará el uso de un **combinador** para tratar de reducir el número de parejas emitidas.

2) Felicidad de «Los Simpsons» [3.5pt]

El conjunto de datos que podéis encontrar en `https://www.kaggle.com/prashant111/the-simpsons-dataset` contiene mucha información sobre los

[2]No hace falta que seáis excesivamente precisos en la detección de palabras, únicamente que cumpláis estas dos condiciones

[3]Extraído de `http://qwone.com/~jason/20Newsgroups/`

episodios de «Los Simpsons». En este apartado nos centraremos en el fichero `simpsons_script_lines.csv` que contiene todas las frases que aparecen en cada uno de los capítulos (diálogos, carteles, descripciones, etc.). En base a las palabras que aparecen en estos capítulos, nos gustaría obtener la «alegría acumulada» de cada uno de ellos y mostrarlos de mayor a menor.

Para obtener la «alegría acumulada» seguiremos un enfoque bastante simple: usaremos el fichero `happiness.txt` y por cada capítulo sumaremos la felicidad media (`happiness_average`) de todas sus palabras. Si alguna palabra no aparece en el fichero, no sumaremos nada. Recordad que el archivo `happiness.txt` está separado por **tabuladores**, donde cada línea tiene el siguiente formato:

```
word happiness_rank happiness_average happiness_standard_deviation twitter_rank
google_rank nyt_rank lyrics_rank
```

Antes de empezar, revisad con cuidado el fichero `simpsons_script_lines.csv` para entender bien qué hay en cada campo. El texto aparece varias veces, y en algunos campos es más fácil de procesar que en otros.

El resultado final que se debe mostrar por pantalla es un listado de identificador de episodio junto con su «felicidad acumulada», ordenados de mayor a menor felicidad. La salida debería parecerse a la siguiente *(los valores no son reales)*:

```
234 5487.21
548 5208.99
33 5102.87
...
```

Pistas:

1) Procesar las líneas del fichero `simpsons_script_lines.csv` es complejo porque contienen campos de texto entrecomillados con comas a su vez, por lo que utilizar `split()` generará una división incorrecta. Para evitar estos problemas es mejor utilizar la biblioteca `csv` de Python para que maneje estos detalles, concretamente un objeto `csv.reader` (`https://docs.python.org/3/library/csv.html#csv.reader`). Este objeto puede recibir una lista unitaria con una línea y luego se recorrerá para obtener la línea correctamente dividida en una lista (ver p. ej. `https://stackoverflow.com/a/35822856`).

2) El fichero `simpsons_script_lines.csv` es un fichero real sin limpiar, por lo que puede tener alguna línea que no cumple con el formato esperado. Para evitar que esto os dé problemas al ir transformando los RDD, recomiendo filtrar al principio para eliminar aquellas líneas mal formadas tan pronto como sea posible y dejar únicamente aquellas que tienen el número de elementos esperado.

3) Éxito de los capítulos de «Los Simpsons» [3pt]

En este apartado seguiremos con el conjunto de datos del apartado anterior pero lo extenderemos a los guiones, ubicaciones y personajes. Queremos dar respuesta a varias preguntas sobre si existe alguna relación entre la puntuación IMDB de un determinado episodio y:

1. El número de **ubicaciones diferentes** que aparecen en el episodio
2. El número de **personajes femeninos diferentes** que aparecen en el episodio

3. La cantidad total de **palabras que aparecen en los diálogos** del episodio

4. La cantidad total de **líneas de diálogo** que hay en el episodio

Tened en cuenta que en los dos últimos puntos hablamos expresamente de *diálogos*, así que habría que excluir aquellas líneas que representan carteles, descripción de eventos, etc.

Desarrollaremos un programa Spark utilizando **DataFrames**[4][5] en lugar de RDDs para resolver este problema. Los DataFrames son estructuras bidimensionales similares a tablas que tratan de acercar la programación en Spark al mundo de las bases de datos relacionales (SQL). Son secuencias de entradas, todas ellas con el mismo número y tipo de atributos. Además, la evaluación de DataFrames utiliza internamente un optimizador de consultas llamado *Catalyst*[6] que busca el plan de ejecución más eficiente.

Notas:

- Este apartado requiere que exploréis y entendáis el tipo de datos `DataFrame` disponible en Apache Spark. Para ello deberéis consultar la documentación del sistema y la bibliografía propuesta en las trasparencias.

Tareas a realizar y orden recomendado:

1. Crea un DataFrame `locations` con 3 columnas: identificador de episodio, puntuación IMDB y número de ubicaciones diferentes que aparecen en dicho episodio.

2. Genera un DataFrame `characters` con 3 columnas: identificador de episodio, puntuación IMDB y número de personajes femeninos diferentes que aparecen. Tened en cuenta que hay muchos personajes sin género consignado en nuestros datos de entrada. Estos personajes no se pueden contabilizar como femeninos ni masculinos, así que los ignoraremos.

3. Construye un DataFrame `script` con 4 columnas: identificador de episodio, puntuación IMDB, número total de palabras que aparecen en los diálogos del episodio y el número total de diálogos en el episodio. **Cuidado:** no hay que tener en cuenta las líneas del *script* que no son diálogos.

4. Finalmente, calcula el *Coeficiente de correlación de Pearson*[7] (`https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Pearson`) entre la valoración IMDB y cada una de las variables que queríamos comprobar. Puedes realizar este cálculo «*a mano*» sobre los DataFrames o utilizar funciones del API de Spark, que es mucho más sencillo y cómodo.

[4] `https://spark.apache.org/docs/latest/sql-getting-started.html`

[5] `https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html`

[6] Spark SQL: Relational Data Processing in Spark, SIGMOD'15. `http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf`

[7] Este coeficiente mide la relación lineal entre dos variables aleatorias, tomando valores en el rango $[-1.,1]$: ≈ -1 si están inversamente relacionadas, ≈ 1 si están directamente relacionadas y ≈ 0 si no existe relación lineal.