2021

# Practica 1

Hadoop y Spark

ADRIAN RUBIO PINTADO Y CARLOS RAMOS MATEOS

# Contenido

# Parte 1

## Ejercicio 1.1: ¿Qué ficheros ha modificado para activar la configuración del HDFS? ¿Qué líneas ha sido necesario modificar?

El archivo core-site.xml informa al demonio de Hadoop dónde se ejecuta NameNode en el clúster. Contiene los ajustes de configuración para Hadoop Core, como los ajustes de E/S que son comunes a HDFS y MapReduce.

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

El archivo hdfs-site.xml contiene los valores de configuración para los demonios HDFS; el NameNode, el NameNode secundario y los DataNodes. Aquí, podemos configurar hdfs-site.xml para especificar la replicación de bloques predeterminada y la verificación de permisos en HDFS. El número real de réplicas también se puede especificar cuando se crea el archivo. Se utiliza el valor predeterminado si no se especifica la replicación en el momento de creación.

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

La función de stop-dfs.sh lo que hace es parar el funcionamiento del demonio encargado de los HDFS, por lo que es el primer paso necesario para pasar de una ejecución pseudo-distribuida con HDFS a una Standalone. En el caso de que este fuera el único paso para seguir, ahora podríamos lanzar un comando de ejecución de .jar como los vistos en el tutorial de instalación. Sin embargo, al lanzar esta operación vemos que en la terminal aparece un mensaje de error:

21/09/19 13:31:18 WARN ipc.Client: Failed to connect to server: localhost/127.0.0.1:9000: try once and fail.

java.net.ConnectException: Conexión rehusada

Esto se debe a que en la configuración del core-site.xml aún sigue estando configurada de forma pseudo-distribuida, por lo que, eliminando la propiedad incluida en la configuración, podemos a volver a pasar la ejecución a Hadoop sin HDFS, una forma para hacerlo sería comentando las líneas tal como aparece en el ejemplo inferior.

```
<!-- Put site-specific property overrides in this file. -->

<configuration>

<!--    <property>

            <name>fs.defaultFS</name>

            <value>hdfs://localhost:9000</value>

        </property>-->

</configuration>
```

# Ejercicio 3

## Ejercicio 3.1: ¿Dónde se crea hdfs? ¿Cómo se puede elegir su localización?

El sitio físico donde se aloja el hdfs se encuentra en la siguiente ruta por defecto /tmp/hadoop-${user.name}/dfs/data . En nuestro caso al haberlo creado con el usuario bigdata, es: /tmp/hadoop-bigdata/dfs/data

Esto se puede modificar mediante las propiedades de dfs.datanode.data.dir (para las carpetas de datos) y hadoop.tmp.dir(para los temporales) pertenecientes a los ficheros de configuración hdfs-default.xml y core-default.xml respectivamente.

```
<!--    <property>

            <name>dfs.datanode.data.dir </name>

            <value>path </value>

        </property>-->


<!--    <property>

            <name> hadoop.tmp.dir </name>

            <value>path </value>

        </property>-->
```

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

Para formatear todo el contenido del hdfs deberemos usar el siguiente comando: bin/hdfs namenode –format.

El comando tiene esta forma ya que el namenode es lugar centralizado de un sistema de archivos HDFS que mantiene el árbol de directorios de todos los archivos en el sistema de archivos y rastrea dónde se guardan los datos del archivo en todo el clúster.

Cuando formateamos namenode, formatea los metadatos relacionados con los nodos de datos. Al hacer eso, toda la información de los nodos de datos se pierde y se pueden reutilizar para obtener nuevos datos.

### Ejercicio 3.3: Si estás utilizando hdfs ¿Cómo puedes volver a ejecutar WordCount como si fuese single.node?

*Tras hablar con el profesor, se aclaró que se refería al paso de pseudo-distributed a standalone que ya fue respondido en el ejercicio 1.*

### Ejercicio 3.4: ¿Cuál son las 10 palabras más utilizadas?

Este es el resultado generado en la salida del programa WordCount, tras haber ordenador el par palabra - número de repeticiones del fichero generado mediante un archivo de Excel.

| | |
|---|---|
| que | 3054 |
| de | 2811 |
| y | 2579 |
| a | 1427 |
| la | 1423 |
| el | 1232 |
| en | 1155 |
| no | 906 |
| se | 753 |
| los | 696 |

### Ejercicio 3.5: ¿Cuántas veces aparece:

• El articulo “el”?

Aparece un total de 1232 veces.

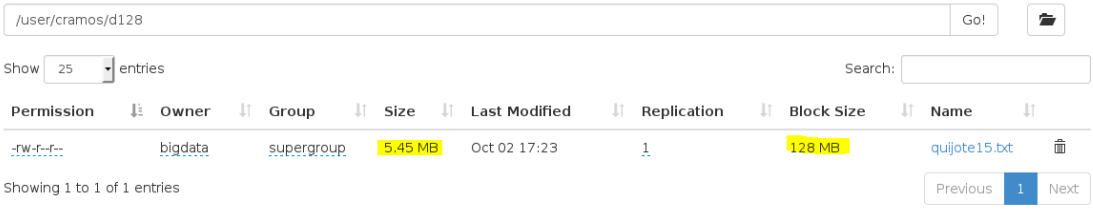
• La palabra “dijo”?

Aparece un total de 272 veces.

### Ejercicio 3.6: El resultado coincide utilizando la aplicación wordcount que se da en los ejemplos. Justifique la respuesta.

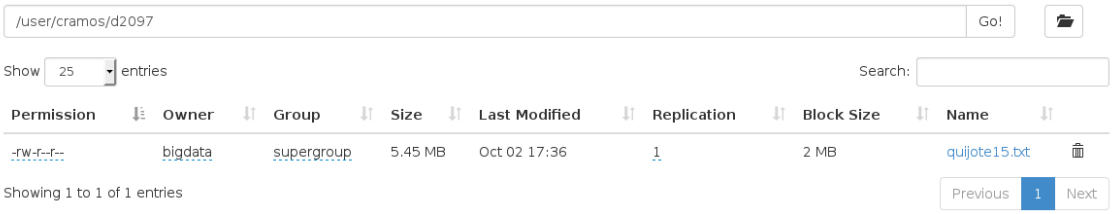
No coincide, esto se debe a que la sentencia de los ejemplos no se contempla la modificación que hemos realizado posteriormente para el ejercicio, por lo que no es capaz de parsear adecuadamente las palabras al no ser capaz de distinguir entre mayúsculas/minúsculas e incapaz de ignorar caracteres especiales como los signos de puntuación.

### Ejercicio 4.1: Usar el tamaño de bloque por defecto de HDFS (128 MB en Hadoop2.8)

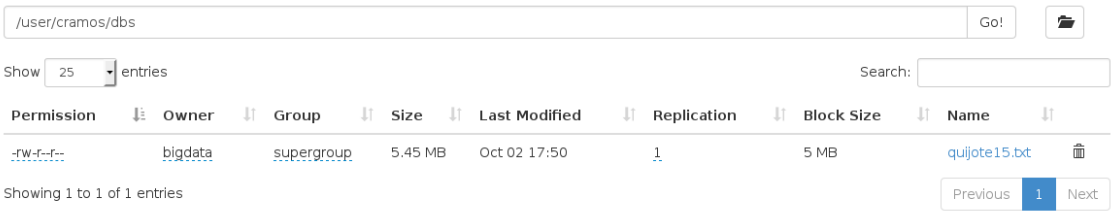



## Ejercicio 4.2: Indicar el tamaño de bloque en la línea de comandos al escribir el fichero.

sudo /opt/hadoop/bin/hdfs dfs -D dfs.blocksize=2097152 -put /home/bigdata/quijotex9.txt /user/bigdata




## Ejercicio 4.3: Editar el fichero de configuración hdfs-site.xml y modificar el tamaño de bloque con el parámetro dfs.block.size




## Ejercicio 4.4: Comprobar el efecto del tamaño de bloques en el funcionamiento de la aplicación WordCount. ¿Cuántos procesos Maps se lanzan en cada caso? Indique como lo ha comprobado.

En cada una de estas ejecuciones hemos recibido una salida por terminal la cual explica los estados por los que pasa los Jobs map/reduce y finalmente saca un resumen de la ejecución:

File System Counters

FILE: Number of bytes read=23256098

FILE: Number of bytes written=35520132

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=11434248

HDFS: Number of bytes written=87722

HDFS: Number of read operations=13

HDFS: Number of large read operations=0

HDFS: Number of write operations=4

Aquí se habla acerca de los ficheros de entrada y salida, como se puede ver al haber realizado la lectura y escritura sobre HDFS también se indica el tamaño y las lecturas realizadas sobre estos.

```
Map-Reduce Framework
        Map input records=99612
        Map output records=1017324
        Map output bytes=9589968
        Map output materialized bytes=11624622
        Input split bytes=117
        Combine input records=0
        Combine output records=0
        Reduce input groups=7542
        Reduce shuffle bytes=11624622
        Reduce input records=1017324
        Reduce output records=7542
        Spilled Records=2034648
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=1370
        Total committed heap usage (bytes)=838860800
```

Este es a nuestro parecer el fragmento de resumen que nos resulta más relevante para esta práctica, aquí podemos ver los tamaños leídos en los mapas, así como su el número de mapas y el número de “barajeos” que se han hecho sobre los mapas. También se pueden ver lo leído por las operaciones reduce y el número necesarios de uniones para unificar la salida en un resultado único (tal como hace el reduce).

```
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=5717124
File Output Format Counters
        Bytes Written=87722
```

Por último, tenemos una salida que nos indica las conclusiones de la ejecución (hubo algún error/excepción al hacer el mapa o en las operaciones reduce, …) De esta salida de ejemplo se puede ver que claramente no hubo ningún tipo de problema.

Por lo que, volviendo a la pregunta inicial, hemos deducido que si observamos el número de Merged Map outputs que se han realizado, podemos inducir el número de mapas que se han elaborado por lo que revisamos los resultados de las tres salidas respectivamente.

- **4.1 Shuffled Maps =1 & Merged Map outputs=1**: Aquí podemos ver que para un tamaño de blocksize de 128 realmente tan solo necesitamos un único mapa, ya que 128Mb > 5Mb, que es lo que ocupa nuestro fichero.
- **4.2: Shuffled Maps =3 & Merged Map outputs=3:** Para este documento tenemos un tamaño de blocksize aproximado a 2Mb, por lo tanto tiene sentido que sean necesarios tres mapas para tratar el fichero de entrada, ya que 3 * 2Mb = 6Mb > 5Mb,si tan solo utilizaramos dos mapa s nos estaríamso quedando cortos a la hora de tratar todo el fichero ya que 2*2Mb = 4Mb < 5Mb .
- **4.3 Shuffled Maps =1 & Merged Map outputs=1:** Inicialmente había pensado que iban a ser necesarios 2 mapas, ya que nuestro fichero excedía pro poco los 5Mb, al final hemos concluido que el tamaño de BlockSize que hemos introducido no es exactamente de 5Mb, pero que en la interfaz web se nos está mostrando un redondeo a las unidades. Lo suficiente como para poder trabajar con un único bloque.

# Parte 2

## Ejercicio 2. Pregunta - TS1.1 ¿Cómo hacer para obtener una lista de los elementos al cuadrado?

Mapeamos una lambda que eleve al cuadrado:

```
[ ]  #Mapeamos un lambda
     pnumeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])

     rdd = pnumeros.map(lambda e: e**2)

     print(rdd.collect())
```

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

## Ejercicio 2. Pregunta - TS1.2 ¿Cómo filtrar los impares?

Usamos filter con una lambda que compruebe que el resto es 1:

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

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

## Ejercicio 2. Pregunta - TS1.3 ¿Tiene sentido esta operación? ¿Si se repite se obtiene siempre el mismo resultado?

No, no tiene sentido ya que la función pasada a reduce debe de ser asociativa y conmutativa. En este caso es una resta, que no tiene ninguna de las 2 propiedades. Dado que en reduce el orden no está definido, al paralelizar, cada vez que hacemos la operación obtenemos resultados distintos:

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

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

```
5
```

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

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

```
3
```

## Ejercicio 2. Pregunta - TS1.4 ¿Cómo lo ordenarías para que primero aparezcan los impares y luego los pares?

Cogemos por un lado los números impares y por otro los pares ordenando los números (no se pide que impares y pares estén ordenados entre ellos). Luego hacemos la unión de ambos.

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
pares = numeros.filter(lambda e: e%2==0)
impares = numeros.filter(lambda e: e%2==1)
nums_ordenados_impar_par = impares.union(pares)

nums_ordenados_impar_par.collect()
```

```
[1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
```

## Ejercicio 2. Pregunta - TS1.5 ¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?

```
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())
```

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

El primero tiene 6 elementos, el segundo 4 (ya que estamos aplicando lamba a cada elemento y cada elemento produce una salida). En el primero aplicamos flatmap, donde cada elemento puede producir más de una salida, por eso el primer rdd tiene más elementos.

## Ejercicio 2. Pregunta - TS1.6 ¿De qué tipo son los elementos del rdd palabras_map? ¿Por qué palabras_map tiene el primer elemento vacío?

Son listas. Porque al aplicar la función split sobre un string vacío no obtenemos ningún elemeno(es decir, una lista vacía).

## Ejercicio 2. Pregunta - TS1.7 Prueba la transformación distinct si lo aplicamos a cadenas.

Vemos cómo funciona también para cadenas:

```
rep_info = sc.parallelize(['a', 'A','b', 'b', 'c', 'd','e','f','f'])
rdd_distinct_info = rep_info.distinct()
print(rdd_distinct_info.collect())
```

```
['b', 'c', 'd', 'a', 'A', 'e', 'f']
```

## Ejercicio 2. Pregunta - TS1.8 ¿Cómo se podría obtener la misma salida pero utilizando una sola transformación y sin realizar la unión?

Añadiendo un “or” a la lambda por ejemplo:

```
log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13', 'E: e45'])

infos_and_errors = log.filter(lambda elemento: elemento[0]=='I' or elemento[0]=='E' )

print (infos_and_errors.collect())
```

```
['E: e21', 'I: i11', 'I: i11', 'E: e45']
```

## Ejercicio 2. Pregunta - TS1.9 ¿Cómo explica el funcionamiento de las celdas anteriores?

**¿Tiene sentido esta transformación?**

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

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

    5
```

No, la resta no es asociativa ni conmutativa para meterla en reduce.

**¿y esta tiene sentido esta operación? ¿Qué pasa si ponemos elem2+"-"+elem1?**

```
hola-que-tal-bien
```

Si queremos que siempre de la misma salida: "hola-que-tal-bien" no, ya que esa lambda no es conmutativa.

```
print (pal_minus.reduce(lambda elem1,elem2: elem2+"-"+elem1))
```

```
bien-tal-que-hola
```

Ejecutando de este modo obtenemos el orden inverso a la ejecución anterior.

**Dos últimas celdas:**

```
r = sc.parallelize([('A', 1),('C', 4),('A', 1),('B', 1),('B', 4)])
rr = r.reduceByKey(lambda v1,v2:v1+v2)
print (rr.collect())
```

```
[('C', 4), ('A', 2), ('B', 5)]
```

```
r = sc.parallelize([('A', 1),('C', 4),('A', 1),('B', 1),('B', 4)])
rr1 = r.reduceByKey(lambda v1,v2:v1+v2)
print (rr1.collect())
rr2 = rr1.reduceByKey(lambda v1,v2:v1)
print (rr2.collect())
```

```
[('C', 4), ('A', 2), ('B', 5)]
[('C', 4), ('A', 2), ('B', 5)]
```

En este caso estamos haciendo un reduceByKey donde juntamos cada elemento por el valor de las claves especificamos en cada lambda que sume los valores correspondientes si dos claves son iguales.

En al última casilla, rr2 no hace reduceByKey de r (como rr1), sí que hace reduceByKey de rr1, donde todos los elementos tienen clave única, por lo que realmente no está aplicando nada y obtenemos la misma salida.

## Ejercicio 2. Pregunta - TS1.10 Borra la salida y cambia las particiones en parallelize ¿Qué sucede?

```
%ls -la salida/*
```

```
-rw-r--r-- 1 root root 390 Oct  2 21:56 salida/part-00000
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00001
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00002
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00003
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00004
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00005
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00006
-rw-r--r-- 1 root root 500 Oct  2 21:56 salida/part-00007
-rw-r--r-- 1 root root   0 Oct  2 21:56 salida/_SUCCESS
```

Cambiando a 8 particiones, por ejemplo, Que en salida obtenemos 8 particiones del RDD, desde part-00000 a part-00007 y ahora part-00000 solo contiene los primeros 125 números (hasta 124).

## Ejercicio 2. Pregunta - TS2.1 Explica la utilidad de cada transformación y detalle para cada una de ellas si cambia el número de elementos en el RDD resultante. Es decir si el RDD de partida tiene N elementos, y el de salida M elementos, indica si N>M, N=M o N<M.

- map: N=M Devuelve un nuevo rdd pasándole una función func a cada elemento del dataset origen
- flatmap: N<=M lo mismo que map pero cada entrada puede ser mapeada a varias salidas(tambien conocido como secuencia).
- filter: N>=M Devuelve un rdd con aquellos elementos del dataset cuya función pasada quede evaluada como verdadero.
- distinct: N>=M Devuelve un rdd cuyos elementos son unicos y no repetidos
- sample: N>=M Devuelve una fracción de los elementos del rdd origen. Con o sin remplazamiento y de manera aleatoria. Es posible pasarle una semilla.
- union: N<=M Devuelve un rdd con la union del dataset origen con el pasado por argumento.

## Ejercicio 2. Pregunta - TS2.2 Explica el funcionamiento de cada acción anterior

```
[ ]  numLines = quijote.count()
     numChars = charsPerLine.reduce(lambda a,b: a+b) # also charsPerLine.sum()
     sortedWordsByLength = allWordsNoArticles.takeOrdered(20, key=lambda x: -len(x))
     numLines, numChars, sortedWordsByLength
```

- En la primera línea contamos el número de líneas que tiene el quijote con una acción count, ya que cada elemento en ese rdd corresponde a una línea. Contando el número de elementos obtenemos el número de líneas.
- numChars: para calcular el número de caracteres, totales utilizamos el rdd anterior charsperline y le aplicamos una lambda para que nos cuente todos los conteos de cada línea individual para obtener el total de caracteres
- sortedWordsByLength: De nuevo utilizando el rdd anterior allWordsNoArticles, donde tenemos todas las palabras, y ordenamos por claves (las palabras en sí), de mayor a menor según la longitud de las palabras. Es decir, obtenemos las 20 palabras más largas.

## Ejercicio 2. Pregunta - TS2.3 Explica el propósito de cada una de las operaciones anteriores

```
import requests
import re
allWords = allWords.flatMap(lambda w: re.sub("""[;|:|\.|,|-|-|"|'|\s"""," ", w.lower()).split(" ")).filter(lambda a: len(a)>0)
allWords2 = sc.parallelize(requests.get("https://gist.githubusercontent.com/jsdario/9d871ed773c81bf217f57d1db2d2503f/raw/585de69b0631c805dabc6280506717943b82ba4a/el_quijote_ii.txt").iter_lines())
allWords2 = allWords2.flatMap(lambda w: re.sub("""[;|:|\.|,|-|-|"|'|\s"""," ", w.decode("utf8").lower()).split(" ")).filter(lambda a: len(a)>0)
```

allWords y allWords2 representan todas las palabras del quijote. allWords se ha obtenido del fichero local, allWords2 de un url con el archivo en remoto. Para obtener todas las palabras se ha aplicado un flatmap a cada línea, cuya función lambda consiste en una expresión regular que quita los espacios para ver los conjunto de "aparentes palabras" y luego filtrarlas para quitar los símbolos y signos de puntuación.

```
allWords.take(10)
```

```
['el',
 'ingenioso',
 'hidalgo',
 'don',
 'quijote',
 'de',
 'la',
 'mancha',
 'miguel',
 'de']
```

Tomamos 10 palabras del primer conjunto

```
allWords2.take(10)
```

```
['don',
 'quijote',
 'de',
 'la',
 'mancha',
 'miguel',
 'de',
 'cervantes',
 'saavedra',
 'segunda']
```

Tomamos 10 palabras del segundo conjunto

```
[ ]  words = allWords.map(lambda e: (e,1))
     words2 = allWords2.map(lambda e: (e,1))

     words.take(10)
```

Cada palabra la convertirmos(en otro rdd) en un diccionario tal que (clave=palabra,valor=1)

```
frequencies = words.reduceByKey(lambda a,b: a+b)
frequencies2 = words2.reduceByKey(lambda a,b: a+b)
frequencies.takeOrdered(10, key=lambda a: -a[1])
```

Ya que tenemos diccionarios con claves repetidas, siguiendo el modelo map reduce, hacemos un reducebykey para obtener la frecuencia de cada palabra.

```
[ ]  res = words.groupByKey().takeOrdered(10, key=lambda a: -len(a))
     res # To see the content, res[i][1].data
     #for i in range(0,10):
     #  print(res[i][1].data)

     # el: print(sum(res[0][1].data)) da 1232 como abajo
```

```
1232
```

```
[('el', <pyspark.resultiterable.ResultIterable at 0x7ff36477d0d0>),
 ('hidalgo', <pyspark.resultiterable.ResultIterable at 0x7ff36477d2d0>),
 ('don', <pyspark.resultiterable.ResultIterable at 0x7ff36477d350>),
 ('mancha', <pyspark.resultiterable.ResultIterable at 0x7ff36477d1d0>),
 ('saavedra', <pyspark.resultiterable.ResultIterable at 0x7ff36477d090>),
 ('que', <pyspark.resultiterable.ResultIterable at 0x7ff36477dbd0>),
 ('condición', <pyspark.resultiterable.ResultIterable at 0x7ff36477dc10>),
 ('y', <pyspark.resultiterable.ResultIterable at 0x7ff36477de10>),
 ('del', <pyspark.resultiterable.ResultIterable at 0x7ff36477db10>),
 ('d', <pyspark.resultiterable.ResultIterable at 0x7ff36477dd50>)]
```

Agrupamos las palabras por clave para obtener la frecuencia y cogemos las 10 con más frecuencia

```
[ ]  joinFreq = frequencies.join(frequencies2)
     joinFreq.take(10)
```

Hacemos un join de las frecuencias obtenidas por cada método descrito arriba y tomamos 10 ejemplos (join da el siguiente formato de salida (K, (V, W))) siendo K la clave de unión y V y W valores de las tablas

```
joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1]))).takeOrdered(10, lambda v: -v[1]), joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1]))).takeOrdered(10, lambda v: +v[1])
```

```
([('pieza', 0.8),
  ('corral', 0.8),
  ('rodela', 0.7777777777777778),
  ('curar', 0.75),
  ('valle', 0.75),
  ('entierro', 0.75),
  ('oh', 0.7142857142857143),
  ('licor', 0.7142857142857143),
  ('difunto', 0.7142857142857143),
  ('pago', 0.6666666666666666)],
 [('teresa', -0.9767441860465116),
  ('roque', -0.96),
  ('paje', -0.9565217391304348),
  ('duque', -0.9565217391304348),
  ('blanca', -0.9565217391304348),
  ('gobernador', -0.9503105590062112),
  ('diego', -0.9459459459459459),
  ('tarde', -0.9428571428571428),
  ('mesmo', -0.9381443298969072),
  ('letras', -0.9354838709677419)])
```

Para cada uno de los métodos frecuencies y frecuencies2 calcula el cociente entre la diferencia de ambos resultados y la suma de ambos resultados. Por ejemplo "pieza" en frecuencies nos da "9" y en frecuencies 2 da "1" por lo que 8/10 = 0.8 Todos los resultados los da ordenados de mayor a menor (tomando solo los 10 primero), esto es, el resultado de esta operación.

Repetimos el resultado solo que ahora con resultados negativos.

Caso particular de “pieza”:

```
[ ]  test = frequencies.filter(lambda e: e[0] == 'pieza')
     t = test.collect()

     print(t[0][1])
```

```
9
```

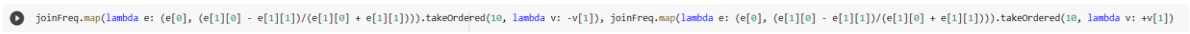
```
     test2 = frequencies2.filter(lambda e: e[0] == 'pieza')
     test2.collect()
```

```
[('pieza', 1)]
```

## Ejercicio 2. Pregunta - TS2.4 ¿Cómo puede implementarse la frecuencia con groupByKey y transformaciones?

Como hemos realizado en los apartados anteriores. Creamos un conjunto clave valor (K=palabra,1) para cada palabra, y hacemos un groupbykey. Finalmente, con la secuencia de groupbykey(de 1's) calculamos su longitud obteniendo el resultado(frecuencia).

## Ejercicio 2. Pregunta - TS2.5 ¿Cuál de las dos siguientes celdas es más eficiente? Justifique la respuesta.




O

```
result = joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1][1])))
result.cache()
result.takeOrdered(10, lambda v: -v[1]), result.takeOrdered(10, lambda v: +v[1])
```

La segunda. Ya que la primera calcula la lambda sobre el rdd para obtener result 2 veces. En la segunda cacheamos el resultado para queries que requieren de la misma transformación intermedia, haciendo más rápido la ejecución (y no teniendo que recorrer dos veces el rdd innecesariamente).

## Ejercicio 2. Pregunta - TS2.6 Antes de guardar el fichero, utilice coalesce con diferentes valores ¿Cuál es la diferencia?

Que reduce el número de particiones del rdd cuando lo guarda. Lo podemos ver haciendo un ls al directorio (ver código). Cabe destacar que como el número de particiones es originalmente 2, si ponemos un 1 lo reduce a 1 partición, 2 lo mantiene, y valores superiores lo mantiene también (puesto que solo reduce el número de particiones). Lógicamente el valor 0 no es aceptado.

Con 1 partición:

```
!rm -rf palabras_parte2
#allWords2.saveAsTextFile("palabras_parte2")

allWords2.coalesce(numPartitions=1).saveAsTextFile("palabras_parte2")
```

```
%ls -la /content/palabras_parte2/

total 1096
drwxr-xr-x 2 root root    4096 Oct  5 08:00 ./
drwxr-xr-x 1 root root    4096 Oct  5 08:00 ../
-rw-r--r-- 1 root root 1097115 Oct  5 08:00 part-00000
-rw-r--r-- 1 root root    8580 Oct  5 08:00 .part-00000.crc
-rw-r--r-- 1 root root       0 Oct  5 08:00 _SUCCESS
-rw-r--r-- 1 root root       8 Oct  5 08:00 ._SUCCESS.crc
```

Con 2 particiones:

```
[118] !rm -rf palabras_parte2
      #allWords2.saveAsTextFile("palabras_parte2")

      allWords2.coalesce(numPartitions=2).saveAsTextFile("palabras_parte2")
```

```
%ls -la /content/palabras_parte2/

total 1100
drwxr-xr-x 2 root root   4096 Oct  5 08:00 ./
drwxr-xr-x 1 root root   4096 Oct  5 08:00 ../
-rw-r--r-- 1 root root 666813 Oct  5 08:00 part-00000
-rw-r--r-- 1 root root   5220 Oct  5 08:00 .part-00000.crc
-rw-r--r-- 1 root root 430302 Oct  5 08:00 part-00001
-rw-r--r-- 1 root root   3372 Oct  5 08:00 .part-00001.crc
-rw-r--r-- 1 root root      0 Oct  5 08:00 _SUCCESS
-rw-r--r-- 1 root root      8 Oct  5 08:00 ._SUCCESS.crc
```

Con un número de superior (no funciona ya que solo reduce y no sube de las 2 particiones):

```
!rm -rf palabras_parte2
#allWords2.saveAsTextFile("palabras_parte2")

allWords2.coalesce(numPartitions=6).saveAsTextFile("palabras_parte2")
```

```
%ls -la /content/palabras_parte2/

total 1100
drwxr-xr-x 2 root root   4096 Oct  5 08:01 ./
drwxr-xr-x 1 root root   4096 Oct  5 08:01 ../
-rw-r--r-- 1 root root 666813 Oct  5 08:01 part-00000
-rw-r--r-- 1 root root   5220 Oct  5 08:01 .part-00000.crc
-rw-r--r-- 1 root root 430302 Oct  5 08:01 part-00001
-rw-r--r-- 1 root root   3372 Oct  5 08:01 .part-00001.crc
-rw-r--r-- 1 root root      0 Oct  5 08:01 _SUCCESS
-rw-r--r-- 1 root root      8 Oct  5 08:01 ._SUCCESS.crc
```

# Parte 3

Para el ejercicio opcional hemos optado por la realización del primer ejercicio propuesto. Este consiste en la elaboración de un tutorial siguiendo el paradigma MapReduce visto en clase utilizando esta vez un proyecto Apache Maven.

El dataset de estudio va a ser un fichero csv que recoge un histórico de la calidad del aire en Castilla León, que ha sido descargado desde la web de datos abiertos: http://datosabiertos.jcyl.es de esta comunidad.

La estructura del dataset se almacena en un único fichero con la siguiente estructura:

| Fecha | CO (mg/m3) | NO (ug/m3) | NO2 (ug/m3) | O3 (ug/m3) | PM10 (ug/m3) | PM25 (ug/m3) | SO2 (ug/m3) | Provincia | Estación | Latitud | Longitud |
|---|---|---|---|---|---|---|---|---|---|---|---|

Para el problema propuesto sin embargo tan solo tiene interés las columnas de provincia y CO ya que el ejercicio propuesto consiste en una estimación de la calidad del aire con mediciones desde el 1997 hasta el 2013.

Con este dataset, vamos a calcular la media de la concentración de CO agrupando por provincia, con el objetivo de determinar cuál de estas tiene mejor calidad de aire en dicha franja de tiempo.

Para ello vamos a seguir la filosofía Map/Reduce vista en clase. Es decir, tendremos primero una clase Mapper de nombre AirQualityMapper que definirá la entrada clave/valor, en nuestro caso, este recorrerá todas las líneas del fichero, las cuales parsearemos fácilmente con ";" ya que recordemos que el dataset está en formato ".csv." Finalmente almacenaremos el par clave/valor que será el nombre de la provincia y su valor de concentración de CO respectivamente.

La otra clase que tendremos será el Reducer, la cual tomará el diccionario creado por la clase previa (en uno o varios mapas dependiendo de la configuración del hadoop) y realizará un sumatorio sobre el listado de valores (concentración) para cada clave(provincia) de nuestro diccionario.

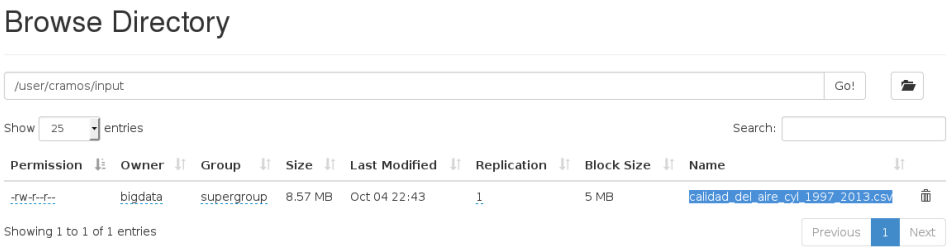
De manera adicional se define una clase que en el artículo se define como driver, la cual actuará como main del proyecto. En este fragmento de código se realiza el control de parámetros, las llamadas correspondientes que habilitan la invocación del método por terminal y la invocación del Job para las clases Mapper y Reducer para unos argumentos de entrada y salida.

Para subir los datos hemos utilizado la distribución hdfs de manera similar a lo hecho en la primera parte de la practica:

```
bin/hdfs dfs -mkdir /user/cramos/input

bin/hdfs dfs -put /home/bigdata/Escritorio/calidad_del_aire_cyl_1997_2013.csv
/user/cramos/input
```

Debido a como dejamos los ficheros de configuración hdfs para apartados anteriores, el blocksize del archivo será de 5Mb y una replicación tal como se ve en la imagen inferior.




Consideramos que esta aplicación es buena, ya que sigue una función de escalabilidad y funcionalidad similar al WordCount visto en clase, por un lado, tendremos el mapper que genera diferentes listados de concentración de CO, y el reducer que se encargará de agrupar las salidas de los diferentes listados y calcular una media total.

Sospechamos que la principal limitación de esta solución se encontrará en la capacidad de almacenamiento, ya que los resultados de cada iteración requieren ser almacenados en disco, primero como listado de valores y luego como una media. Además de esto, otra limitación del programa puede ser el tiempo de ejecución provocado por la lectura/escritura de estos pasos intermedios, por lo cual una mejor solución podría ser una solución Spark que fuera capaz de almacenar estos estados intermedios en memoria en vez de en disco.

Como conclusión, excluyendo al propio resultado del problema, hemos visto como un esquema de solución bastante similar al WordCount es fácilmente extrapolable a un problema con un dataset real, como es en nuestro caso un análisis de calidades de aire.