Práctica 1 - Hadoop

Francisco Javier Sáez Maldonado, José Antonio Álvarez Ocete

Parte 1

Proceso

Describiremos a continuación los pasos que se han ido siguiendo para obtener los resultados finales de la práctica.

Instalamos java

```
yum install java-1.8.0-openjdk-devel
```

Cambiamos la versión por defecto que se usaba en las transparencias Incluso si instalamos la versión 1.7 se instalará la versión 1.8. Hemos de cambiar los paths a las versión 1.8 para que funciona correctamente:

```
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk
```

Editamos el archivo `/opt/hadoop/etc/hadoop/hadoop.env.sh`:

```
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk
```

Compilación

Tomamos el código que se ha proporcionado y lo compilamos utilizando la orden

```
bin/hadoop fs -cat /user/bigdata/compilar.bash | exec bash -s WordCount
```

Ejecución

Primero hay que iniciar el NameNode y el DataNode:

```
sbin/start-dfs.sh
```

Iniciar el ResourceManager y el NodeManager:

```
sbin/start-yarn.sh
```

Recuerda que hemos de subir el archivo utilizando:

```
/opt/hadoop/bin/hdfs dfs -put Quijote.txt /user/root
```

Lanzamos nuestro trabajo de MapReduce:

```
sudo /opt/hadoop/bin/hadoop jar WordCount.jar uam.WordCount Quijote.txt output/
```

Obtenemos en el directorio output la salida. En concreto, dos archivos:

- Un archivo SUCCESS indicando que la tarea ha sido exitosa.
- Un archivo `part-r-00000` que tiene la salida del programa que queríamos ejecutar.

Mostramos una parte del fichero para mostrar parte de la salida (la salida completa se puede encontrar en este archivo).

```
"Tablante", 1
"dichosa    1
"el 8
"y  1
"¡Oh,   1
(Y  1
(a  1
(al 1
(como   1
(creyendo   1
(de 2
(habiéndose 1
(por    2
(porque 2
(pues   1
(que    21
```

Como podemos comprobar, las palabras quedan con ciertos símbolos de puntuación que no nos interesa que estén para realizar el conteo correcto de palabras.

Modificación del programa

Para arreglar esto, solo debemos cambiar una linea en la función `Map` del archivo `WordCount.java`. En concreto, la dejamos de la siguiente forma:

```
StringTokenizer itr = new StringTokenizer(value.toString().
        toLowerCase().replaceAll("[^a-z ]", ""));
```

como vemos, hemos pasado las palabras a minúsculas usando `toLowerCase` y luego hemos eliminado todo aquello que no sean letras usando `replaceAll`.

Una vez realizada esta modificación, debemos compilar de nuevo el programa como lo hemos hecho anteriormente y, seguidamente, ejecutar de nuevo el programa java para que realice el conteo de palabras. En este caso, indicamos que la salida la realice sobre la carpeta `output2/` para tener las dos salidas por separado y poder compararlas. Obtenemos los dos mismos ficheros que anteriormente.

Hecho esto, utilizamos el archivo `script.py` que hemos creado usando `Python`, que toma todas las palabras que hay en los ficheros de salida y el número de veces que aparece cada una, las ordena y toma las 10 primeras para mostrarlas por pantalla.

El resultado que se obtiene es que las palabras son las mismas y prácticamente en el mismo orden, salvo una pequeña variación entre dos de ellas. Podemos verlo gráficamente en la siguiente figura:

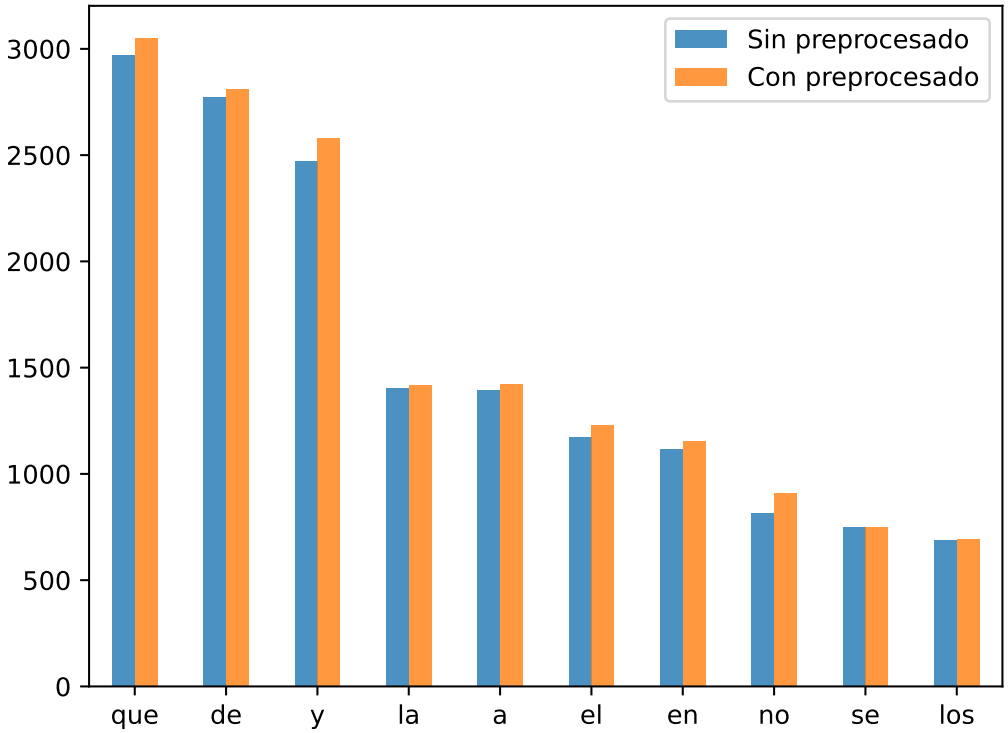

Figure 1: Comparación de palabras

Se puede apreciar que, al aplicar el preprocesado, la palabra a aparece un número de veces ligeramente superior al que lo hace cuando no tenemos el preprocesador previo del texto.

Comparación de resultados con otros ejemplos

TODO

Cuestiones planteadas

Pregunta 1.1. ¿ Qué ficheros ha modificado para activar la configuración del HDFS? ¿ Qué líneas ha sido necesario modificar?

Hemos modificado el fichero `/opt/hadoop-2.8.1/etc/hadoop/hadoop-env.sh` añadiendo la línea `export JAVA_HOME= /usr/lib/jvm/jre-1.7.0-openjdk` para especificar la instalación de `Java` que queremos utilizar

Como se explica en https://stackoverflow.com/questions/17569423/what-is-best-way-to-start-and-stop-hadoop-ecosystem-with-command-line, el script `stop-all.sh` detiene todos los daemons de Hadoop a la vez, pero está obsoleto. En lugar de eso es recomendable parar los daemons de HDFS y YARN por separado en todas las máquinas utilizando `stop-dfs.sh` y `stop-yarn.sh`.

A continuación, para instalar Hadoop pseudo-distribuido, hemos modificado el fichero `/opt/hadoop/etc/hadoop/core-site.xml`:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

Y añadimos al fichero `/opt/hadoop/etc/hadoop/hdfs-site.xml` lo siguiente:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

A continuación, hemos realizado la instalación del sistema pseudo-distribuido usando YARN, así que hemos modificado los siguientes ficheros **para configurar el uso de YARN (no de HDFS)**. `etc/hadoop/mapred-site.xml`:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Y también `etc/hadoop/yarn-site.xml`:

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>

    <property>
        <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

Ejercicio 1.2: Para pasar a la ejecución de Hadoop sin HDFS, ¿ es suficiente con parar el servicio con `stop-dfs.sh`? ¿ Cómo se consigue ?

Pregunta 3.1: ¿ Dónde se crea hdfs ? ¿ Cómo se puede decidir su localización ?

El sistema de archivos HDFS se crea donde la variable `dfs.datanode.data.dir` indique. Esta variable se puede modificar en el archivo `hdfs-site.xml`. (Su valor por defecto)[https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml] es

```
file://${hadoop.tmp.dir}/dfs/data
```

y, podemos ver que la variable `hadoo.tmp.dir` tiene el valor `/tmp/hadoop-${user.name}`.

Pregunta 3.2: ¿ Cómo se puede borrar todo el contenido del HDFS, incluido su estructura ?

Pregunta 3.3: Si estás usando hdfs, ¿ cómo puedes volver a ejecutar WordCount como si fuese single.node ?

Recordamos que hemos hecho una serie de cambios en archivos `xml` para configurar Hadood para que funcionase de forma pseudo-distribuida. Para ejecutarlo como si fuese *single-node*, deberíamos eliminar los cambios que hemos hecho en estos ficheros `xml`: `core-site.xml` y `hdfs-site.xml`. Así, volveríamos al modo por defecto de Hadoop y conseguiríamos que funcionase como si fuese `single-node`.

Pregunta 3.4: ¿ Cuáles son las 10 palabras más utilizadas ?

Esta pregunta ya ha sido mostrada anteriormente en la Figura 1. Las palabras más utilizadas son:

`que,de,y,la,a,el,en,no,se,los`

Pregunta 3.5: Cuántas veces aparecen las siguientes palabras: el, dijo

Para esta pregunta, volvemos a usar nuestro fichero `script.py` que nos imprimirá cuántas veces aparece cada una de ellas en el texto, primero usando preprocesado y a continuación sin utilizarlo.

```
Número de veces que aparece la palabra
     el
         - Sin preprocesado 1173
         - Con preprocesado 1228
     dijo
         - Sin preprocesado 196
         - Con preprocesado 271
```

Puede comprobarse ejecutando el script indicado.

Parte 2 : Tutorial de Spark

Seguimos el tutorial de spark y contestamos a las preguntas que se nos piden

Pregunta TS1.1 ¿Cómo hacer para obtener una lista de los elementos al cuadrado?

Bastará en este caso usar sobre el RDD la función map pasándole como parámetro la función `lambda x: x*x` que eleva el número al cuadrado. El código es este, con el resultado debajo:

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
cuadrados = numeros.map(lambda x : x*x)
print(cuadrados.collect())
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Pregunta TS1.2 ¿Cómo filtrar los impares?

El código se nos entrega resuelto en este caso. Es bastante sencillo, si usamos sobre el RDD la función `filter` dándole como parámetro la función que calcula si cada elemento es impar, esto es, si su módulo 2 es 1 : `lambda x : x%2 == 1`, nos da todos los impares. El código completo es:

```
rddi = numeros.filter (lambda e: e%2==1)
print (rddi.collect())
```

```
[1, 3, 5, 7, 9]
```

Pregunta TS1.3 ¿Tiene sentido esta operación? ¿Si se repite se obtiene siempre el mismo resultado?

Nos encontramos con el siguiente código:

```
#Tiene sentido esta operación?
numeros = sc.parallelize([1,2,3,4,5])

print (numeros.reduce(lambda elem1,elem2: elem1-elem2))
```

Esta operacion no tiene sentido, pues para hacer el reduce necesitamos que la operación sea conmutativa, es decir, que $a - b = b - a$, lo cual **NO** es cierto en todos los casos. Por eso además no producirá siempre los mismos resultados. Si ejecutamos en múltiples ocasiones veremos que los resultados no son los mismos siempre.

Pregunta TS1.4 ¿Cómo lo ordenarías para que primero aparezcan los impares y luego los pares?

Lo haremos para el caso general. Nos damos cuenta primero de que si el vector tiene n posiciones y tratamos de tomar $m > n$ posiciones, Spark se quedará con todas las posibles. Por tanto, lo primero que hacemos es obtener un entero con el máximo de elementos

```
numeros = sc.parallelize([3,2,1,4,5])

n = numeros.count()
```

Por último, basta ahora tomar elementos ordenados especificando cuál es el criterio de orden. En este caso, queremos que los elementos pares sean los más grandes, por lo que debemos pasarle como argumento la función `lambda x: x%2 == 0`. De esta manera, escribiendo la línea siguiente, obtenemos el resultado.

```
print(numeros.takeOrdered(n,lambda elem: elem%2== 0))

[3, 1, 5, 2, 4]
```

Pregunta TS1.5 ¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?

Nos encontramos con el siguiente código

```
lineas = sc.parallelize(['', 'a', 'a b', 'a b c'])

palabras_flat = lineas.flatMap(lambda elemento: elemento.split())
palabras_map = lineas.map(lambda elemento: elemento.split())

print (palabras_flat.collect())
print (palabras_map.collect())
```

Cuya salida es:

```
['a', 'a', 'b', 'a', 'b', 'c']
[[], ['a'], ['a', 'b'], ['a', 'b', 'c']]
```

Para ver cuántos elementos tiene cada uno, podemos hacer

```
print(palabras_flat.count())
print(palabras_map.count())

6
4
```

Claramente, `palabras_flat` tiene más. Esto ocurre porque en `flatMap`, cada elemento de entrada al que se le aplica la función `lambda`, puede tener como salida 0 o más (un vector) de elementos. Por tanto:

- Con `flatMap` para cada elemento de `lineas` obtenemos un vector que luego separaremos por elementos e ignoraremos los elementos que estén vacíos. Por ello, de 4 elementos iniciales pasamos a 6 que es el total de letras que tenemos en el RDD.
- Con `map`, a cada uno de los vectores iniciales se le aplica la función indicada y se introduce **tal cual** en el RDD resultante, no se separan los elementos de los vectores obtenidos como ocurría en el caso anterior.

Pregunta TS1.6 ¿De qué tipo son los elementos del rdd palabras_map? ¿Por qué palabras_map tiene el primer elemento vacío?

Siguiendo la respuesta de la pregunta anterior, en `palabras_map` los elementos obtenidos tras aplicar a cada elemento de `líneas` la función, son vectores, por lo que los elementos de `palabras_map` son vectores.

Además, tiene el primer **elemento vacío** porque la función split aplicada sobre el elemento `''` devuelve un vector vacío pues no hay nada que separar.